PORT KNOCKING - A STATIC IMPLEMENTATION

What is port knockng first of all? Think of it as an additional layer of security useful in certain situations. Situations espically where you need access to a protocol on a machine sitting in a hostile environment. Think of a web-server hosted at an ISP to which you would like SSH access, or a server sitting at a business partners network to which you want to upload files, or any other similar circumstance. Generally such machines are single purpose machines -such as the web server- but because they are not on-site you need to administer them using remote access, but yet leaving such protocols open is dangerous. Port knocking can help you out of this quandry.

What is It?
Port knocking simply is a server listening for a certain "*handshake*", a specific sequence of ports being accessed, or "*knocked*", then once it sees such a sequence it will take a specified action - like opening up a specific port temporarily. Much like passwords there are different ways to do port knocking;

- *Covert*, this is where the "*knock*" is hidden inside any normal traffic the server might see
- *Dynamic*, this is where the "*knock*" sequence is varied from session to session
- *One-time*, this is where each "*knock*" sequence is only used once
- *Static*, this is where the "*knock*" sequence remains constant across usage

In this article we will be implementing a simple static port-knocking system. Static port-knocks are vulnerable to replay attacks, if an attacker can monitor enough traffic to and from the server he could deduce the "*knock*" sequence. This is a real risk, but in certain situations the chances of this or negligable. So it is for those situations that we could use static port-knocking.

What do we Need?
I am going to be using a linux server, with a bash shell, iptables, the commonly distributed tool tcpdump and some knowledge of bash scripting would be useful. I use this script on a machine where I want SSH terminal access. Now the firewall denies SSH access to the machine even though the server is running. This means means that under normal circumstances I cannot gain SSH terminal access. As always, these scripts are not works of art, I am sure they could be improved and if you want to do so please feel free. We will also need a home directory for the scripts, perhaps something like */admin/pknock*.

The Listener
The first script we need is the process which will listen for the "*knock*" sequence. You will need to create a named pipe for the script to use (*mkfifo -m 700 /admin/pknock/logfile*), and then for the script, I call it the *listend* script.

```
TDMP=/usr/sbin/tcpdump
LOG=/admin/pknock/logfile
LOPT="-i eth1 -q -nn -c 3 "dst port (111 or 2222 or 33333)""

TST=1
while [ $TST == 1 ]
 do
  $TDMP $LOPT | grep -e "<servers_ip>" > $LOG
  TST=1
 done
```

This *listend* script runs tcpdump and captures 3 packets specified by the filter -which is the knock sequence - , also since we are using pipes we do not need to worry about using up spans of disk space. Also, since we are only interested in packets the server see's, we do not put the network interface into promiscous mode.

The Checker
The second script needed is the process which will check the logs created by the first and see if the "*knock*"

sequence had been sent, if it had, then it must open up the SSH port. It is also in this script where the "*knock*" sequence is set.

```
TDMP=`which tcpdump`
LLOG=/admin/pknock/logfile
IPT=/usr/local/sbin/iptables

##next is the knock sequence
LOPT="port (111 or 2222 or 33333)"

##start the process loop
TST=1
while [ $TST == 1 ]
 do
  ##get time 5 minutes ago
  BDATE5=`date +%R -d '-5 minutes'`
  BDATE4=`date +%R -d '-4 minutes'`
  BDATE3=`date +%R -d '-3 minutes'`
  BDATE2=`date +%R -d '-2 minutes'`
  BDATE1=`date +%R -d '-1 minutes'`
  BDATE0=`date +%R`
  ##check for the knock
  KEEP=`cat $LLOG`
  ##check for all 3 ports
  echo $KEEP | grep ".111" $2> /dev/null
  CHK1=`echo $?`
  echo $KEEP | grep ".2222" $2> /dev/null
  CHK2=`echo $?`
  echo $KEEP | grep ".33333" $2> /dev/null
  CHK3=`echo $?`
  ##check if it is recent enough - last 5 minutes and that all were present
  if [ $CHK1 == "0" ] && [ $CHK2 == "0" ] && [ $CHK3 == "0" ]
   then
    PDATE=`echo $KEEP | cut -f 1,2 -d ":"`
    if [ "$BDATE5" == "$PDATE" ] ||  [ "$BDATE4" == "$PDATE" ] || [ "$BDATE3" ==
"$PDATE" ] || [ "$BDATE2" == "$PDATE" ] || [ "$BDATE1" == "$PDATE" ] || [ "$BDATE0" ==
"$PDATE" ]
     then
      ##insert rule to allow ssh access
      ##i insert it after my packet checks of which I have 12
      $IPT -I INPUT 13 -p tcp -m tcp --dport 22 -j ACCEPT
      logger "pknock: allow access"
      ##give chance to login and then close again
      sleep 60
      $IPT -D INPUT 13
     else
      sleep 5
    fi
  fi
  TST=1
 done
```

This is the script where you can change what must happen when the "*knock*" happens.

Starting it All

I generally want my two port knocking processes to startup as the machine starts up, so I put the following lines into my */etc/rc.d/rc.local* (you can put them into any other startup script you want - if you want them to startup with the server)

```
nohup /admin/pknock/listend 2> /dev/null &
nohup /admin/pknock/checkd &
```

Knocking on the door

Ok, you have got the scripts running, now you want to test them. How do you do the actual "*knock*"? Well, there are many ways - you could use telnet, a port scanner where you can specify the ports, web browse, or anything

else to send packets to the ports. My favourite way is to use *netcat* because it is simple and you can get it for linux and windows. Under linux a simple command to "*knock*" would be..

```
for x in 111 2222 33333; do  nc -w 1 -z <server_ip> $x; done
```

Final Words

Port knocking is cool and fun, and does add another layer of security to addressing the risks of remote management access. But remember, it adds another layer thats all. If you implement port knocking you should still not run vulnerable versions of the server code, or use bad passwords, or any of the other fundamental rules of securing a service. As always, have fun and learn.

Updates

Well curious seekers, I have being playing around with some of the newer firewall modules and I have also had requests for an easier port knocking implementation. So try this using the new *recent* match for iptables (this does assume that the *mangle* and *nat PREROUTING* policy is *ACCEPT*)..

```
$IPT -N PKNCK
$IPT -N SSHKNCK
$IPT -A PKNCK -m state --state NEW -m tcp -p tcp --dport 22 -m recent --seconds 300 --rcheck --name
ALLOW -j ACCEPT
$IPT -A PKNCK -m state --state NEW -m tcp -p tcp -m recent --name ALLOW --remove -j DROP
$IPT -A PKNCK -m state --state NEW -m tcp -p tcp --dport 33333 -m recent --rcheck --name KEEP -j
SSHKNCK
$IPT -A PKNCK -m state --state NEW -m tcp -p tcp -m recent --name KEEP --remove -j DROP
$IPT -A PKNCK -m state --state NEW -m tcp -p tcp --dport 2222 -m recent --name KEEP --set -j DROP
$IPT -A PKNCK -m state --state NEW -m tcp -p tcp --dport 111 -m recent --name KEEP --set -j DROP
$IPT -A PKNCK -j RETURN
$IPT -A SSHKNCK -m recent --name ALLOW --set -j DROP
$IPT -A INPUT -p tcp -m tcp -j PKNCK
```