Safe and Correctness Strategies for Updating Firewall Policies

A. Kartit[†]**, A. Radi**[††]**, M. El Marraki**[†††]**and B. Regragui**[††††]

University Mohammed V, Faculty of Sciences, Rabat, Morocco

Summary
Policy deployment is the process by which policy editing commands are issued on firewall, so that the target policy becomes the running policy. Due to the sensitive nature of information transmitted during a policy deployment, the communication between management tool and firewall should be confidential [1]. Much research has already addressed to the specification of policies, conflict detection and optimization, but very little research is devoted to the security and correctness of firewall policy deployment. In this paper, we make some contributions to the correctness of Firewall Policy Deployment and propose an effective solution that will allow us to secure the deployment process of a political target. We show that the category of type I policy editing [2] is incorrect and could lead to security vulnerabilities. We then provide a correct algorithm for Type I Deployment. Our algorithm can be used even for the deployment of policies whose size is very large.

Key words:
Target Policy Deployment (TPD), Firewall Policy Management (FPM), Securing Exchanges (SE), Security of Policy Deployment (SPD).

1. Introduction

Network firewalls are devices or systems that control the flow of traffic between networks employing different security postures. The network traffic flow is controlled according to a firewall policy. The filtering decision is based on a firewall policy defined by network administrator.
An administrator may want to configure in real time an active policy to replace it with a new policy. This configuration is still problematic because it must reconcile the continued service and avoid security breaches. The ordered list of operations to be applied to achieve a new configuration is particularly sensitive. As a result, these policies require automatic tools for providing a right environment to specify, configure and deploy target security policy. Much research has dealt with the specification {[3], [4], [5]} policies, conflict detection {[6], [7], [8]} and the optimization problem {[9], [10]}, but very few studies have interested to the deployment of policies. That is why we have tried to focus on problems associated with the deployment of policies to make it easier for network administrators.
Only recently, some researchers have proposed deployment strategies for two important categories of policy deployment [2]. In this paper, we analyze the algorithms provided in [2] and show that these algorithms have serious flaws. We present an improved correctness formalization that can be used as a basis for formulating correct deployment strategies. Our work is focused on language editing policy type I. We will demonstrate that the algorithm ”Scanning Deployment” already proposed is incorrect and we propose another version of this algorithm which is correct and will allow us to replace a source policy with a target policy.
Generally, firewalls are configured to protect against unauthenticated access the external network. They ensure, among other things, a filtering function at different levels of the OSI layer and prevent intruders to log on machines of the internal network.
However, this system firewall is insufficient if not accompanied by other protections. Indeed, it does not provide security services like Authentication of the source data, integrity and confidentiality [11].This in mind that we thought to implement other security protocols within the firewall (SSL, SSH and IPSec) to ensure the security of data exchange with other firewalls and especially Security of Policy Deployment.

2. Firewall Background

A firewall is a perimeter security device that filters packets that traverse across the boundaries of a secured network “Fig. 1”. The filtering decision is based on a firewall policy defined by network administrator.

It is possible to use any field of IP, UDP, or TCP headers [2]. However, the following five fields are most commonly used: protocol type, source IP address, source port, destination IP address and destination port “Fig. 2”.
Any field in a packet’s header can be used for the matching process. However, the same five fields are most commonly used. In a packet, each of these fields has an atomic value. If all the fields of a packet p match with the corresponding fields of a rule r, then p is accepted or rejected according to the decision field of r. If p does not match to any rule in policy, then the default match-all rule is applied.

Manuscript received March 5, 2011
Manuscript revised March 20, 2011

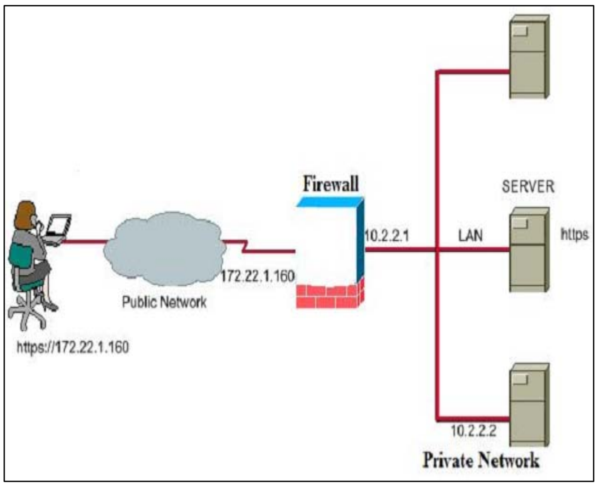
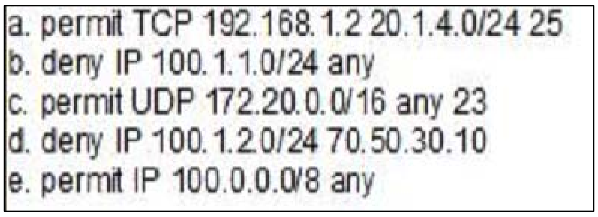


Fig. 1 Firewall-architecture.

Fig. 2 Example of a firewall policy.



a. permit TCP 192.168.1.2 20.1.4.0/24 25
b. deny IP 100.1.1.0/24 any
c. permit UDP 172.20.0.0/16 any 23
d. deny IP 100.1.2.0/24 70.50.30.10
e. permit IP 100.0.0.0/8 any

3. Policy Deployment

The deployment of a firewall policy should have the following characteristics [2]: correctness, confidentiality, safety and speed.
Correctness: A deployment is correct if it successfully implements the target policy on the firewall. After a correct deployment the target policy becomes the running policy. Correctness is an essential requirement for any deployment.
Confidentiality: Confidentiality refers to securing the communication between a management tool and a firewall. Due to the sensitive nature of information transmitted during a deployment, the communication between management tool and firewall should be confidential.
Safety: A deployment is safe if no legal packet is rejected and no illegal packet is accepted during the deployment. Deployment safety is a new and challenging area of research.
Speed: A deployment should be done in the shortest time, so that the desired state of affairs is achieved as quickly as possible. A deployment algorithm should have a good running time, so that it is applicable even for large policies. Different firewalls support different policy editing commands. The set of policy editing commands that a firewall supports is called its policy editing language.
A firewall has a new running policy every time an editing command is applied. Thus a deployment can be viewed as a sequence of running policies $I = H_0, H_1, \ldots, H_{n-1}, H_n = T$, with H_{i+1} derived by applying an editing command to H_i.
In [2], the authors classify policy editing languages into two representative classes, Type I and Type II, and provide deployment algorithms for both types of languages. Type I editing supports only two commands, append and delete. Command (app r) appends a rule r at the end of the running policy R, unless r is already in R, in which case the command fails. Command (del r) deletes r from R, if it is present. As Type I editing can transform any running policy into any target policy [2], therefore it is complete. Most older firewalls and some recent firewalls, such as FWSM 2.x and JUNOSe 7.x, only support Type I editing.
Indeed, the deployment algorithm type I used is called "Scanning Deployment".

3.1 The Algorithm “Scanning Deployment”

Algorithm 1: Scanning Deployment (already existed) [2]

```
Scanning_Deployment (I, T) {
/* An algorithm using only app and del
to transform policy I into policy T */
S ← empty stack
H← empty hash table
/* Phase 1: add rules */
i ← 1
for t ← 1 to SizeOf(T) do
    while i ≤ SizeOf(I) and I[i]<> T[t] do
        /* I[i] needs to be deleted */
        S. PUSH (I[i])
        H.ADD (I[i])
        i ← i + 1
    if i > SizeOf(I) then
        if H.Contains(T[t]) then
            H.Remove(T[t])
            IssueCommand( del T[t])
        IssueCommand( app T[t])
/* Phase 2: clean up */
for j ← SizeOf(I) down to i do
    IssueCommand( del I[j])
while not S.IsEmpty() do
    r ← S.POP()
    if H.Contains(r) then
        IssueCommand( del r)
}
```

I[i]: is the i^{th} rule of the original policy. In the real case can be replaced for example by “Permit TCP 207.160.100.1 20.30.40.0/24 25”.

Shortcoming of this algorithm: Phase 2 of the algorithm does not give good results. We will show this through a sample run “Figure 3”.

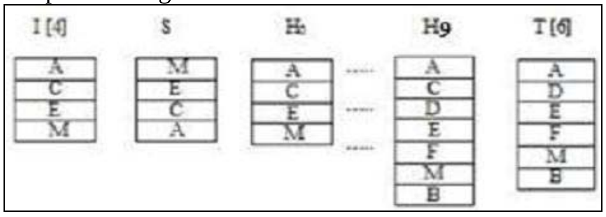


Fig. 3 Scanning_Deployment phase 1 running example

We completed the first phase with i = 5 and t = 7, Sizeof(I) = 4, so we will never run the loop:

```
for j ← SizeOf(I) down to i do
IssueCommand (del I[j])
```

After running phase 2, the algorithm gives the following result: “Figure 4”.

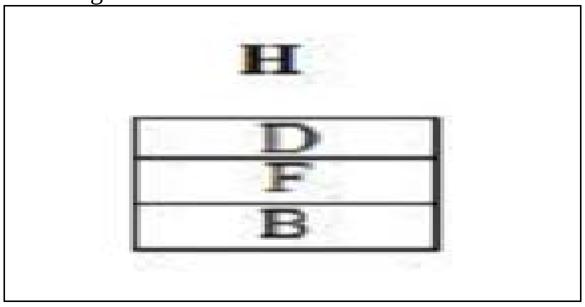


Fig. 4 Scanning_Deployment phase 2 running example

It is therefore clear that H is different from T. Therefore, the algorithm is not correct.

3.2 Our contribution:”Enhanced Scanning Deployment”

We start by giving a simple deployment algorithm for an initial policy I and target policy T that will allow us to correct the algorithm "scanning deployement". I and T are coded as arrays of characters, so that I[i] refers to the ith rule of I. Initially, the running policy H equals I. In phase1, the algorithm appends to the end of H every rule r in T, starting from r = T [1]. If r is already in I, then it removes r from H before appending it back. In phase 2, it removes from H every rule r that is in I but not T. the new algorithm is called: “Enhanced_Scanning_Deployment”

Algorithm 1: Scanning Deployment (new release)

```
Enhanced_Scanning_Deployment (I,T) {
/* an algorithm using only app and del
to transform policy I into policy T */
H← empty hash table
/* Phase 1: add rules */
i←1;
for (t=1 to SizeOf(T)) do
while((i<=SizeOf(I)) AND (I[i]<>T[t]))
do
/* I[i] needs to be deleted */
H. ADD(I[i]);
i ← i + 1;
end while
if (i>SizeOf(I)) then
if (H.Contains(T[t])) then
H.Remove(T[t]);
IssueCommand(del T[t]);
end if
IssueCommand(app T[t]);
end if
end for
/* Phase 2: clean up */
s←sizeof(I)+sizeof(T)-sizeof(I∩T);
k←1;
While (s>sizeof(T)) do
t←1; find←false;
While((t<=sizeof(T))AND(find=false)) do
If (H(k)=T(t)) then
k←k+1;
find ←true ;
else
t←t+1 ;
end if
end while
If (find=false) then
Issuecommand(del(H(k));
s←s-1;
end if
end while
}
```

This algorithm gives good results whatever the size of the original and target policy.

We will show this through the previous example. “Figure 5”, “Figure 6” *and* “Table1”.

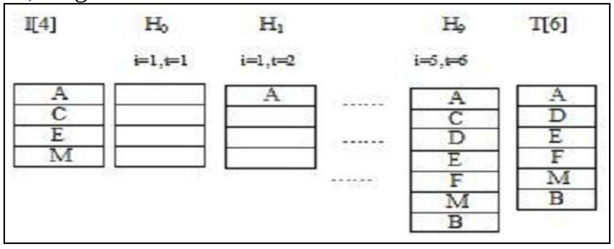


Fig. 5 Enhanced_Scanning_Deployment phase 1 running example

Policy size / Time (s)	Policy 1 (size=200)	Policy 2 (size=1000)	Policy 3 (size=2000)	Policy 4 (size=4000)
Time0 (0,00)	0%	0%	0%	0%
Time1(0,003132)	100%	20%	10%	5%
Time2(0,01566)	100%	100%	50%	25%
Time3(0,03132)	100%	100%	100%	50%
Time4(0,06264)	100%	100%	100%	100%

Table 2: Results of experiments (in seconds)

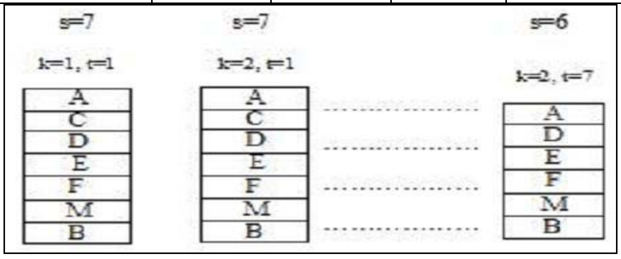


Fig. 6 Enhanced_Scanning_Deployment phase 2 running example

Table1: Enhanced_Scanning_Deployment phase 2 manual running example

s	t	k	find	Sizeof(T)
7	1	1	false	6
7	1	2	true	6
7	1	2	false	6
7	2	2	false	6
7	3	2	false	6
7	4	2	false	6
7	5	2	false	6
7	6	2	false	6
6	7	2	false	6

Having finished the execution of the algorithm ”Enhanced_Scanning_Deployment”, policy being implemented is identical to the policy target (H=T). Therefore, we can say that this new version of the Algorithm is correct.

3.3 Implementation and Performance Evaluation of the “Enhanced Scanning Deployment”

To test and evaluate the performance of the new algorithm, we implemented it in C++, and all tests are performed on ACCENT with Intel(R) Core(TM) 2 DUO CPU 2.00Ghz (2 CPUs) processor and 4GB of RAM. We use four firewall policies with 200, 1000, 2000 and 4000 rules to convert initial policy to the target policy. The results of each test on policies 1-4 are given in the table below “Table2”. All times are represented in seconds.

It’s clear that “Enhanced Scanning Deployment” takes a fraction of second to calculate the correctness of deployment for policies as large as Policy 4 “Figure 7”.

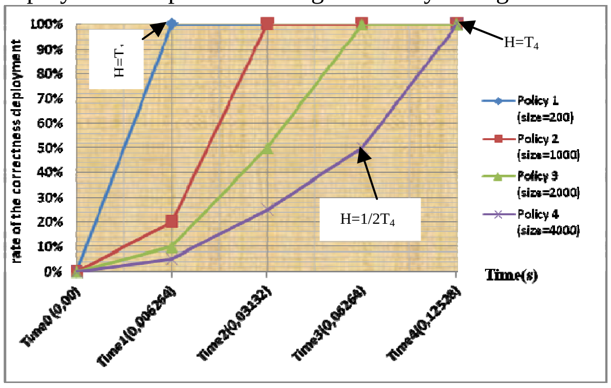


Fig. 7 Time required for a correct deploying for the target policy

4. Implementation of the Security Solution

4.1 The solution adopted

After having compared the three security protocols (SSL, SSH, IPSec) [1], it was found that the solution based on VPN over SS is best suited for the deployment of policies because it is easy to setup, need non-administrative access and work reliably.
To most users SSH appears to be terminal emulator similar to Telnet. The users do not see the encryption and therefore the security is transparent for the user. For system administrators SSH is a popular remote administration platform.

4.2 Creating a VPN tunnel

Here is the sequence of commands that we have entered in our command line:

```
pppd debug updetach noauth \
pty "ssh -l login -t -t @distante \
pppd noauth 200.100.254.254:200.100.253.253"
```

In the first command line, “noauth” request that pppd does not care about the authentication part. This return to SSH.
In the second command line, “pty” option allows here to pass the following commands to the remote shell that we have just opened. The -t option to ssh, in turn, forces the allocation of pseudo-tty on the remote machine.
The last line assigns a private address at each end virtual network. We can connect our machines from one to another without any worries. We just create our VPN.

4.3 Establish the IP_forwarding

The IP_FORWARDING is a flag that tells the Linux kernel if the packets must pass through the machine or, on the contrary, it must be stopped. By default, the current distributions, it is initialized to 0 at startup. The following script allows you to initialize it to 1 if you want one of two machines, or both, serve as a gateway for other machines on your network, allowing not only communication from one station to another but also from one network to another.

```
#!/bin/sh
echo " Setting up IP forwarding rules "
echo 1 > /proc/sys/net/ipv4/ip_forward
echo -n "/proc/sys/net/ipv4/ip_forward: "
cat /proc/sys/net/ipv4/ip_forward
for forwarding in /proc/sys/net/ipv4/conf/*/forwarding
do
echo -n "$forwarding: ";
interface=‘dirname $forwarding‘
interface=‘basename $interface‘
case "$interface" inppp*|eth1)
# interface list when the transfer must be #enabled
echo 1 > $forwarding
;;
*) # it desactivates interfaces that do not require the transfer.
echo 0 > $forwarding
;;
esac
cat $forwarding
done
```

4.4 SSH server configuration

The SSH server is shipped with default configuration file named sshd_config. By default, it listens on port 22; we will modify it to listen on port 9870. This results in two things immediately:

-Robots that scan port 22 to find a fallible will not bore your ssh server.

-Logs authentication normally concerns only access attempts to your vpn.

Here is the entire configuration file used for our example:

```
# vpn/etc
# Specific configuration of the port Port 9870
PidFile /var/run/sshd_vpn.pid
HostKey /etc/ssh/ssh_host_key
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
# Configuration of log levels
SyslogFacility AUTH
LogLevel INFO
RSAAuthentication yes
AllowUsers sshvpn
# Restrictions
#IgnoreRhosts yes
IgnoreUserKnownHosts yes
PermitRootLogin no
StrictModes yes
PasswordAuthentication no
PermitEmptyPasswords no
ChallengeResponseAuthentication no
RhostsAuthentication no
RhostsRSAAuthentication no
X11Forwarding no
PrintMotd no
KeepAlive yes
```

4.5 Configuring the VPN

Since our initial tests have been successful, we certainly will desire to automate our VPN. To do this, it is useful and proper to create a configuration file that contains the variables necessary for the creation of our VPN tunnel:

```
# VPN1 Configuration File
# /opt/ssh-vpn/etc/vpn1
# The networks are connected to a side,
# following the route command:
client_network=192.168.2.0/24
server_network=192.168.1.0/24
# Do you want information to debug?
client_debug="no"
server_debug="yes"
# Take different IPs for each VPN required.
server_ppp_ip=192.168.254.254
client_ppp_ip=192.168.254.253
# is there a PPP authentication required?
client_require_pap="yes"
server_require_pap="yes"
client_require_chap="no"
server_require_chap="no"
# Need non-standard pppd arguments? Put them here.
#client_pppd_args="usepeerdns"
#server_pppd_args="proxyarp"
# Need additional arguments ssh? Put them here
#client_ssh_args="-C"
#server_ssh_args=""
```

5. Conclusion

In this paper, we showed, through examples, that the policy language edition type I is not accurate but we could make it correct through the changes we have made on the algorithm “Scanning Deployment». So, our algorithm

"enhanced scanning deployment" has allowed us to deploy the policy target with accuracy regardless of its size. Indeed, we have chosen and implemented VPN over SSH to ensure the security of policy deployment.
We will be soon working on language editing Type II policies to make deployment very effective, safe and fast.

Acknowledgments

I would like to thank my wife who helped me in correcting this paper. The authors also wish to thank many anonymous referees for their suggestions to improve this paper.

References

[1] A. Kartit, M. El Marraki, A. Radi and B. Regragui, "On the Security of Firewall Policy Deployment", Journal of Theoretical and Applied Information Technology, ISSN: 1817-3195, Volume 22, n°2, pages 22 – 27, 2010.
[2] C. C. Zhang, M. Winslett, and C. A. Gunter, "On the Safety and Efficiency of Firewall Policy Deployment", In SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy, pages 33-50,Washington, DC, USA, 2007.
[3] E. Al-Shaer and H. Hamed, "Modeling and Management of Firewall Policies", Network and Service Management, IEEE Transactions on, 1(1):2-10, April 2004.
[4] Y. Bartal, A. J. Mayer, K. Nissim, and A.Wool. Firmato, "A Novel Firewall Management Toolkit", In IEEE Symposium on Security and Privacy, pages 17-31, 1999.
[5] M. G. Gouda and A. X. Liu, "Firewall Design: Consistency, Completeness, and Compactness", In ICDCS, pages 320-327, 2004.
[6] F. Baboescu and G. Varghese, "Fast and Scalable Conflict Detection for Packet Classifiers", In ICNP, pages 270-279, 2002.
[7] Z. Fu, S. F.Wu, H. Huang, K. Loh, F. Gong, I. Baldine, and C. Xu, "IPSec/VPN Security Policy: Correctness, Conflict Detection, and Resolution", In POLICY, pages 39-56, 2001.
[8] A. X. Liu, "Change-impact analysis of firewall policies", In ESORICS, pages 155-170, 2007.
[9] H. Hamed and E. Al-Shaer, "Dynamic rule-ordering optimization for highspeed firewall filtering", In ASIACCS, pages 332-342, 2006.
[10] J. Qian, "ACLA: A framework for Access Control List (ACL) Analysis and Optimization",
[11] booktitle = Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security Issues of the New Century. page 4, Deventer, The Netherlands, The Netherlands,2001.

Ali Kartit received the Master and DESA degrees in Network & Telecommunications and informatics & Telecommunications from Henry poincaré university (France-Nancy1) and Faculty of sciences (Morocco-Rabat) in 2003 and 2006, respectively. During 2009-2011, he stayed in Network & Data mining Laboratory. He is a PhD student in the field of security management of computer networks at the Faculty of sciences in Rabat (FSR). He is certified Cisco and Exchange 2003 Server.