Attacks on Port Knocking Authentication Mechanism

Conference Paper *in* Lecture Notes in Computer Science · May 2005

DOI: 10.1007/11424925_134 · Source: DBLP

4 authors, including:

Joaquín Torres
Complutense University of Madrid
24 PUBLICATIONS **200** CITATIONS
SEE PROFILE

Juan Tapiador
University Carlos III de Madrid
187 PUBLICATIONS **5,592** CITATIONS
SEE PROFILE

Julio Hernandez-Castro
Universidad Politécnica de Madrid
266 PUBLICATIONS **4,732** CITATIONS
SEE PROFILE

Attacks on Port Knocking Authentication Mechanism

Antonio Izquierdo Manzanares, Joaquín Torres Márquez,
Juan M. Estevez-Tapiador, and Julio César Hernández Castro

Universidad Carlos III de Madrid, Avda de la Universidad 30,
28911 Leganés (Madrid), Spain
{aizquier, jtmarque, jestevez, jcesar}@inf.uc3m.es

Abstract. Research in authentication mechanisms has led to the design and development of new schemes. The security provided by these procedures must be reviewed and analyzed before they can be widely used. In this paper, we analyze some weaknesses of the port knocking authentication method that makes it vulnerable to many attacks. We will present the NAT-Knocking attack, in which an unauthorized used can gain access to the protected server just by being in the same network than an authorized user. We will also discuss the DoS-Knocking attack, which could lead to service disruptions due to attackers "knocking" on many ports of the protected server. Finally, we will review further implementation issues.

1 Introduction

Authentication has been an issue in communications security, as it is the mechanism that allows principals to identify each other involving some kind of operations prior to communication with each other. As we see, authentication is the "access point" to the communication, so it is a potential target for an attacker. This means the whole authentication process must be secured using protocols and mechanisms that allow each principal to verify each other's identity. Traditionally it has been said that authorization should rely on something the principal knows (such as a password or a pass phrase [10]), something the principal is (such as biometric values [1]), and/or something the principal has (e.g. a smart card [1]).

Port knocking [4] authorization mechanism relies on something the principal knows. In this case, the password is not a character sequence but a port sequence: The server we are willing to communicate keeps all of its network ports closed and these ports must be "knocked" in the correct sequence in order for the server to open the desired communication port. The procedure for "knocking" a port consists on sending a packet to that port, so the server will notice a connection attempt against a closed port and log it.

The main advantage of port knocking is that, as knocked ports are closed during the authentication procedure, it should not be possible to identify whether a server is using port knocking to authenticate the clients [6], resulting in a

stealth mechanism. However, this stealth property has been attacked in [5] as well as the "security through obscurity" that seems to cover the whole port knocking proposal.

In this paper we describe several further security issues of port knocking and we will discuss whether port knocking really provides some advantages over traditional methods.

1.1 Port Knocking

Port knocking, as described in [4] is an authentication method that provides network level access to services using a password composed of several ports. The server keeps all of its ports closed (e.g. using a firewall), and the clients transmit the port-sequence making connections to such ports. This is called "knocking" a port.

The server logs all these connection attempts, and an external process parses the logs and looks for correct sequences. When this process finds a valid sequence it updates the connection policies allowing communications from the client to the desired service. In order to achieve this the client's IP and the port to open must be encoded in the port sequence.

Figure 1 shows an example of the whole process where the client computes the sequence to use a service (SSH in this example) based on its IP address (*500, 850, 771,..., 1840*). After that, the client starts making connections to those ports, in order to generate entries in the server's log. A process in the server is parsing that log, and when it detects a valid sequence it computes the service port (port 22 in this example) and the client's address, and modifies the connection policies in order to open the requested port for that client. Consequently, from the client's point of view, the ports it has to "knock" can be calculated as:

$$p_1 = f_1\left(port\_to\_open, client\_address, \ldots\right), \tag{1}$$

$$p_2 = f_2\left(port\_to\_open, client\_address, \ldots\right), \tag{2}$$

etc..., where p_1 is the first port used in the knocking sequence (500 in our example), p_2 is the second one, etc... From the server's point of view, all the parameters the port knocking server has to take into account when modifying the firewall policies can be expressed as:

$$opened\_port = f_3\left(p_1, p_2, \ldots, p_n\right), \tag{3}$$

$$IP\_allowed = f_4\left(p_1, p_2, \ldots, p_n\right), \tag{4}$$

and so on.

2 NAT-Knocking: The Problem of Sharing Network Addresses

As stated previously, port knocking is based on the idea of opening ports in the firewall just to clients that have provided the correct password through the

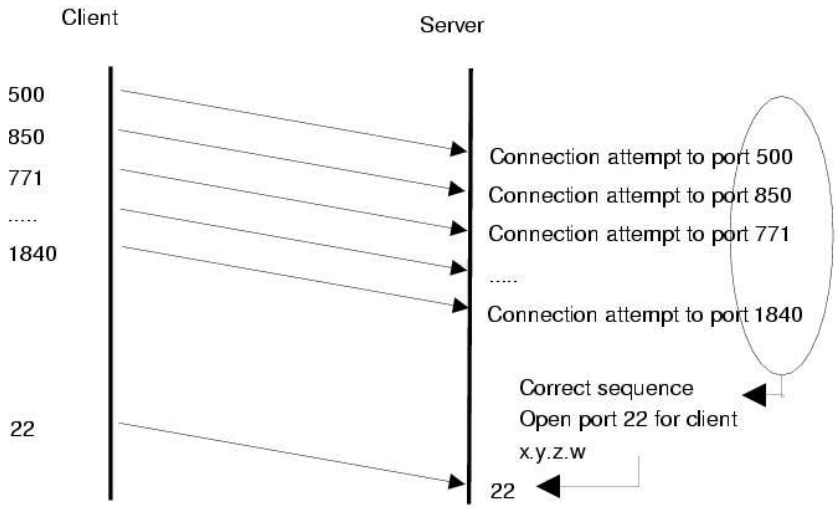

Fig. 1. Illustration of the port knocking authentication mechanism

appropriate "knocks" in the server. To identify the clients, port knocking makes the sequence dependant of the client's network address (as stated in equation 4). However, this raises the question of what might happen if two clients shared the same address, e. g.: if Network Address Translation (NAT, [7]) is used. As shown in Figure 2, when packets from inside the NAT exit the private network they all share the same source address (the public address for the NAT), and packets cannot be used to identify the source behind the NAT without accessing the router's NAT tables.

We can see in Figure 2 how two hosts A and B share the same public IP address, so they have to use NAT to access other networks. Packet 1 (created by A) is translated into packet 2, where the source address and port have changed to the public address and a free port in the NAT device. The same process is done with packet 3 from B. As we can see, from the outside of the NAT it is impossible to determine which packet comes from which source using the source address.

As port knocking relies in the network address to open required ports to trusted clients, it cannot differentiate among all the clients behind the same NAT device. As a result, when a client in a network that uses NAT identifies itself to the port knocking server and gets access to the service port, it has given access to every device that uses its same NAT (and, consequently, shares the same public network address).

Furthermore, when the trusted user is "knocking" the firewall there is no need for a potential attacker to watch the authentication sequence, she just needs to wait until the service port is open to have access to it without knowing the authentication sequence. An example can be seen in Table 1: Client 172.16.8.102 is the authorized user that authenticates using portk nocking with

Table 1. Network capture of the NAT-Knocking Attack

No.	Source	Destination	Protocol	Info
1	172.16.8.102	163.117.149.93	TCP	32987 → 7682 [SYN]
2	172.16.8.102	163.117.149.93	TCP	32988 → 7697 [SYN]
3	172.16.8.102	163.117.149.93	TCP	32989 → 7810 [SYN]
4	172.16.8.102	163.117.149.93	TCP	32990 → 7800 [SYN]
5	172.16.8.102	163.117.149.93	TCP	32991 → 7811 [SYN]
6	172.16.8.102	163.117.149.93	TCP	32992 → 7809 [SYN]
7	172.16.8.102	163.117.149.93	TCP	32993 → 7673 [SYN]
8	172.16.8.102	163.117.149.93	TCP	32994 → 7686 [SYN]
9	172.16.8.102	163.117.149.93	TCP	32995 → 7603 [SYN]
10	172.16.8.102	163.117.149.93	TCP	32996 → 7682 [SYN]
11	172.16.8.102	163.117.149.93	TCP	32997 → 7602 [SYN]
12	172.16.8.102	163.117.149.93	TCP	32998 → 7887 [SYN]
13	172.16.8.102	163.117.149.93	TCP	32999 → 7699 [SYN]
14	172.16.8.102	163.117.149.93	TCP	33000 → 7808 [SYN]
15	172.16.8.102	163.117.149.93	TCP	33001 → 7629 [SYN]
16	172.16.8.102	163.117.149.93	TCP	33002 → 7602 [SYN]
17	172.16.8.102	163.117.149.93	TCP	33003 → 7686 [SYN]
18	172.16.8.102	163.117.149.93	TCP	33004 → 7663 [SYN]
19	172.16.8.102	163.117.149.93	TCP	33005 → 7655 [SYN]
20	172.16.8.102	163.117.149.93	TCP	33006 → 7692 [SYN]
21	172.16.8.102	163.117.149.93	TCP	33007 → 7992 [SYN]
22	172.16.8.102	163.117.149.93	TCP	33008 → 7839 [SYN]
23	172.16.8.102	163.117.149.93	TCP	33009 → 7637 [SYN]
24	172.16.8.102	163.117.149.93	TCP	33010 → 7990 [SYN]
25	172.16.8.102	163.117.149.93	TCP	33011 → 80 [SYN]
26	163.117.149.93	172.16.8.102	TCP	80 → 33011 [SYN, ACK]
27	172.16.8.102	163.117.149.93	TCP	33011 → 80 [ACK]
28	172.16.8.102	163.117.149.93	HTTP	GET /info.php HTTP/1.1
29	163.117.149.93	172.16.8.102	TCP	80 → 33011 [ACK]
30	163.117.149.93	172.16.8.102	HTTP	HTTP/1.1 200 OK
31	**172.16.8.100**	163.117.149.93	TCP	1196 → 80 [SYN]
32	163.117.149.93	**172.16.8.100**	TCP	80 → 1196 [SYN,ACK]
33	**172.16.8.100**	163.117.149.93	TCP	1196 → 80 [ACK]
34	**172.16.8.100**	163.117.149.93	HTTP	GET /info.php HTTP/1.1
35	163.117.149.93	**172.16.8.100**	TCP	80 → 1196 [ACK]

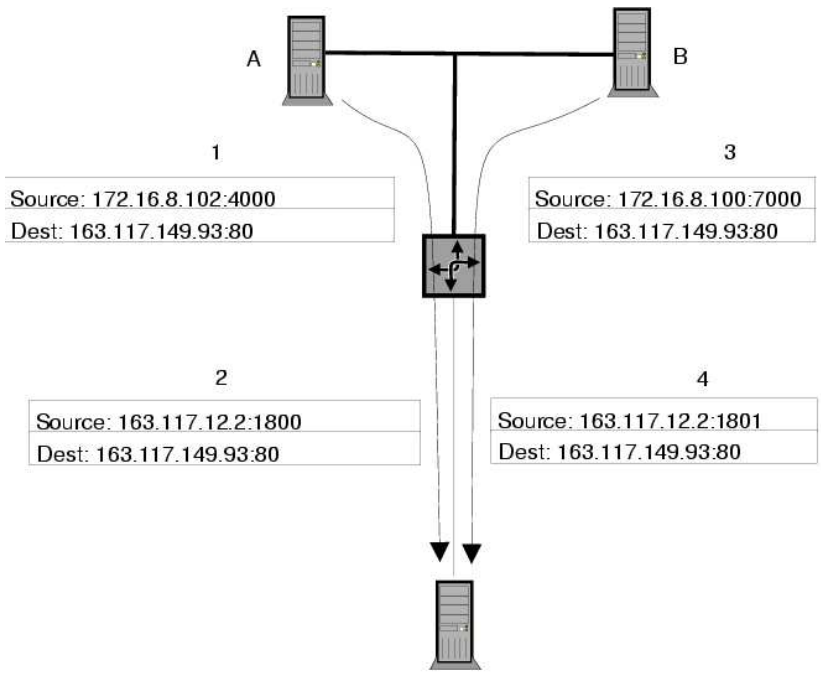

Fig. 2. Sample NAT configuration, with two hosts A and B sharing the same network address

server 163.117.149.93 requesting the opening of port 80. We can see all the knocks sent to the server (*frames 1 to 24*) and how after that the client is able to connect to port 80 and proceeds with normal communication (*frames 25 to 30*). However, client 172.16.8.102 is using NAT, and another client in the same network (*172.16.8.100*) can connect to the service opened by the authorized user without performing the knock sequence (*frames 31 to 35*).

3 DoS-Knocking: Handling Large Amounts of Work

In order to work properly, port knocking server must control the connection attempts made against it, so it can look for authentication patterns and open ports when requested by authorized users. This implies that a process must parse the logs in real time to automatically detect the sequences.

When analyzing the way port knocking fulfills this duty we realize that the parser process must use a buffer for each different client that makes a connection against a closed port, so the process is able to track each authentication

sequence.[1] If an attacker manages to send forged packets with random source network addresses (the same way some worms propagate [3]), the parser process would have to create a buffer for each one of the addresses, making this process to consume high amounts of memory. In the Appendix is included the source code to generate an attack consisting of sending packets with random fake source addresses to random ports on the target.

Another two considerations about DoS attacks on portk nocking are about parameter encryption and the parser performance. [4] recommends parameter encryption so that integrity and confidentiality can be achieved, and [9] proposed a solution based on one-time-passwords to avoid replay attacks. However, when using encryption the server must decrypt the contents of the authorization sequence for all the clients that try to authenticate using port knocking. This can lead to high CPU loads, as shown in Table 2, where a Pentium-class computer (166 MHz and 32 MB of RAM) could overload a Pentium III-class server (500 MHz and 192 MB of RAM). Rows in bold are the status when the attack (60.000 packets with different forged source address aimed at random ports) took place. Last two bold rows of the table show status when attack had finished, but the server was still processing the fake knocks. Furthermore, we should consider whether having a process parsing logs in real time is something our system can handle, or maybe that will heavily decrease its performance.

In order to prevent this kind of attacks it could be necessary to use port knocking just for the range of addresses that would use it and, if possible, to have separate logs for those knocks that come from those addresses and those that don't, so we reduce the work of the parser. Additionally, we should consider using a fast encryption algorithm that doesn't require a lot of CPU cycles to decrypt the contents of the port sequence, such as TEA [8].

4 Other Implementation Issues

Apart from the already discussed flaws in port knocking, a real implementation presents many problems that need to be faced, the first of all being the port range being used to knock. Although [5] criticized the shortage of ports being used from the cryptographic point of view, we can also assure that a small range to perform the knocks makes it possible for an attacker to discover (by capturing several authentication sequences) the range of valid ports and perform brute-force attacks. On the other hand, if the implementation had chosen a large range of valid ports, it would be cryptographically more securer, but it would be easier to perform a DoS-Knocking attack, as most of the ports are valid and the parser process would have to create buffers for each knock.

A proposed solution for this implementation issue would be to choose well-balanced groups of large ranges that take into account our needs of confidentiality and performance. If out main concern is performance, we should choose a handful

[1] Even if we select a reduced range of port to watch instead of considering the whole range of 65.535 ports as "knocking-available" we still need a large amount of them to provide the system with a minimum security level.

Table 2. Status of the port knocking server as reported by `vmstat` during the proposed DoS-Knocking attack, with time interval between measures being 5 seconds

Procs		Memory				CPU		
R	B	Swpd	Free	Buf	Cache	Us	Sy	Id
1	0	0	7536	8920	119524	0	0	100
2	0	0	7536	8920	119524	2	0	98
2	0	0	7536	8920	119524	0	1	99
1	0	0	7536	8920	119524	0	1	99
1	0	0	3784	8924	119524	0	0	100
5	**0**	**0**	**3976**	**8924**	**117472**	**12**	**47**	**42**
6	**0**	**0**	**3264**	**8916**	**116648**	**23**	**77**	**0**
5	**0**	**0**	**3628**	**8920**	**112636**	**58**	**42**	**0**
4	**0**	**0**	**3708**	**8920**	**111668**	**81**	**19**	**0**
3	**0**	**0**	**3816**	**8920**	**110696**	**85**	**15**	**0**
5	**0**	**0**	**3960**	**8920**	**109720**	**92**	**8**	**0**
4	**0**	**0**	**3052**	**8920**	**109732**	**88**	**12**	**0**
5	**0**	**0**	**3576**	**8928**	**107916**	**50**	**50**	**0**
4	**0**	**0**	**3832**	**8928**	**106092**	**41**	**59**	**0**
3	**0**	**0**	**3908**	**8924**	**102180**	**89**	**11**	**0**
3	**0**	**0**	**3900**	**8928**	**102180**	**83**	**17**	**0**
4	**0**	**0**	**3900**	**8928**	**102180**	**89**	**11**	**0**
1	**0**	**0**	**3900**	**8928**	**102180**	**78**	**13**	**9**
1	0	0	3896	8928	102180	0	0	100
1	0	0	3896	8928	102180	0	0	100
2	0	0	7688	8932	102180	1	0	99
1	0	0	7688	8932	102180	1	1	98

of large ranges (for example, 10 ranges of 1000 ports) so the ranges are not small enough to permit identification with few data captures, and at the same time they are not large enough to favor DoS-Knocking attacks. On the other hand, if we must provide our system with maximum security we should consider using all available ports in order to maximize the key to our network (as [5] details).

Anyways, traffic analysis is always possible, giving an attacker the clue about the authentication mechanism being used. If an attacker can gather traffic data that include several authentications, she may realize that prior to any connection to a protected port there are a number of connection attempts against closed ports (and depending on the number of authentications captured, it could be possible to identify the valid ports ranges being used). Therefore, we have a situation that revokes the main advantage of port knocking: being an stealth method of authenticating. [5] proposes using UDP instead of TCP in order to make the authentication really stealth. However, given an attacker with enough data collected from the network, the amount of UDP traffic to closed ports would be clearly higher than expected and with no apparent reason, giving the attacker an idea of what is going on.

5 Conclusions

The analysis of port knocking authentication method has revealed both some design flaws and implementation problems that could provide access to unauthorized users and/or cause a DoS attack on the port knocking server. We have described the NAT-Knocking attack and stated that the problem of access control based on network address when NAT is used cannot be solved as long as network address is the only identifier used. Regarding this, port knocking does not improve the security offered by other network-level devices and mechanisms, such as firewalls.

We have also described the DoS-Knocking attack and we have proven that a low-end computer is able to take a much powerful server to the limits of its computing capabilities. We have seen that encryption and the clients-tracking mechanism tend to exhaust both memory and CPU, so it is possible that under peaks of work legitimate users could perform a DoS attack without pretending to.

Finally, partly due to implementation and partly due to the design, we have shown how traffic analysis is possible for an attacker with enough data captured from the network, so she could have information about the authentication procedure. This fact attacks directly the basis of port knocking: being an stealth authentication scheme.

Appendix: DoS-Knocking sample code

Code used to perform the DoS-Knocking attack.

```
#include   "forgeit.h"
#define INTERFACE  "eth0"
#define INTERFACE_PREFIX 14

char SOURCE[16],DEST[16];
int SOURCE_P,DEST_P;

int main(int argc, char *argv[])
{
   int i, quantity, starting_port, range, fd_send;
   if(argc != 5) {
      printf("\tusage: %s ip_dst quantity init_port port_qty\n",
                  argv[0]);
      exit(0);
      }

   DEV_PREFIX = INTERFACE_PREFIX;
   memset(SOURCE,0,16);
   memset(DEST,0,16);
```

```
   srand(time(NULL));
   strncpy(DEST,argv[1],15);
   quantity = atoi(argv[2]);
   starting_port = atoi(argv[3]);
   range = atoi(argv[4]);
   fd_send = open_sending();

   for (i=0;i<quantity;i++)
   {
      snprintf(SOURCE,15,"%d.%d.%d.%d"
                  ,1+(int)(254.0* rand()(RAND_MAX+1.0))
                  ,1+(int)(254.0*rand() / (RAND_MAX+1.0))
                  ,1+(int)(254.0*rand() / (RAND_MAX+1.0))
                  ,1+(int)(254.0*rand() / (RAND_MAX+1.0)));

      SOURCE_P=1024+(int)(60000.0*rand() / (RAND_MAX+1.0));
      DEST_P=starting_port+(int)(((float)range)*rand() /
                  (RAND_MAX+1.0));
      transmit_TCP (fd_send, SOURCE, SOURCE_P,DEST, DEST_P,SYN);
   }

   return 0;
}
```

(Example based on Brecht Claerhout's `ipspoof` and `sniper-rst` code [2], modified to improve packet-sending performance)

References

1. Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed. J. Wiley & Sons (2001)
2. Claerhout, B.: `http://cs.ecs.baylor.edu/~donahoo/NIUNet/hacking/hijack/`
3. Knowles, D., Perriot, F., Szor, P.: W32.Blaster.Worm Report. Symantec Security Response (2003)
4. Krzywinski, M.: Port Knocking: Network Authentication Across Closed Ports. SysAdmin Magazine 12 (2003) 12-17
5. Narayanan, A.: A critique of Port Knocking NewsForge, August 8 (2004) `http://software.newsforge.com/software/04/08/02/1954253.shtml`
6. Schneier, B.: Port Knocking. Crypto-Gram Newsletter, March 15 (2004) `http://www.schneier.com/crypto-gram-0403.html#5`
7. Srisuresh, P., Egevang, K.: Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (2001)
8. Wheeler, D., Needham, R.: TEA, a Tiny Encryption Algorithm. Fast Software Encryption 1994: 363-366
9. Worth, D.: CÖK - Cryptographic One-Time Knocking. BlackHat 2004.
10. Yan, J., Blackwell, A., Anderson, R., Grant, A.: The Memorability and Security of Passwords. Some Empirical Results. Technical Report No. 500, Computer Laboratory, University of Cambridge (2000)