

# **RPC**

## ***Your Friend or Foe?***

David Hoelzer  
SANS 2001

This page intentionally left blank.

# Contents

RPC Basics	4
Portmapper	13
Decoding RPC	18
Anomalous Traffic	30
Appendices	58

*RPCs - Friends or Foe?*

SANS 2001 - 2

This page intentionally left blank.

# What you should learn

- Understanding of what RPC's are
- Purpose of Portmap
- Transport Methods
- What RPC's should look like
- How to identify anomalous RPC traffic

*RPCs - Friends or Foe?*

SANS 2001 - 3

Welcome to the wonderful world of RPCs!

The purpose of this course is to provide you with a basic understanding of how RPCs function, what their purpose is and, most importantly, what they look like.

The idea is that if we can figure out how RPCs should look, we can identify “anomalous” RPCs, or watch for particular types of RPCs with a minimum of effort.

# RPC Basics

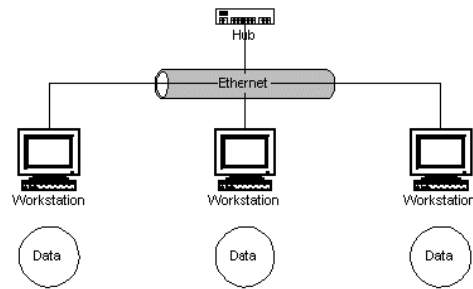
- What are RPC's anyway?
- What are the advantages to using them?
- How do they work?
- What sorts of RPC's are available?

*RPCs - Friends or Foe?*

SANS 2001 - 4

So now it's time to get to work. We'll see if we can lay a groundwork of understanding for the RPC paradigm, explain why it can be a powerful and valuable service, how they work and what sorts of things are available through RPC calls.

# What are RPCs for?



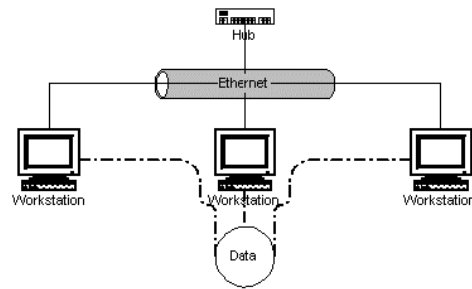
*RPCs - Friends or Foe?*

SANS 2001 - 5

Imagine for a moment that you have a number of workstations, each of which needs to use a particular application to operate on a dataset. I know that this sounds antiquated, but we're trying to get the mindset of the persons who came up with RPCs.

Imagine the issues involved with trying to synchronize the dataset across all of the hosts. You would very shortly be motivated to come up with some sort of distributed file system to simplify your life. However, since we're talking about programmer types here, of course we're going to go for the all-encompassing universal solution!!

# What are RPCs for?



*RPCs - Friends or Foe?*

SANS 2001 - 6

In this slide, we have a picture of the service that we very very quickly realize that we need to have to facilitate easy maintenance of our data set. But there's actually much more to this! The "Data" that is depicted beneath the central workstation, using RPCs, could be virtually ANY resource!

These resources thus become platform independent. In other words, we could create a solution to a programming problem on one platform, design it as an RPC, and then access that RPC from any other platform that supports RPCs!

# RPC Basics

- RPC : Remote Procedure Call
- Provides transparent service to programmer

*RPCs - Friends or Foe?*

SANS 2001 - 7

This page intentionally left blank.

# RPC Basics

## Advantages

- Standard high level interface to services.
- Easy way of providing a single point of service in a networked environment
- Good framework for Distributed Computing Environments

*RPCs - Friends or Foe?*

SANS 2001 - 8

Some persons coming into this class may have the notion, “RPCs are all bad. Why don’t you just not use them?” As we can see from this brief discussion, though, RPCs fill a very important need in the distributed computing environment.

# RPC Basics

How do they work?

- A local program makes a call to a “stub” function.
- Stub function packs the data up and ships it to an appropriate server
- Server processes the request and ships back the response to the client
- The stub function returns the unpacked data to the calling program

*RPCs - Friends or Foe?*

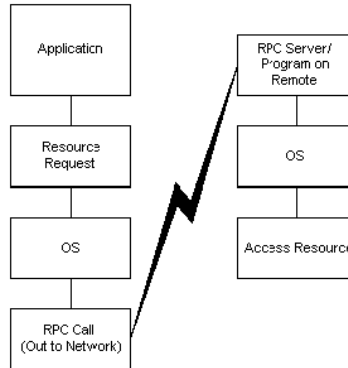
SANS 2001 - 9

We'll take a very brief look at how RPCs work internally. A basic understanding of this is necessary in order to understand the general flow of RPC traffic. At least, what that flow should look like!

By “Stub” function, what we mean is that the runtime library has a function defined in it that, in this case, exists only to forward the request on to somewhere else. The actual functionality exists somewhere else.

The four steps above are somewhat simplified. There is actually some more behind the scenes work going on when we use an RPC resource.

# RPC Basics



*RPCs - Friends or Foe?*

SANS 2001 - 10

This slide is a simple representation of what we discussed on the previous slide. The flow is from top to bottom, from left to right.

1. Our application has a need for some resource
2. This need is formulated into a function call for the resource (Resource Request)
3. The request is passed to the OS
4. Somewhere in the runtime library there is a realization that this is a “remote” resource. An RPC request is formulated and dropped on the wire
5. The RPC Server receives the request and forwards it on into the OS/runtime library
6. The resource is accessed

The result from the RPC is passed back in roughly the reverse order.

# RPC Basics

What sorts of RPC's are there?

- NFS
- Yellow Pages Services (ypasswd)
- rusers
- rquota
- NIS
- sadmind

*RPCs - Friends or Foe?*

SANS 2001 - 11

In truth, you can implement an RPC for absolutely anything. There is provision within the specification for "User" space RPCs.

# RPC Basics

How do we find RPC services on a system?

*RPCs - Friends or Foe?*

SANS 2001 - 12

But now the real question. What happens behind the scenes that allows us to find the right service on the remote machine?

# The Portmapper

- The portmapper maintains a directory of available RPC services on a system
- Allows a service to map to a particular version of an RPC program

*RPCs - Friends or Foe?*

SANS 2001 - 13

The portmapper is an essential part of the RPC paradigm. In fact, RPCs will not function without it. We'll go into more detail on it in the next few slides.

# The Portmapper

## How it works

Familiar error?

RPC Program not registered.

*RPCs - Friends or Foe?*

SANS 2001 - 14

If you've ever seen this error, you were probably trying to do a quick NFS share/mount to facilitate the transfer of a file system or something similar. Essentially, what this error is telling you is that the portmapper has no idea where the program you are requesting is because it has not "registered" itself as open for business.

When a program registers with the portmapper, it contacts the portmapper with the following information:

- Which program number it is (more on that later)
- Which versions/protocols are available
- Which port/ports it is listening on.

# The Portmapper

- All services must register. 'rpcinfo' can be used to see what's what.

```
snoopy# rpcinfo -p
program vers proto port service
100000 4 tcp 111 rpcbind
100000 3 tcp 111 rpcbind
100000 2 tcp 111 rpcbind
100000 4 udp 111 rpcbind
100000 3 udp 111 rpcbind
100000 2 udp 111 rpcbind
100021 1 udp 4045 nlockmgr
100024 1 udp 32772 status
100024 1 tcp 32771 status
```

*(Notes)*

*RPCs - Friends or Foe?*

**SANS 2001 - 15**

RPCInfo basically calls the "DUMP" pragma on the portmapper. This asks the portmapper to dump a list of all currently registered programs with their available versions and protocols.

Notice that there are six versions of "rpcbind" available. This is an internal name for "portmap". In reality, there are (most likely) not six separate daemons running on the system. More likely, the daemon that is running and has registered itself with the portmapper understands (and is listening) both UDP and TCP (and has bound both the UDP and the TCP ports required) as well as versions 2-4 of portmap.

What if I could insert my own RPC program into the portmap list?

The portmapper has 4 different possible functions that can be called:

**PMAPPROC\_SET:** Called to register a program number to a port number

**PMAPPROC\_UNSET:** Called to remove a program from the portmapper directory

**PMAPPROC\_GETPORT:** Called to retrieve the port number that a particular service and version lives on

**PMAPPROC\_DUMP:** Dumps a list of all programs, versions and port numbers

## Traffic Decodes (Anomalous)

```
00:06:55.753351 127.0.0.1.43862 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 55069)
00:18:55.780919 127.0.0.1.43877 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 54248)
00:30:55.644708 127.0.0.1.43894 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 53277)
00:42:55.590878 127.0.0.1.43909 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 52381)
00:54:55.535681 127.0.0.1.43927 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 51485)
01:04:55.443822 127.0.0.1.43939 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 61621)
01:14:55.400732 127.0.0.1.43953 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 6261)
01:24:55.356257 127.0.0.1.43966 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 16437)
01:34:55.309417 127.0.0.1.43981 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 26613)
```

*RPCs - Friends or Foe?*

**SANS 2001 - 16**

This isn't the anomalous decodes section, but since we're talking about the portmapper and registered services, we can't put this one off til later.

10.0.0.20 is an internet visible machine providing some RPC services. While we don't have any more data than the packet headers above, what sort of activity could we be looking at here?

RPC Services are inserted into the portmapper only from the local machine and require root privileges. Is this an attempt to either insert or delete a new service?

In this instance, an excellent correlation would be a subsequent request from a host for a dump to check their work. In this case, the site in question wasn't watching for port 111, they were watching for illegal IP addresses, so we can't be sure.

# The Portmapper

- Programs are identified by number. Clients request the information for a service based on this ID number.

0	-	1fffffff	Defined by Sun
20000000	-	3fffffff	User defined
40000000	-	5fffffff	Transient
60000000	-	ffffffff	Reserved

*RPCs - Friends or Foe?*

**SANS 2001 - 17**

This list comes from RFC 1050. While Sun is designated as the authority for RPC program numbers, they unfortunately have not produced an exhaustive list of the RPC programs that have been registered thus far. You can go to their website and request the information, but so far the only response I've ever received is "We're in the process of compiling the data and will forward it on when it is complete." I have found that various network information tools such as "Nmap" have a fairly comprehensive list. If you obtain the Nmap source distribution, you will find various RPC resource files enumerating most of the common services, what program # they are, and where they commonly live port wise.

The next slide has a list of some of the more common ones.

# The Portmapper

## RPC Service Numbers

100000	Portmap (rpcbind)
100001	rstatd
100002	rusersd
100003	nfsd
100005	mountd
100008	walld
100011	rquota
100012	sprayd
100021	nlockmgr (nfs locking manager)
100024	status
100026	bootparam
100227	nfs_acl
100232	sadmind (Sun admin daemon)

*RPCs - Friends or Foe?*

**SANS 2001 - 18**

Here are a few RPC Program numbers. While there are many many more, this is a list of a few of the most common ones that I have seen probes for.

# RPC Decode

What are RPCs supposed to look like?

*RPCs - Friends or Foe?*

SANS 2001 - 19

Now that we understand some of the theory behind RPCs and how they can be helpful, it's time to start taking a look at what actual RPC decodes look like at the byte level. The information that was used to put together the 'pictures' of what RPCs ought to look like comes from applying the RFC on the XDR (External Data Representation) language and the RPC RFC (which is specified using XDR). It would be a good idea to take some time to go through the XDR RFC and make sure you understand the basics of how an XDR specification works since nearly all RPCs for which there are RFCs are specified in this way.

This becomes important if you decide that you have a need to decode something further up the application layer. For instance, if you have an interest in monitoring for particular NFS result codes, you will need to work out the XDR definition of the NFS application.

# RPC Decode

- The RPC specification is written in XDR
- RFC 1057 - RPC Version 2
- RFC 1832 - XDR

*Notes*

*RPCs - Friends or Foe?*

**SANS 2001 - 20**

RFC 1057 defines the actual internal structure of an RPC call. The format provides a prototype style definition for where each piece of data falls within the header.

RFC 1832 defines the XDR for us. XDR is the “External Data Representation Standard”. Here are some important pieces to know:

Basic Data Size: 4 bytes (double word)

Integer	4 bytes
Enum	4 bytes
Boolean	4 bytes (enum { FALSE = 0, TRUE = 1 })
Hyper Int	8 bytes
Float	4 bytes (IEEE standard for normalized single precision)
Double	8 bytes
Quad	16 bytes
Opaque	4 byte length followed by bytes padded to 4 byte boundary
String	4 byte length followed by bytes padded to 4 byte boundary

# RPC Headers

```
+++++  
| Transaction ID (XID) |  
+++++  
| Message Type (Call, 0) |  
+++++  
| RPC Version |  
+++++  
| RPC Program |  
+++++  
| Program Version |  
+++++  
| Procedure Number |  
+++++  
|<----- Four Bytes Wide ----->|
```

*RPCs - Friends or Foe?*      **SANS 2001 - 21**

This format is consistent for ALL RPC calls. We're in for a wild ride for the rest though!

The RPC header will immediately follow the UDP header if a UDP transport is being used. However, if TCP is the transport layer, then the RPC header will be preceded by a 4 byte length field used to define the total length of the entire RPC that will follow.

# RPC Headers

```
+-----+
|                               |
|           Transaction ID     |
|-----+
|                               |
|           Message Type (Reply, 1) |
|-----+
|                               |
|           Status (Accepted = 0) |
|-----+
|                               |
|           Verifier (Up to 408) |
|-----+
|                               |
|           Accept Status      |
|-----+
|<----- Four Bytes Wide ----->|
```

*RPCs - Friends or Foe?*

**SANS 2001 - 22**

This format is consistent for ALL RPC **replies**. We're in for a wild ride for the rest though!

The RPC header will immediately follow the UDP header if a UDP transport is being used. However, if TCP is the transport layer, then the RPC header will be preceded by a 4 byte length field used to define the total length of the entire RPC that will follow.

We will not go into each of the various cases possible (the various Accept status reports), however an value other than zero for the status indicates an error condition of some kind. Additionally, if the Status is not equal to zero, the request has been refused or rejected. Either of these might be notable events.

# RPC Decode

```
10:24:46.396484 XXX.XXX.160.64.961453077 >
XXX.XXX.85.20.2049: 40 null (DF)
      4500 0044 fec8 4000 fd11 3486 xxxx a040
      xxxx 5514 b5e7 0801 0030 9b40 394e 9c15
      0000 0000 0000 0002 0001 86a3 0000 0003
      0000 0000 0000 0000 0000 0000 0000 0000
      0000 0000
```

5 double words for IP = 20 bytes

8 byte UDP header with a length field of 48

394e 9c15	XID	0000 0000	Message Type (CALL)
0000 0002	RPC Version	0001 86a3	RPC Program
0000 0003	Program Vers	0000 0000	Procedure #
0000 0000	Authentication		

*RPCs - Friends or Foe?*

SANS 2001 - 23

Here we have a full decode of the RPC header in a UDP packet. When first seeing this packet, it might strike us as anomalous. Notice the source port number that TCPDump comes up with.

Decoding the packet, however, we find that there is apparently nothing wrong with it. If TCPDump is able to determine based on the packet contents that it is looking at an RPC that it can understand, will attempt to decode the RPC. If it is able to do so, it will replace the source port number with the RPC Transaction ID.

With an understanding of how to decode an RPC request header, we are able to determine which particular service an attacker is seeking. While the port numbers for common services do tend to be fairly constant, the port assignments are not part of the RPC specification. This means that the port number is subject to change at any time. For instance, if some other service had already bound port 2049 in the server, some other port would have been picked. This is where the port mapper comes in for regular network traffic.

Many sites watch for probes to their portmapper, but how many watch for someone to go right to the common port for the service?

## RPC Decode (UDP Filter)

Please take a moment and design a TCPDump filter to capture all UDP packets that might be RPC version 2 Calls. Please feel free to collaborate.

*RPCs - Friends or Foe?*

SANS 2001 - 24

Using the definition that we already have of an RPC call header, it should be possible for you to create a TCPDump filter to capture all traffic that fits into our definition.

# RPC Decode

Here's a nice, normal, happy reply:

```
10:24:22.378560 XXX.XXX.85.20.2049 > XXX.XXX.160.49.960793180: reply  
ok 24 null (DF)
```

```
4500 0034 9bb4 4000 fe11 96bb xxxx 5514  
xxxx a031 0801 85c9 0020 63f9 3944 8a5c  
0000 0001 0000 0000 0000 0000 0000 0000  
0000 0000
```

*RPCs - Friends or Foe?*

**SANS 2001 - 25**

3944 8a5c	XID (transaction ID)
0000 0001	RPC Reply (1)
0000 0000	Message Accepted (0)
0000 0000	Opaque Verification (Null in this field means nothing follows)
0000 0000	Success! (the accepted "tag" is 0)
0000 0000	Opaque Results of zero length (pads into any other higher level response)

This is the complete decode of the RPC portion of this packet. We can see that additional filters could be built for our IDS to target accepted RPC requests (which could be more alarming if we're not serving them intentionally!). This is where egress filtering and monitoring are soooooo important.

We noticed an NFS volume shared out of our protected network. It turns out that this volume had been used years ago during a collaborative project and had never been unmounted and removed from the exports table on the host. Tagging mysterious RPCs through our DMZ highlighted the problem.

# RPC Decode

Assuming we have a failed/denied RPC:

Here's the hex dump from a failed RPC request:

```
4500 0034 9bb4 4000 fe11 96bb xxxx 7732
xxxx 8821 0801 85c9 001c 63f9 3944 8a5c
0000 0001 0000 0001 0000 0001 0000 0004
```

$udp[12:4] = 1$  and  $udp[16:4] = 1$  and  $udp[20:4] = 1$

*RPCs - Friends or Foe?*

SANS 2001 - 26

This hex dump shows an UDP packet (type 0x11) with no IP options (length 5) of length 0x1c (28 total bytes including the header). The RPC header is as follows:

```
3944 8a5c      XID (Transaction ID)
0000 0001      Message Type (1 = reply)
0000 0001      Reply Type (1 = rejected)
0000 0001      Rejection reason (1 = authentication error)
0000 0004      Authentication error (4 = verifier expired or replayed)
```

What makes RPC headers trickier than normal network headers is that the header format changes depending upon the discriminants in the unions. However, could we build an IDS filter to capture RPC replies that failed authentication? We have everything we need for a signature:

$Udp[12:4] = 1$  and  $udp[16:4] = 1$  and  $udp[20:4] = 1$

# Quick Quiz

## What's happening here:

23:00:26.743193 adc083.internal.com.33354 > ratbert.internal.com.32773:  
udp 164 (DF) (ttl 253, id 28585)

```
4500 00c0 6fa9 4000 fd11 c311 cccc a053
cccc 551b 824a 8005 00ac 6436 39cf e1eb
0000 0000 0000 0002 0001 87cc 0000 0003
0000 0005 0000
```

23:00:26.745514 ratbert.internal.com.32773 > adc083.internal.com.33354:  
udp 20 (DF) (ttl 254, id 40567)

```
4500 0030 9e77 4000 fe11 93d3 cccc 551b
cccc a053 8005 824a 001c 9735 39cf e1eb
0000 0001 0000 0001 0000 0001 0000 0001
```

*RPCs - Friends or Foe?*

**SANS 2001 - 27**

Looking at the RPC code in the slide, please identify the following and answer the following questions:

- 1) Transaction ID: \_\_\_\_\_
- 2) RPC Program Number: \_\_\_\_\_
- 3) RPC Program Name: \_\_\_\_\_
- 4) Is this a Call or a Response: \_\_\_\_\_
- 5) Which protocol is being used? UDP / TCP
- 6) Which RPC Version is being used: \_\_\_\_\_
- 7) Should it concern us that the IP ID number is different in the 2 packets? \_\_\_\_\_
- 8) Are these packets showing stimulus and response or are they unrelated? \_\_\_\_\_

## Quick Quiz #2

```
00:53:50.706574 hornet.32777 > altair.smsc.com.60900: udp
20 (DF) (ttl 253, id 25331)
```

```
4500 0030 62f3 4000 fd11 d08f aa81 a02a
aa81 550c 8009 ede4 001c 8d0d 39ba 80bf
0000 0001 0000 0001 0000 0001 0000 0004
```

```
12:58:51.445325 hornet.32777 > altair.smsc.com.60900: udp
20 (DF) (ttl 253, id 8387)
```

```
4500 0030 20c3 4000 fd11 12c0 aa81 a02a
aa81 550c 8009 ede4 001c 8d0b 39ba 80c1
0000 0001 0000 0001 0000 0001 0000 0004
```

*RPCs - Friends or Foe?*

SANS 2001 - 28

Let's try this again now that we've got the answers to the first quiz...

Looking at the RPC code in the slide, please identify the following and answer the following questions:

- 1) Transaction ID: \_\_\_\_\_
- 2) RPC Program Number: \_\_\_\_\_
- 3) RPC Program Name: \_\_\_\_\_
- 4) Is this a Call or a Response: \_\_\_\_\_
- 5) Which protocol is being used? UDP / TCP
- 6) Which RPC Version is being used: \_\_\_\_\_
- 7) Should it concern us that the IP ID number is different in the 2 packets? \_\_\_\_\_
- 8) Are these packets showing stimulus and response or are they unrelated? \_\_\_\_\_

# Secure RPC/Portmapper

Secure RPC deals with the type of authentication and possibly an encryption method across the RPC body

Secure Portmapper deals with adding access control to your portmapper

*RPCs - Friends or Foe?*

SANS 2001 - 29

This is as good a time as any to bring up some additional features that we may wish to implement with our RPC calls, especially if these systems will be publicly available. Secure RPC implements additional authentication and offers encryption options for the body of the RPC calls/replies. This goes a long way toward insuring that a request is actually from who it appears to be from. The default authentication method in RPC is None, though most common RPC programs (NFS for instance) will default to using the UID and GID as the authentication tokens. These are easily forged. Using stronger authentication and encryption can help to protect you from replay attacks or even silent file snarfing using a tool like “filesnarf” that comes with Dsniff.

Secure Portmapper does **not** secure your RPCs! It implements TCP Wrappers style authentication in the portmapper. While this is a good start, it is not sufficient defense against a skilled and determined attacker.

## Traffic Decodes (Anomalous)

With an understanding of how RPCs ought to look and an idea of how the entire structure functions, we're prepared to examine some anomalous patterns.

- Attacker Strategies:
  - ◆ Attack the port mapper
  - ◆ Compromise the port mapper
  - ◆ Probe for services directly (more stealthy)
  - ◆ Compromise services

*RPCs - Friends or Foe?*

SANS 2001 - 30

We need to put ourselves into the shoes of those malcontents who are out to exploit any hole that they can find in our systems. The types of probes that we detect can tell us a great deal about who is attacking us.

- Attacking the Portmapper: This strategy, while common, is very **loud**. Not your most sophisticated operator. (Doesn't HAVE to be...) Very efficient for large scale RPC scans...
- Compromising the portmapper: A person attempting to compromise the portmapper in some way, perhaps adding or deleting services (which could potentially create a DOS), is likely a bit more sophisticated than the portmap dumpers.
- Probe for service directly: These are ones to watch. If we see someone probing the commonly bound port for an RPC service, they have something very particular in mind for us or anyone that they can compromise.
- Compromise services: This person comes in two flavors. It could be the script kiddy who just ran a scan for everything with a port mapper and then targetted your machine just because it answered. Potentially, though, you could be looking at your more sophisticated hacker who saw the specs (or wrote them!) for the latest RPC exploit and has put together the tools to exploit it. These guys might look for the portmapper or they might go right for the jugular.

## Traffic Decodes (Anomalous)

If we attack the portmapper, chances are high that someone will notice that we're knocking on the door.

Portmapper is just another RPC!

*RPCs - Friends or Foe?*

SANS 2001 - 31

This page intentionally left blank.

## Traffic Decodes (Anomalous)

```
00:06:55.753351 127.0.0.1.43862 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 55069)
00:18:55.780919 127.0.0.1.43877 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 54248)
00:30:55.644708 127.0.0.1.43894 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 53277)
00:42:55.590878 127.0.0.1.43909 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 52381)
00:54:55.535681 127.0.0.1.43927 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 51485)
01:04:55.443822 127.0.0.1.43939 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 61621)
01:14:55.400732 127.0.0.1.43953 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 6261)
01:24:55.356257 127.0.0.1.43966 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 16437)
01:34:55.309417 127.0.0.1.43981 > 10.0.0.20.111: udp 56 (DF) (ttl
 248, id 26613)
```

*RPCs - Friends or Foe?*

**SANS 2001 - 32**

This one should look familiar. We saw it way back when we talked about the portmapper.

10.0.0.20 is an internet visible machine providing some RPC services. While we don't have any more data than the packet headers above, what sort of activity could we be looking at here?

RPC Services are inserted into the portmapper only from the local machine and require root privileges. Is this an attempt to either insert or delete a new service?

In this instance, an excellent correlation would be a subsequent request from a host for a dump to check their work. In this case, the site in question wasn't watching for port 111, they were watching for illegal IP addresses, so we can't be sure.

## Active System Attack Alerts

==--==--==--==--==--==--==--==--==

Dec 26 09:27:41 ext portsentry[12857]: attackalert:

SYN/Normal scan from

host: 172.20.20.1 to TCP port: 111

Dec 26 09:27:41 ext portsentry[12857]: attackalert:

Host 172.20.20.1 has

been blocked via wrappers with string: "ALL:

172.20.20.1 : DENY"

Dec 26 09:27:41 ext portsentry[12857]: attackalert:

Host 172.20.20.1 has

been blocked via dropped route using command:

"/sbin/ipchains -I input -s

172.20.20.1 -j DENY -I"

Here's a knock, knock, knock on the door and someone's active alert leaps into action and blocks the probing machine from probing the portmapper. Would the active alert have caught this probe if it had gone right for the ToolTalk service???

We're also seeing active response in action here. There are significant hazards in enabling automated active response. For the attacker, it becomes immediately apparent that a machine is actively defending itself in this way.

## Which system trace correlates with the Snort trace?

---

```
Feb 21 18:59:03 dns3 rpcbind: refused connect from 159.226.8.2 to dump()
```

```
Feb 21 18:59:09 dns3 rpcbind: refused connect from 159.226.8.2 to  
getport(1000d)
```

---

```
[**] RPC - portmap-request-cmsd [**]  
02/21-18:59:09.518264 159.226.8.2:33553 -> x.x.x.z:111  
UDP TTL:242 TOS:0x0 ID:60474 DF  
Len: 64  
38 B4 8B 69 00 00 00 00 00 00 02 00 01 86 A0 8..i.....  
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 .....  
00 00 00 00 00 00 00 00 00 01 86 E4 00 00 00 04 .....  
00 00 00 11 00 00 00 00 .....  
.....
```

*RPCs - Friends or Foe?*      **SANS 2001 - 34**

The answer is the getport() of course, if nothing else match up the time stamps. February 2000 saw a series of attacks directly against RPC services including the popular getport(mountd), to get the data to directly mount an attack against mountd.

The getport() call sends the program number of the desired service to the portmapper, which will then reply with the port number to which that service is currently bound. Can you see the advantage of going directly after the service? If an active alert could trigger the port to be blocked, why not go right after what we want??

# Traffic Decodes (Anomalous)

Look what we have here:

```
/var/adm/messages:Jan 26 17:10:22 kitty inetd[125]:  
100083[394] from 10.92.239.116 44792
```

*RPCs - Friends or Foe?*

SANS 2001 - 35

This one's straight off of the GIAC pages. This came in the February 6, 2000 1400 edition. Here we have a request to access what looks like an RPC program number. What we **don't** see is a portmapper Getport request preceding it. A more sophisticated attacker, perhaps?

While Sun has been less than forthcoming with their promised list of RPC program numbers, the developers of Nmap are far more generous. Here's a section of the "nmap-rpc" file. Also of note is the "nmap-services" file that comes with the standard distribution:

```
bugtraqd          100071  
kerbd             100078  
ttdbserverd      100083  rpc.ttdbserverd ttdbserverd ttdbserver  
tooltalk  
admin            100087  
autofs           100099  autofs  
event            100101  na.event          # SunNet Manager
```

# Traffic Decodes (Anomalous)

## Failure to Correlate can kill us.

```
12/21-09:50:13.447450 source:753 -> target:111
UDP TTL:128 TOS:0x0 ID:8846
Len: 64
39 5B 72 44 00 00 00 00 00 00 02 00 01 86 A0 9[rD.....
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 01 86 A5 00 00 00 00 .....
00 00 00 11 00 00 00 00 .....
```

Prelude to invasion?? What's going on here?

*RPCs - Friends or Foe?*

SANS 2001 - 36

While we haven't spoken about the portmapper specific functions, it operates under the standard RPC framework. So what are we looking at here?

395b 7244	XID
0000 0000	Request
0000 0002	RPC Version #
0001 86a0	Program 100000 (portmap)
0000 0002	Program Version #
0000 0003	Procedure (getport())
0000 0000	Auth Credentials
0000 0000	Auth Verification
blah blah blah...	
0001 86a5	Program 100005 (Mountd)

Some versions of mountd (linux especially) are exploitable. This particular dump is the basis for the signature in the Arachnids database for the Millenium worm. (IDS13 from [www.whitehats.com](http://www.whitehats.com))

# Traffic Decodes (Anomalous)

## Circumventing the Portmapper

```
12/22-13:27:45.962414 source:780 -> target:32776
TCP TTL:64 TOS:0x0 ID:60306 DF
***PA* Seq: 0x6EF54A16 Ack: 0xE5E01EC7 Win: 0x7D78
TCP Options => NOP NOP TS: 10138892 356693684
80 00 00 5C 32 40 30 A3 00 00 00 00 00 00 02 ... \2@0.....
00 01 86 A5 00 00 00 01 00 00 00 05 00 00 01 .....
00 00 00 34 38 61 42 51 00 00 00 03 64 65 76 00 ... 48aBQ....dev.
00 00 00 00 00 00 00 00 00 00 00 07 00 00 00 00 .....
00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 04 .....
00 00 00 06 00 00 00 0A 00 00 00 00 00 00 00 00 .....
```

*RPCs - Friends or Foe?*

**SANS 2001 - 37**

Another excellent RFC to check out is the NFS Version 3. It's quite lengthy, but if you take a look through it for stuff that you just know is bad, it becomes easier to look at a dump like the above and recognize it for what it is. Here we have a request to program 100005 (0x000186a5). In particular, this is a request for show mounts.

This trace also comes straight from the Arachnids database. What is of special interest for us here is the destination port number. How many of us are blocking port 32776 (or anything around it for that matter). Would this packet, if directed at our web server, for instance, via UDP or TCP, have triggered any alarms internally? While the scanner isn't guaranteed that mountd will live at 32776, it's fairly common.

## Traffic Decodes (Anomalous)

```
Jun 24 05:41:24 dns2 snort[336]: IDS013 - RPC - portmap-request-mountd:  
12.36.232.227:763 -> z.y.w.66:111  
Jun 24 05:41:24 dns2 snort[336]: IDS26 - NFS Showmount: 12.36.232.227:764 ->  
z.y.w.66:32776
```

```
-----  
[**] IDS013 - RPC - portmap-request-mountd [**]  
06/24-05:41:23.838139 12.36.232.227:763 -> z.y.w.66:111 UDP  
TTL:51 TOS:0x0 ID:63764 Len: 64 50 67 15 1B 00 00 00 00  
00 00 00 02 00 01 86 A0 Pg.....  
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00 .....  
..... 00 00 00 00 00 00 00 00 00 01 86 A5 00 00 00 01  
..... 00 00 00 06 00 00 00 00 .....
```

*RPCs - Friends or Foe?*

**SANS 2001 - 38**

Here's an interesting detect. This detect comes from the GIAC daily report from late June. The first two lines look like they came from a Syslog while the second is from the Snort alerts file (hence the rounded time stamps on the first two). What should scare us about this detect is the fact that the portmap request was followed shortly thereafter by an NFS show mount request to port 32776. While we don't have any log files from this, it quite likely indicates that the portmapper answered the mountd request quite happily, directing the intruder to the mountd program which is then queried for showmounts.

Again the question, though. What if the savvy attacker went straight for the service he was interested in, completely bypassing the portmapper. It's true that he would always hit, but he will likely avoid detection more often than not.

## Traffic Decodes (Anomalous)

```
Jun 24 11:50:54 dns3 snort[565]: IDS013 - RPC -  
  portmap-request-mountd: 12.36.232.227:822 -> z.y.w.98:111  
Jun 24 11:50:54 dns3 snort[565]: IDS013 - RPC -  
  portmap-request-mountd: 12.36.232.227:823 -> z.y.w.98:111
```

```
-----  
[**] IDS013 - RPC - portmap-request-mountd [**]  
06/24-11:50:53.860434 12.36.232.227:822 -> z.y.w.98:111 UDP  
  TTL:52 TOS:0x0 ID:5862 Len: 64 0F 54 DA 14 00 00 00 00 00  
  00 00 02 00 01 86 A0 .T.....  
  00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00 .....  
  ..... 00 00 00 00 00 00 00 00 00 01 86 A5 00 00 00 01 ...  
  ..... 00 00 00 06 00 00 00 00 .....
```

*RPCs - Friends or Foe?*

**SANS 2001 - 39**

This detect is actually a correlation that goes with the previous slide. Here we have the same reporter with a detect on the same subnet (evidently.. It's hard to tell with the z.y.w) but a different machine.... But 6 hours later. Having seen the previous slide, we can deduce some information about the targetted network (as no doubt did the attacker). Since there was no call to mountd, it looks as though this machine did not respond to the portmapper request for mountd. My guess is that this machine is not running portmap and the ICMP unreachable are filtered going outbound. The low source port numbers may indicate a windows based host, an Xterminal or perhaps a host that has been root compromised already.

# Traffic Decodes (Anomalous)

## NFS Action????

```
02:08:24.349901 XXX.XXX.50.11.2049 > 38.8.5.2.53: 43661+ (44)
(ttl 63, id 48365)
```

```
4500 0048 bced 0000 3f11 b721 XXXX 320b
2608 0502 0801 0035 0034 8d7a aa8d 0100
0001 0000 0000 0000 0869 6e74 6572 6e61
6c04 4141 4141
```

```
02:08:24.464675 38.8.5.2.53 > XXX.XXX.50.11.2049: 43661
NXDomain q: internal.host 0/1/0 (119) (DF) (ttl 249, id
3980)
```

```
4500 0093 0f8c 4000 f911 6a37 2608 0502
XXXX 320b 0035 0801 007f 2f55 aa8d 8183
0001 0000 0001 0000 0869 6e74 6572 6e61
6c04 4141 4141
```

*RPCs - Friends or Foe?*

**SANS 2001 - 40**

We need to be careful how quickly we react to detects sometimes. For instance, we might see the port number 2049 here and immediately start setting off alarm bells and mailing site contacts demanding to know why they are mounting drives on one of your mail servers, not to mention internal missives to the people maintaining the mail server to find out why they're running NFS on it.

All the while we've taken the simple and made it the difficult. What we really have here is a simple DNS request. If we don't believe the TCPDump DNS decode info, look at the hex dump. You should be able to readily recognize that this is clearly not an RPC.

## Traffic Decodes (Anomalous)

Site: external Host lookup: Date: 20000706 Pattern: port 111  
/usr/local/logger/one\_day\_pat.pl -x -d 20000706 -l external -p 'port 111 '

15:55:17.021889 midgaard.PROTECTED.COM.64477 > teleport.ltx-tr.com.sunrpc: udp 88  
(ttl 27, id 99)

```
4500 0074 0063 0000 1b11 0894 XXXX 3278
9921 2068 fbdd 006f 0060 6875 0001 3762
0000 0000 0000 0002 0001 86a0 0000 0002
0000 0003 0000
```

15:55:20.013561 midgaard.PROTECTED.COM.64477 > teleport.ltx-tr.com.sunrpc: udp 88  
(ttl 27, id 100)

```
4500 0074 0064 0000 1b11 0893 XXXX 3278
9921 2068 fbdd 006f 0060 6875 0001 3762
0000 0000 0000 0002 0001 86a0 0000 0002
0000 0003 0000
```

*RPCs - Friends or Foe?*

**SANS 2001 - 41**

There's some good news and some bad news in this detect. This came in from my DMZ sensor on its way out of the network. What that means in my case is that the packet has successfully navigated its way out of the secure network and is now free to go anywhere it likes. (This, by the way, is a detect that shows the value of having a sensor way out beyond your perimeter where you can see what bad stuff is trying to get in and failing as well as picking up things that are escaping that you **thought** were being blocked somewhere) What makes this especially bad news is that we have no shared RPC services with LTX. They manufacture test equipment for IC's. The site from which this detect comes is involved with the production and testing of VLSI chips. Running this down, it turned out that the test floor had recently obtained several new LTX testers on a special end of the quarter deal from LTX. The systems were supposed to be new, however, this machine attempting to phone home seems to indicate that it used to live somewhere else...

What other hidden features are in our software that could be escaping our network undetected?

## Traffic Decodes (Anomalous)

```
08/03-11:15:26.051255  [**] Attempted Sun RPC high port
      access [**] 205.188.153.111:4000 -> MY.NET.217.126:32771
08/03-11:16:25.969658  [**] Attempted Sun RPC high port
      access [**] 205.188.153.111:4000 -> MY.NET.217.126:32771
08/03-11:17:25.864701  [**] Attempted Sun RPC high port
      access [**] 205.188.153.111:4000 -> MY.NET.217.126:32771
08/03-11:19:25.608726  [**] Attempted Sun RPC high port
      access [**] 205.188.153.111:4000 -> MY.NET.217.126:32771
08/03-11:20:25.541202  [**] Attempted Sun RPC high port
      access [**] 205.188.153.111:4000 -> MY.NET.217.126:32771
08/03-11:21:25.411544  [**] Attempted Sun RPC high port
      access [**] 205.188.153.111:4000 -> MY.NET.217.126:32771
```

*RPCs - Friends or Foe?*

**SANS 2001 - 42**

Here's a nice new one hot off the press. This is another example of when your newly honed RPC decoding skills will come in handy.

This one came into Giac on August 4<sup>th</sup>, 2000. Unfortunately, we have no other information on this one, but that port 4000 gives me a sneaky suspicion that we're looking at something that may be ICQ related. Without a hex dump, it's almost impossible to be certain.

## Traffic Decodes (Anomalous)

```
11:04:30.367702 out.side.mach.61.963737197 >  
  an.nfs.server.20.2049: 40 null (DF) (ttl 253, id  
  31166)  
      4500 0044 79be 4000 fd11 b995 xxxx xx3d  
      xxxx xx14 dd88 0801 0030 9927 3971 766d  
      0000 0000 0000 0002 0001 86a3 0000 0003  
      0000 0000 0000
```

(Exercise)

*RPCs - Friends or Foe?*

**SANS 2001 - 43**

What have we here? What jumps out at first glance is the nice bold “Null”. This NULL isn’t meant to indicate that this packet contains no data, rather we’re seeing a “normal” packet in an NFS session. TCPDump was working very happily trying to decode the NFS session data, but there wasn’t enough data captured to determine what’s happening in this particular packet except to see that it’s a request.

As an exercise for yourself, please take a look at the packet above and figure out how we know that this is an RPC request.

## Traffic Decodes (Anomalous)

### Remember our UDP Filter?

```
14:28:47.554126 204.34.198.40.123 > a.good.machine.120.64770:
v1 server strat 1 poll 6 prec -16 dist 0.000000 disp
0.000030 ref USNO@3174143276.283615112 orig
3174143277.626258850 rec -0.927163124 xmt -0.926677703 (ttl
49, id 18625)
0x0000 4500 004c 48c1 0000 3111 d19b cc22 c628
0x0010 xxxx xx78 007b fd02 0038 13cd 0c01 06f0
0x0020 0000 0000 0000 0002 5553 4e4f bd31 952c
0x0030 489b 0000 bd31 952d a052 7fdf bd31 952cH
0x0040 b2f7 f000 bd31 952c b317 c000
```

*RPCs - Friends or Foe?*

**SANS 2001 - 44**

There is actually only one consistent false positive that I have seen with the filter described earlier. NTP Time requests. However, it is possible to add discriminants to our filter statement to avoid these. For instance, in this case, the USNO is actually prominent in the hex:

**5553 4e4f : USNO**

# Traffic Decodes (Anomalous)

## Root Exploit! Which service?

```
10:53:07.629833 bad.boy.83.252.710 > our.host.50.11.932: u
dp 384 (ttl 64, id 62899)
0x0000 4500 019c f5b3 0000 4011 a893 xxxx 53fc E.....@.....S.
0x0010 xxxx 320b 02c6 03a4 0188 bfa8 587d 1f69 ..2.....X}.i
0x0020 0000 0000 0000 0002 0001 86b8 0000 0001 .....
0x0030 0000 0002 0000 0000 0000 0000 0000 0000 .....
```

(Full decode in notes)

*RPCs - Friends or Foe?*

SANS 2001 - 45

(For full decode, see Appendix I)

# Traffic Decodes (Anomalous)

```
20:16:17.591120 ool-18bcd36.dyn.cable.net.64304 >
ftp.defender.com.sunrpc: udp 0 (ttl 51, id 29757)
0x0000 4500 001c 743d 0000 3311 421d cccc dc36 E...t=..3.B....6
0x0010 cccc 3203 fb30 006f 0008 32c7 0000 0000 ..2..0.o..2.....
0x0020 0000 0000 0000 0000 0000 0000 0000 .....

20:16:18.575286 ool-18bcd36.dyn.cable.net.64304 >
ftp.defender.com.sunrpc: udp 0 (ttl 51, id 33019)
0x0000 4500 001c 80fb 0000 3311 355f cccc dc36 E.....3.5_...6
0x0010 cccc 3203 fb30 006f 0008 32c7 0000 0000 ..2..0.o..2.....
0x0020 0000 0000 0000 0000 0000 0000 0000 .....

20:16:19.575718 ool-18bcd36.dyn.cable.net.64304 >
ftp.defender.com.sunrpc: udp 0 (ttl 51, id 16634)
0x0000 4500 001c 40fa 0000 3311 7560 cccc dc36 E...@...3.u`...6
0x0010 cccc 3203 fb30 006f 0008 32c7 0000 0000 ..2..0.o..2.....
0x0020 0000 0000 0000 0000 0000 0000 0000 .....
```

**Is there more to this story? Is there something wrong with the analyst's conclusion?**

*RPCs - Friends or Foe?*

**SANS 2001 - 46**

This capture came from a remotely monitored site. As you can see from the resolved names, the destination machine is an FTP server. An important factor in analyzing traffic sometimes is “Analyst Overload”. I don’t know about your organization, but in mine our security analysts also spend a good deal of their time dealing with other sorts of network planning, deployment and management as well as “firefighting”. Only a portion of their time is actually devoted to handling security incidents.

In this particular case, this probe came in in early October of 2000. The analyst on duty saw this report in the hourly shadow report:

```
20:16:17.591120 ool-18bcd36.dyn.cable.net.64304 > ftp.defender.com.sunrpc: udp 0 (ttl 51, id 29757)
20:16:18.575286 ool-18bcd36.dyn.cable.net.64304 > ftp.defender.com.sunrpc: udp 0 (ttl 51, id 33019)
20:16:19.575718 ool-18bcd36.dyn.cable.net.64304 > ftp.defender.com.sunrpc: udp 0 (ttl 51, id 16634)
```

He pulled the trace seen in the above slide and since he had some other major issues developing in the office, he dismissed this as a simple probe for portmappers. Since he saw only stimulus and no response, he quickly concluded that all was well, especially since he didn’t think there were any RPC services running on the ftp server.

Is there more to this story? Is there something wrong with the analyst's conclusion?

# Traffic Decodes (Anomalous)

## The next day...

```
10:16:59.780039 ool-18bcd36.dy.0 > ftp.defender.com.nfs: 0 proc-
930349079 (ttl 51, id 30453)
0x0000 4500 001c 76f5 0000 3311 3f65 cccc dc36 E...v...3.?e...6
0x0010 cccc 3203 fb32 0801 0008 2b33 0000 0000 ..2..2....+3....
0x0020 0000 0000 0000 0000 0000 0000 0000 0000 .....
10:17:00.774485 ool-18bcd36.dyn.cable.net.64306 >
ftp.defender.com.2049: udp 0 (ttl 51, id 743)
0x0000 4500 001c 02e7 0000 3311 b373 cccc dc36 E.....3..s...6
0x0010 cccc 3203 fb32 0801 0008 2b33 058f df70 ..2..2....+3...p
0x0020 5010 2238 ff46 0000 0500 0203 1000 P."8.F.....
10:17:01.775185 ool-18bcd36.dyn.cable.net.64306 >
ftp.defender.com.2049: udp 0 (ttl 51, id 10970)
0x0000 4500 001c 2ada 0000 3311 8b80 cccc dc36 E...*...3.....6
0x0010 cccc 3203 fb32 0801 0008 2b33 0000 0000 ..2..2....+3....
0x0020 7002 ffff 74e0 0000 0103 0301 0204 p...t.....
```

**What's up with the first line? What stimulus is missing?**  
*RPCs - Friends or Foe? SANS 2001 - 47*

The next day, this comes in:

```
10:16:59.780039 ool-18bcd36.dyn.cable.net.0 > ftp.defender.com.nfs: 0 proc-930349079 (ttl 51, id 30453)
10:17:00.774485 ool-18bcd36.dyn.cable.net.64306 > ftp.defender.com.2049: udp 0 (ttl 51, id 743)
10:17:01.775185 ool-18bcd36.dyn.cable.net.64306 > ftp.defender.com.2049: udp 0 (ttl 51, id 10970)
```

At first, I thought that the first line must have been a typo of some sort, but please look at the TCP port numbers in the hex dump. So the question is, then, why does TCPDump report port numbers of 0 for the source and the destination?

Another important question to consider is what stimulus is missing from these packets. It is easy to see these and to dismiss them since there is no obvious response, but does that lack of response tell us, or more importantly the intruder, something??

# Traffic Decodes (Anomalous)

The next, next day...

```
02:04:07.752572 ool-18bcd36.dyn.cable.net.64314 >
ftp.defender.com.sunrpc: SF 1434520070:1434520070(0) win 512 (ttl
51, id 113)
0x0000 4500 0028 0071 0000 3306 b5e8 cccc dc36 E..(.q..3.....6
0x0010 cccc 3203 fb3a 006f 5581 0a06 0000 0000 ..2..:..oU.....
0x0020 5003 0200 8139 0000 0101 080a 0051 P....9.....Q
02:04:07.753393 ftp.defender.com.sunrpc > ool-
18bcd36.dyn.cable.net.64314: S 2602735588:2602735588(0) ack
1434520071 win 9112 <mss 536> (DF) (ttl 254, id 13191)
0x0000 4500 002c 3387 4000 fe06 77cd cccc 3203 E..,3.@...w...2.
0x0010 cccc dc36 006f fb3a 9b22 97e4 5581 0a07 ...6.o.:..."U...
0x0020 6012 2398 186a 0000 0204 0218 0051 `.#..j.....Q
02:04:07.779352 ool-18bcd36.dyn.cable.net.64314 >
ftp.defender.com.sunrpc: R 1434520071:1434520071(0) win 0 (ttl
242, id 41084)
0x0000 4500 0028 a07c 0000 f206 56dc cccc dc36 E..(.|....V....6
0x0010 cccc 3203 fb3a 006f 5581 0a07 0000 0000 ..2..:..oU.....
0x0020 5004 0000 8337 0000 0101 080a 09f8 P....7.....
RPCs - Friends or Foe? SANS 2001 - 48
```

And then the next day... At this point, the analyst sent all of the data to me since it was pretty obvious that this attacker had some interest in our ftp server. The analyst also now saw potentially hazardous response to the stimulus here, a SYN/FIN. The subject of the email was:

Subject: SOMEONE TURNED ON THE PORTMAPPER ON THE FTP SERVER!!!!

Sometimes when we're overloaded, we forget simple facts. He had failed to recognize that the lack of a response from the FTP server on UDP 111 actually told the intruder what he wanted to know... Namely, there was an active portmapper on the system. In fact, if he hadn't sent this SYN/FIN, he likely could have gotten in another day or two of work before getting tagged as a real bad boy.

## Traffic Decodes (Anomalous)

```
11:08:25.009933 ool-18bcd36.dyn.cable.net.64328 >
ftp.defender.com.sunrpc: S 2874479527:2874479527(0) win 32120 <mss
1460,sackOK,timestamp 13042308 0,nop,wscale 0> (DF) (ttl 51, id
41225)
... (SYN/ACK ACK) ...
11:08:25.048545 ool-18bcd36.dyn.cable.net.64328 >
ftp.defender.com.sunrpc: P 2874479528:2874479572(44) ack 259445994
win 32120 <nop,nop,timestamp 13042312 5644474> (DF) (ttl 51, id
41227)
0x0000 4500 0060 a10b 4000 3306 d515 cccc dc36 E..`.@.3.....6
0x0010 cccc 3203 fb48 006f ab55 13a8 0f76 d4ea ..2..H.o.U...v..
0x0020 8018 7d78 0cee 0000 0101 080a 00c7 0288 ..}x.....
0x0030 0056 20ba 8000 0028 59bd f6a6 0000 0000 .V.....(Y.....
0x0040 0000 0002 0001 86a0 0000 0002 0000 0004 .....
0x0050 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

*RPCs - Friends or Foe?*

**SANS 2001 - 49**

Later that same morning, our friend returns. This time he pulls a full dump from our portmapper. Judging from the TCP, my guess is that he did a simple `rpcinfo -p ftp.defender.com`. This seemed a strange departure from his previous stealthy tactics. His direct probes for specific RPC services the day before still had my attention, though. We held off firing off a message to his ISP to wait and see what he was after.

## Traffic Decodes (Anomalous)

Later that night...

```
23:13:02.079582 ool-18bcdc36.dyn.cable.net.64354 >
ftp.defender.com.sunrpc: udp 56 (ttl 51, id 41485)
0x0000  4500 0054 a20d 0000 3311 1415 cccc dc36      E..T....3.....6
0x0010  cccc 3203 fb62 006f 0040 32de 5b1e 96b4      ..2..b.o.@2.[...
0x0020  0000 0000 0000 0002 0001 86a0 0000 0002      .....
0x0030  0000 0003 0000 0000 0000 0000 0000 0000      .....
0x0040  0000 0000 0001 86b8 0000 0001 0000 0011      .....
0x0050  0000 0000                                     ....
23:13:02.082855 ftp.defender.com.sunrpc > ool-
18bcdc36.dyn.cable.net.64354: udp 28 (DF) (ttl 254, id 33161)
0x0000  4500 0038 8189 4000 fe11 29b4 cccc 3203      E..8..@...)...2.
0x0010  cccc dc36 006f fb62 0024 c084 5b1e 96b4      ...6.o.b.$..[...
0x0020  0000 0001 0000 0000 0000 0000 0000 0000      .....
0x0030  0000 0000 0000 8004                                     .....
```

*RPCs - Friends or Foe?*

**SANS 2001 - 50**

Later that same night, in comes this request. Our friend has returned to a UDP portmap request, this time a GETPORT for service 000186b8 which works out to be 100024 (statd or status). I wonder what's going to come next??? We can see our portmapper happily reporting that statd is alive and kicking on port 8004 (32772) UDP.

# r00tm3n0w

```
21:13:02.124700 ool-18bcd36.dyn.cable.net.64355 > ftp.defender.com.32772:
udp 1112 (ttl 51, id 41486)
0x0000  4500 0474 a20e 0000 3311 0ff4 cccc dc36  E..t...3.....6
0x0010  cccc 3203 fb63 8004 0460 706b 3e99 bfc5  ..2..c...`pk>...
0x0020  0000 0000 0000 0002 0001 86b8 0000 0001  .....
0x0030  0000 0002 0000 0000 0000 0000 0000 0000  .....
0x0040  0000 0000 0000 0400 9090 9090 9090 9090  .....
0x0050  9090 9090 9090 9090 9090 9090 9090 9090  .....
0x0060  9090 9090 9090 9090 9090 9090 9090 9090  .....
0x0070  9090                                     ..
21:13:04.228168 ftp.defender.com.32772 > ool-18bcd36.dyn.cable.net.64355:
udp 32 (DF) (ttl 254, id 33162)
0x0000  4500 003c 818a 4000 fe11 29af cccc 3203  E..<..@...)...2.
0x0010  cccc dc36 8004 fb63 0028 b45e 3e99 bfc5  ...6...c.(^>...
0x0020  0000 0001 0000 0000 0000 0000 0000 0000  .....
0x0030  0000 0000 0000 0000 0000 0000  .....

```

**RPCs - Friends or Foe? SANS 2001 - 51**

Our friend from cable land now launches an exploit at our status daemon. Notice the '90909090' hex that repeats through the final part of what was captured in the request. 0x90 in Intel machine code is a NOP. Buffer overflow exploits generally overflow the stack, inserting their own malicious code to be executed. The NOP's are inserted because it's a tricky thing to figure out where exactly you need to jump to. The NOPs allow the exploiter to jump to anywhere in the neighborhood of the exploit code since the processor will zip through the NOPs and start executing the exploit itself.

In this particular case, our intruder would have benefited from a bit more reconnaissance since the machine being exploited was a Sparc, not an X86 based system.



# Scanning Tools

- Nmap
- Hping
- Nessus
- Satan
- Scripts!

*RPCs - Friends or Foe?*

SANS 2001 - 53

**Nmap:** Nmap is an excellent all around network/host scanner. The program has many options, including “stealth” and decoy scanning. This scanner also has a “scan RPC” option.

**Hping:** Hping is an excellent testing tool and can be a very good tool to use for “low and slow” scans since you have a very high degree of control over the packets sent. It performs TCP and UDP “pings”, I.e., stimulus and response probes and can be very useful for determining responses from systems based on various TCP code bits, etc.

**Nessus:** A very user friendly vulnerability scanner. The scanner functions by sending various tests to the target hosts in an attempt to identify vulnerable versions of software.

**Satan:** A much older cousin of Nessus, Satan also does vulnerability scanning. Unless you have a very stable network, permission and nothing critical happening, I’d advise against the highest “aggressive” scanning option. I’ve had it take down networks.

**Scripts:** Writing scanning scripts for RPCs is a fairly simple undertaking since users will normally have access to rpcinfo. Using this simple command it’s a pretty simple task to write a scanner of your own. The advent of Perl and other higher level scripting language with library facilities has made socket based scripting far easier as well.

# Perl Scanner

```
#!/usr/bin/perl
use Socket;
@start = (10,12,0,1);
@end = (190,29,0,254);
open(OUTPUT, ">scan_results");
for($a = $start[0]; $a <= $end[0]; $a++)
{
    for($b = $start[1]; $b <= $end[1]; $b++)
    {
        for($c = $start[2]; $c <= $end[2]; $c++)
        {
            for($d = $start[3]; $d <= $end[3]; $d++)
            {
                &probe("$a.$b.$c.$d");
            }
        }
    }
}
...
$remote = $_[0];
$iaddr = inet_aton($remote);
$port = 111;
```

*RPCs - Friends or Foe?*

SANS 2001 - 54

(See Appendix II for full source code)

# Scanning

Another script from a compromised host:

```
#!/usr/bin/perl
```

```
$start = $ARGV[0];
```

```
$end = $ARGV[1];
```

```
if($start eq "" or $end eq "") { die("Usage: rpcscan.pl start  
end"); }
```

```
($sa,$sb,$sc,$sd) = split(/\./, $start);
```

```
($ea,$eb,$ec,$ed) = split(/\./, $end);
```

*RPCs - Friends or Foe?*

SANS 2001 - 55

(See Appendix III for full source code)

# TCP RPCs

TCP RPCs prepend a 4 byte RPC length field to all  
RPCs

## TCPDump Filter

ip and ((tcp[tcp[12]\*4 + 8:4] = 0 or tcp[tcp[12]\*4 + 8:4] =  
1) and tcp[tcp[12]\*4+12:4] = 2)

*RPCs - Friends or Foe?*

SANS 2001 - 56

Since TCP segments can be arbitrary sizes (that is, larger than the maximum packet size... There is of course a limit), RPC inserts an additional length field before the XID in the RPC header.

In order to use the filter above, you MUST increase your default snaplength. Otherwise you will never match any packets!

## Lessons Learned

- RPC programs may live where we least expect them
- If we *must* run RPC services, it is possible to flag anomalous activity with network analysis
- While RPC can be secured, we need to examine what sort of authentication is being applied in our organization
- Relying on someone coming in the front door (portmapper) can lead to a bad, bad day.

*RPCs - Friends or Foe?*

SANS 2001 - 57

This page intentionally left blank.

# RPC Program Numbers

The following list of program numbers in the notes pages comes from the Nmap 2.53 distribution. For the official list, contact [rpc@sun.com](mailto:rpc@sun.com), but be prepared for a very, very long wait.

*RPCs - Friends or Foe?*

SANS 2001 - 58

(For a list of RPC Program numbers, please see Appendix IV)

# Course Revision History

*RPCs - Friends or Foe?*

SANS 2001 - 59

v1.5 – David Hoelzer, Feb 2001

V1.6 – David Hoelzer, Mar 2001 – slide reordering, new audio