Analyzing Distributed Denial Of Service Tools: The Shaft Case

Sven Dietrich – NASA Goddard Space Flight Center
Neil Long – Oxford University
David Dittrich – University of Washington

ABSTRACT

In this paper we present an analysis of Shaft, an example of *malware* used in distributed denial of service (DDoS) attacks. This relatively recent occurrence combines well-known denial of service attacks (such as TCP SYN flood, smurf, and UDP flood) with a distributed and coordinated approach to create a powerful program, capable of slowing network communications to a grinding halt.

Denial of service attack programs, *root kits*, and network *sniffers* have been around in the computer underground for a very long time. They have not gained nearly the same level of attention by the general public as did the Morris *Internet Worm* of 1988, but have slowly progressed in their development. As more and more systems have come to be required for business, research, education, the basic functioning of government, and now entertainment and commerce from people's homes, the increasingly large number of vulnerable systems has converged with the development of these tools to create a situation that resulted in distributed denial of service attacks that took down the largest e-commerce and media sites on the Internet.

In contrast, we provide a comparative analysis of several distributed denial of service tools (e.g., Trinoo, TFN, Stacheldraht, and Mstream), look at emerging countermeasures against some of these tools. We look at practical examples of these techniques, provide some examples from test environments and finally talk about future trends of these distributed tools.

Introduction

Network-based attacks are nothing new, but up to last year the techniques utilized were focused on simple point-to-point denial of service. By denial of service we mean overwhelming the victim host or network to the point of unresponsiveness to the legitimate user. We provide a little overview, by no means complete, of previous point-to-point denial of service techniques. There are four major point-to-point techniques: TCP SYN flooding, UDP flooding, ICMP flooding, and Smurf attacks. The first one misbehaves from the standard three-way TCP handshake causing resource consumption and bandwidth consumption, whereas the remaining ones intend to consume the victim's bandwidth.

The year 1999 saw an emergence of new denial of service tools. The change was inevitable: the growth of network pipes made simple point-to-point tools either useless, or the improved tracking capabilities easily shut down the source of the problem. Even though some solutions or at least containment methods exist for the above, the distributed variants as an evolution of coordinated many-to-one attacks escape the traditional model sufficiently. Rather than relying on a single source, attackers could now take advantage of some hundred, thousand, even ten thousand or more systems to inflict denial of service onto their victims.

Analysis

The DDoS Network Model

A Distributed Denial of Service Network follows a hierarchical model, with one or more *attackers* controlling a so-called *handler*, which in turn controls the hordes of *agents* that execute the commands relayed to them.

The communication between the attacker and the handler, and between the handler and the agents is referred to as the *control traffic* of the network, whereas the communication between the agents and the victims is referred to as the *flood traffic*. Control traffic can be TCP, UDP, ICMP, or a combination of the three. Flood traffic consists of traffic generated by each individual point-to-point denial of service technique, or sometimes a combination thereof.

In order to remove himself from view, the attacker introduces additional layers between the victim host(s) and himself. He can access the handler via a variety of mechanisms, the most popular being a simple *telnet*. More sophisticated tools use, or can take advantage of, more advanced techniques, such as encrypted TCP connections (ssh is a possibility for TFN, blowfish-encrypted proprietary as in Stacheldraht) or non-standard methods such as embedding commands in ICMP or UDP packets (e.g., LOKI [22, 23] or Q). Additional care is taken to protect the handler, as it is the key control point and effectively the *anonymizer* of the network. So as to eliminate a single point of failure, more than one handler is found in practice, and in most cases each handler has equal power over its agents.

The agents are controlled from the handler, often using a different protocol than the one in effect between attacker and handler. It is speculated that this is done to evade correlation. The communication is not necessarily bidirectional, as there have been cases of oblivious transfers. Instructed by the attacker, the handler can control the numerous agents to perform the attacks by proxy. As we will see, some DDoS tools provide a clear overview of the DDoS network, e.g., enabling to determine the status and performance of each individual agent.

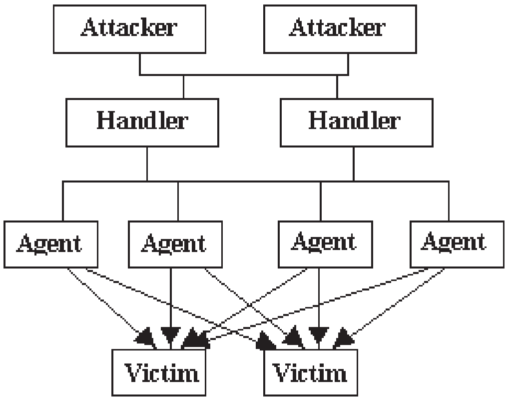

Figure 1: A typical DDoS network

Findings

Shaft was initially detected through anomalous network activity. With the help of the network analyzer Argus [1], spikes (see Figure 2) in the packet flows led to the discovery of the *shaftnode* agent on the compromised system within the local network. It was one of about 100 nodes in a Shaft DDoS network. The successful retrieval of an attack binary, the *shaftnode* agent, and eventually its source, permitted a thorough analysis of its functionality.

Since the *shaftmaster* handler was not retrieved until four months later, it took simulation, thorough analysis of Argus [1] logs and a pinch of creativity to reconstruct the functionality of this attack tool. Simulation and analysis tools such as the Unix debugging tool *strace*, and disassembly of binaries are the main contributors to the understanding of Shaft.

Communication Features

As a first step, it was important to identify the network communication aspect of Shaft. Shaft (in the analyzed version, 1.72) is modeled after Trinoo [11], in that communication between handlers and agents is achieved using the unreliable IP protocol UDP. See Stevens [29] for an extensive discussion of the TCP and UDP protocols. Remote control is established via a simple telnet connection to the handler. Shaft uses *tickets* for keeping track of the transactions issued to its individual agents. Both passwords and ticket numbers have to match for the agent to execute the request. A simple letter-shifting (Caesar cipher, see Schneier [27]) is in use.

Command Structure

Next, analyzing the command structure of the tool provided additional understanding of the capabilities of Shaft. Using available source and the simple Unix command *strings*, we established the command syntax of both the agent and the handler. It provided insight into the capabilities of the handler that was not apparent from the agent source. A full listing of both the agent and handler commands can be found in Appendix 1 and 2, respectively.

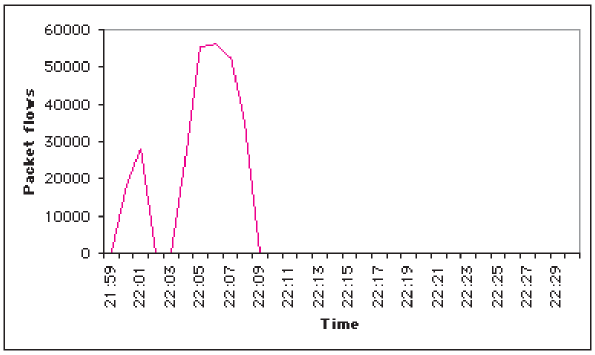

Figure 2: Spikes in network activity.

Detection

Brief Description of Installation Methods

As with previous DDoS tools, 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.). The reader is referred to Dittrich's Trinoo analysis [11] for a description of possible installation methods of this type of tool.

Further findings [32] have revealed that the Shaft DDoS tools were indeed used in conjunction with a *root kit*, an inetd-based (inetd is the Unix server that handles most incoming connections such as telnet and ftp) trojan, a trojaned secure shell (SSH) daemon, and a set of Unix shell scripts to automatically distribute the tools out to the individual agent systems. The present inetd-based trojan has been known to exist in the wild as early as May 1999.

The distribution Unix shell script (from [32]), as sent with netcat [19] to the trojaned system, is as follows:

```
#!/bin/sh
echo "oir##t"
echo "QUIT"
sleep 5
echo "cd /tmp"
```

```
sleep 5
echo "rcp user@host:shaftnode ./"
sleep 5
echo "chmod +x shaftnode"
sleep 5
echo "./shaftnode"
echo "exit"
```

This shell script installs the shaftnode agent on the system, by performing a remote copy from a repository host into the /tmp directory, making it executable and launching it. The reader is referred to [32] for a complete discussion of the installation, trojaning and rootkit-ing of the handler host.

Algorithmic Overview of Attacks

Upon launch, the shaft agent (the "shaftnode") reports back to its default handler (its "shaftmaster") by sending a "new <upshifted password>" command, which registers the new agent in the pool of agents available to the handler. For the default password of "shift" found in the analyzed code, this would be "tijgu". Therefore a new agent would send out "new tijgu", and all subsequent messages would carry that password in it. Only in one case does the agent shift in the opposite direction for one particular command, e.g., "pktres rghes". While it was initially unclear whether this was a mistake, a more thorough analysis of the shaftmaster revealed that both shifts were used in an attempt to evade analysis.

Incoming commands arrive as space separated items: command, upshifted password, command argument, socket number, ticket, and optional arguments, which can be represented as the message flow diagram between handler H and agent A:

1. *A → H: "new", f(password)*
2. *H → A: cmd, f(password), [args], Na, Nb*
3. *A → H: cmdrep, f(password), Na, Nb, [args]*
4. Jump to step 2.

- f(X) is the Caesar cipher function on X
- Na, Nb are numbers (tickets, socket numbers)
- cmd, cmdrep are commands and command acknowledgments
- args are command arguments

The flooding occurs in bursts of 100 packets per host, with the source port and source address randomized. This number is hard-coded, but it is believed that more flexibility can be added. Whereas the source port spoofing only works if the agent is running as a root privileged process, the author has added provisions for packet flooding using the UDP protocol and with the correct source address in the case the process is running as a simple user process. It is noteworthy that the random function is not properly seeded, which may

Time	Protocol	Src IP/Port	Flow	Dst IP/Port
21:39:22	tcp	z.z.z.z.53982	↔	x.x.x.x.21
21:39:32	tcp	x.x.x.x.1023	→	y.y.y.y.514
21:39:56	udp	x.x.x.x.33198	→	z.z.z.z.20433
21:45:20	udp	z.z.z.z.1765	→	x.x.x.x.18753
21:45:20	udp	x.x.x.x.33199	→	z.z.z.z.20433
21:45:59	udp	z.z.z.z.1866	→	x.x.x.x.18753
21:45:59	udp	x.x.x.x.33200	→	z.z.z.z.20433
21:45:59	udp	z.z.z.z.1968	→	x.x.x.x.18753
21:45:59	udp	z.z.z.z.1046	→	x.x.x.x.18753
21:45:59	udp	z.z.z.z.1147	→	x.x.x.x.18753
21:45:59	udp	z.z.z.z.1248	→	x.x.x.x.18753
21:45:59	udp	z.z.z.z.1451	→	x.x.x.x.18753
21:46:00	udp	x.x.x.x.33201	→	z.z.z.z.20433
21:46:00	udp	x.x.x.x.33202	→	z.z.z.z.20433
21:46:01	udp	x.x.x.x.33203	→	z.z.z.z.20433
21:48:37	udp	z.z.z.z.1037	→	x.x.x.x.18753
21:48:37	udp	z.z.z.z.1239	→	x.x.x.x.18753
21:48:37	udp	z.z.z.z.1340	→	x.x.x.x.18753
21:48:37	udp	z.z.z.z.1442	→	x.x.x.x.18753
21:48:38	udp	x.x.x.x.33204	→	z.z.z.z.20433
21:48:38	udp	x.x.x.x.33205	→	z.z.z.z.20433
21:48:38	udp	x.x.x.x.33206	→	z.z.z.z.20433
21:48:56	udp	z.z.z.z.1644	→	x.x.x.x.18753
21:48:56	udp	x.x.x.x.33207	→	z.z.z.z.20433
21:49:59	udp	x.x.x.x.33208	→	z.z.z.z.20433
21:50:00	udp	x.x.x.x.33209	→	z.z.z.z.20433
21:50:14	udp	z.z.z.z.1747	→	x.x.x.x.18753
21:50:14	udp	x.x.x.x.33210	→	z.z.z.z.20433

Table 1: Compromise flow on Nov 28.

lead to predictable source port sequences and source host IP sequences.

The source port is generated with (R *mod* (65535-1024)+1024) where R is the output of the rand() function. This will generate source ports greater than 1024 at all times.

The source IP is of the form R1.R2.R3.R4 where R1, R2, R3, R4 are the outputs of rand() *mod* 255. The source IP numbers can (and will) contain a zero in the leading octet.

Additionally, the sequence number for all TCP packets is fixed, namely 0x28374839, which helps with respect to detection at the network level. The ACK and URGENT flags are randomly set, except on some platforms. Destination ports for TCP and UDP packet floods are randomized.

The client must choose the duration (''time''), size of packets, and type of packet flooding directed at the victim hosts. Each set of hosts has its own duration, which gets divided evenly across all hosts. This is unlike TFN [6] which forks an individual process for each victim host. For the type, the client can select UDP, TCP SYN, ICMP packet flooding, or the combination of all three. Even though there is potential for having a different type and packet size for each set of victim hosts, this feature is not exploited in this version.

When a general command is issued, it is sent to all hosts listed in a hidden file containing all the Shaft agents, in general with a timeout of 30 seconds. To date, no mechanism to alter that timeout has been found. Some commands have longer timeouts, up to 300 seconds. A list of outstanding tickets (transactions waiting to complete) is available to the attacker with the ''ltic'' command, which lists the ticket number and its corresponding remaining time. The attacker can visually correlate the ticket number to the actual command by scrolling back in his screen buffer and comparing the number that was printed after the execution of the command, similar to seeing a process id displayed when sending a process into the background on a Unix system.

The author of Shaft seems to have a particular interest in statistics, namely packet generation rates of its individual agents. The statistics on packet generation rates are possibly used to determine the ''yield'' of the DDoS network as a whole. This would allow the attacker to stop adding hosts to the attack network when it reached the necessary size to overwhelm the victim network, and to know when it is necessary to add more agents to compensate for loss of agents due to attrition during an attack (as the agent systems are identified and taken off-line).

Packet Flow Analysis

In this section we will look at a practical example of an attack carried out with the Shaft distributed denial of service attack tool, as seen from the attacking network perspective.

The handler is listening on port 20433, and an existing connection on port 20432 is awaiting the commands of the attacker. The packet flow is illustrated in Table 1.

There is quite some activity between the handler and the agent, as they go through the command request and acknowledgement phases. There was also what appeared to be testing of the impact on the local network itself with, among others, UDP packet flooding against the broadcast address (first 2-3 spikes), followed by ICMP flooding as shown in Figure 2. See Figure 3 for a fine-grained view.

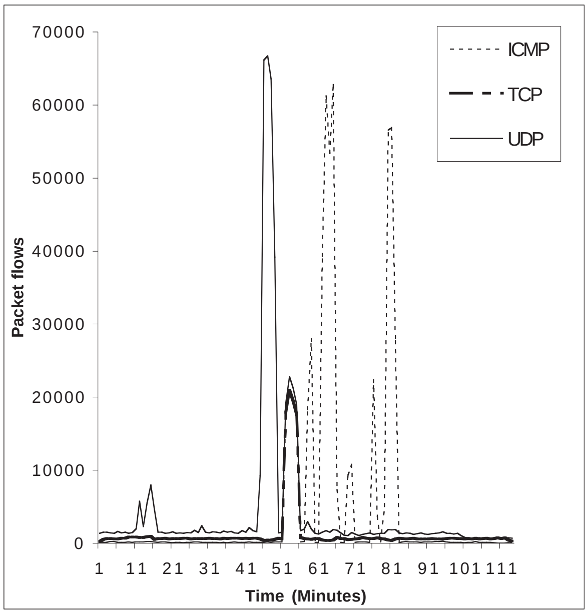

Figure 3: 28 Nov 1999 floods 21:00-23:00.

The interesting portion is the first three lines. It shows the penetration from the handler (z.z.z.z) using the inetd-based trojan with source port 53982 and destination port 21 (any inetd related port would have worked), the download of the shaftnode binary from y.y.y.y via rcp (remote copy, port 514), and the registration of the shaftnode agent with its shaftmaster handler. The theory that these were the traces of the penetration was confirmed by findings [32] on the handler host. The ten second delay between the packet on port 21 and the remote copy on port 514 is consistent with the script mentioned in the section on installation methods. Later that night, the attacker performed several attacks (three UDP and one combination TCP/UDP/ICMP) in order to test the Shaft network further, as illustrated in Figure 4. Let us look at the individual phases from a later attack after it became possible to record the packet contents, as well as general flow data, subsequent to a determination of the agent, handler and communication ports. Subsequently, the handler continued to send such packets even though the agent had been disabled and the host integrity recovered. This is illustrated in Table 2.

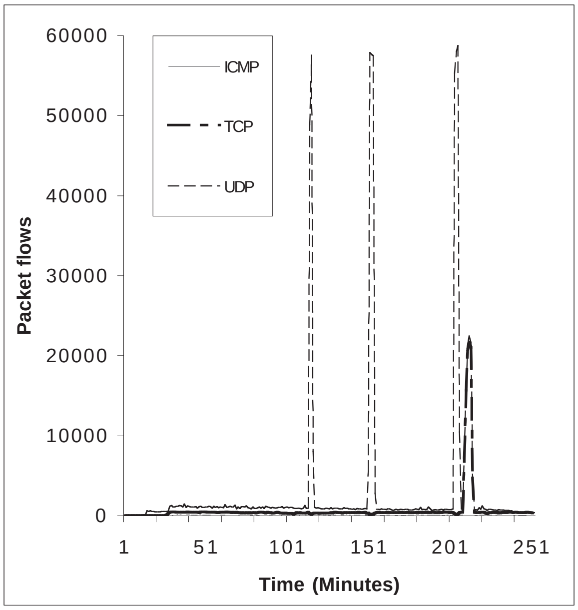

Figure 4: Further testing 29 Nov 1999 02:00-07:00.

time	flow	command
18:06:40	Z → X	alive tijgu hi 5 8170
18:09:14	Z → X	time tijgu 700 5 6437
18:09:14	X → Z	time tijgu 5 6437 700
18:09:16	Z → X	size tijgu 4096 5 8717
18:09:16	X → Z	size tijgu 5 8717 4096
18:09:23	Z → X	type tijgu 2 5 9003

Table 2: Setup and configuration phase on Dec 4.

time	flow	command
18:09:24	Z → X	own tijgu a.a.a.a 5 5256
18:09:24	X → Z	owning tijgu 5 5256 a.a.a.a
18:09:24	Z → X	pktres tijgu a.a.a.a 5 1993
18:09:24	Z → X	own tijgu b.b.b.b 5 78
18:09:24	Z → X	pktres tijgu j.j.j.j 5 8845
18:09:24	Z → X	own tijgu c.c.c.c 5 6247
18:09:25	Z → X	own tijgu d.d.d.d 5 4190
18:09:25	Z → X	own tijgu e.e.e.e 5 2376
18:09:25	X → Z	owning tijgu 5 78 b.b.b.b
18:09:26	X → Z	owning tijgu 5 6247 c.c.c.c
18:09:27	X → Z	owning tijgu 5 4190 d.d.d.d
18:09:28	X → Z	owning tijgu 5 2376 e.e.e.e
18:21:04	X → Z	pktres rghes 5 1993 51600
18:21:04	X → Z	pktres rghes 0 0 51400
18:21:07	X → Z	pktres rghes 0 0 51500
18:21:07	X → Z	pktres rghes 0 0 51400
18:21:07	X → Z	pktres rghes 0 0 51400

Table 3: Host list and statistics.

The handler issues an "alive" command, and says "hi" to its agent, assigning a socket number of "5" and a ticket number of 8170. We will see that this "socket number" will persist throughout this attack. A time period of 700 seconds is assigned to the agent, which is acknowledged. A packet size of 4096 bytes is specified, which is again confirmed. The last line indicates the type of attack, in this case "the works", i.e., UDP, TCP SYN and ICMP packet flooding combined. Failure to specify the type would make the agent default to UDP packet flooding.

Now the list of hosts to attack and which ones they want statistics from on completion, as shown in Table 3. To protect the identity of the victims, the hosts IP number have been replaced with a.a.a.a through j.j.j.j.

time	flow	command
18:24:25	Z → X	own tijgu e.e.e.e 5 4493
18:25:53	Z → X	own tijgu b.b.b.b 5 9392
18:27:05	Z → X	own tijgu a.a.a.a 5 3085
18:27:06	X → Z	owning tijgu 5 3085 a.a.a.a
18:33:52	Z → X	own tijgu c.c.c.c 5 1878
18:33:53	X → Z	owning tijgu 5 1878 c.c.c.c
18:36:04	X → Z	pktres rghes 0 0 104100
18:36:20	Z → X	pktres tijgu a.a.a.a 5 1511
18:36:21	X → Z	owning tijgu 5 1754 a.a.a.a
18:37:33	X → Z	pktres rghes 0 0 81700
18:38:13	Z → X	own tijgu f.f.f.f 5 3126
18:38:13	Z → X	pktres tijgu f.f.f.f 5 4697
18:38:14	X → Z	owning tijgu 5 3126 f.f.f.f
18:38:47	X → Z	pktres rghes 5 1511 76600
18:39:15	Z → X	own tijgu g.g.g.g 5 4272
18:39:16	X → Z	owning tijgu 5 4272 g.g.g.g
18:39:41	Z → X	own tijgu c.c.c.c 5 8850
18:39:41	Z → X	pktres tijgu c.c.c.c 5 9924
18:40:43	Z → X	own tijgu c.c.c.c 5 2672
18:41:25	X → Z	owning tijgu 5 5195 h.h.h.h
18:45:33	X → Z	pktres rghes 5 9924 53700
18:48:01	X → Z	pktres rghes 0 0 48800
18:49:54	X → Z	pktres rghes 5 4697 45700
18:50:56	X → Z	pktres rghes 0 0 44900
18:51:22	X → Z	pktres rghes 0 0 45700
18:53:04	X → Z	pktres rghes 0 0 63700
18:54:47	Z → X	own tijgu i.i.i.i 5 2086
18:54:47	Z → X	pktres tijgu i.i.i.i 5 6980
18:54:47	X → Z	owning tijgu 5 2086 i.i.i.i
19:06:27	X → Z	pktres rghes 5 6980 241200

Table 4: More hosts and statistics.

Now that all other parameters are set, the handler issues several "own" commands, in effect specifying the victim hosts. Those commands are acknowledged by the agent with an "owning" reply. The flooding occurs as soon as the first victim host gets added. The handler also requests packet statistics from the agents for certain victim hosts (e.g., "pktres tijgu a.a.a.a 5 1993"). Note that the reply comes back with the same identifiers ("5 1993") at the end of the 700 second packet flood, indicating that 51600 sets of packets were sent. One should realize that, if successful, this means 51600 x 3 packets due to the configuration of all three (UDP, TCP, and ICMP) types of packets. In turn, this results in roughly 220 4096 byte packets per second per host, or about 900 kilobytes per second per victim host from this agent alone, about 4.5 megabytes per second total for this little exercise. A graphical view can be seen in the first portion (minutes 1 through 12) of Figure 5.

Note the reverse shift (''shift'' becomes ''rghes'', rather than ''tijgu'') for the password on the packet statistics. Continuing on with the attack, as shown in Table 4, the attacker selects new targets in a staggered manner, but still keeping the established settings of the 700 second combination type attack of 4096 byte packets. The yields of the attack vary from roughly 800 kilobytes per second per host in a multi-target setting to 4.2 megabytes per second per host in a single target setting (Target I). The staggered approach can be observed in the right two thirds of Figure 5 (minutes 14 through 50).

Cryptographic Aspects

Shaft incorporated several noteworthy techniques for keeping information secret. For one, the letter-shifting or Caesar cipher, was applied several times within this tool. As described previously, the transaction password ''shift'' was shifted by one letter upwards to generate the string ''tijgu'' observed on the network. However, a different shift in the opposite direction generated ''rghes'' for the return statistics. While the author(s) of this program did not encrypt the entire message exchange between handler and agent, they did nevertheless *obfuscate* the real strings, such as applying the shift to the handler IP numbers in the binary and also the port numbers in the case of a ''switch'' command, namely by adding an offset to the real port number.

As with the original Trinoo tool, the Shaft handler contained 13-character strings, strangely resembling Unix crypt() output. Through close analysis of the handler code, it was established that they represented the access passwords to the control port of the program, that is where the attacker would connect to and perform the distributed denial of service from a convenient, but not quite menu-driven, command line. The actual passwords were recovered in a similar fashion to the ones from Trinoo and Stacheldraht, except that the above shifts had to be performed on the ciphertext first.

Similar to the handler settings in earlier tools, the author of Shaft attempts to keep the list of its agents in a non-trivial format. Other tools encrypt that list using Blowfish, but this tool packs the four octets of the IP number into a 4-byte integer and writes the ASCII representation of that number to a file, one per line. For example, adding the agent with IP address 127.0.0.1 using ''+node 127.0.0.1'' would yield a line containing ''16777343'', which is:

$$127 \times 256^0 + 0 \times 256^1 + 0 \times 256^2 + 1 \times 256^3.$$

In order to extract the IP numbers from the list, one would apply the reverse transformation.

Anomaly Detection

The network flooding which took place was initially noticed after a cursory glance at the hourly network flow data files recorded using an Argus [1] monitor at the main Internet connection point. Without any such monitoring the activities would have almost certainly gone unnoticed since the floods took place over the Sunday-Monday night. Other IDS records

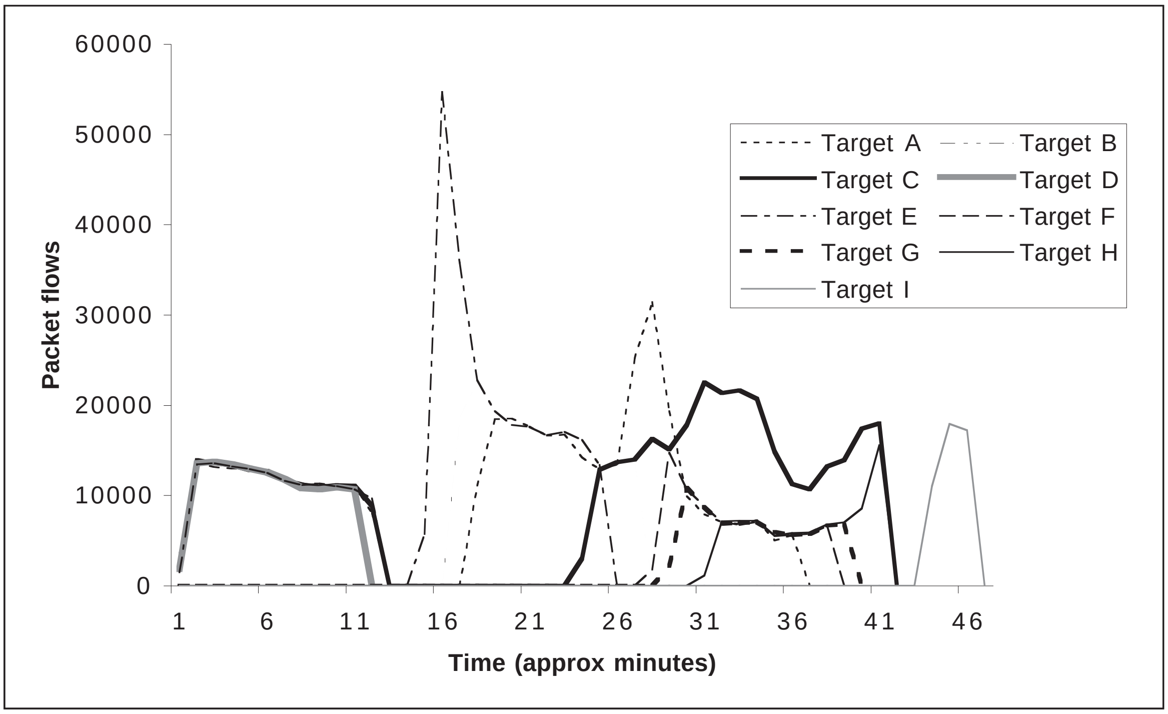

Figure 5: 4 Dec 1999 floods.

indicated a possible UDP portscan had taken place but for host IPs not possible on the local net blocks.

For example, the typical argus data file at that time of day (night) would be 4Mbytes but these grew to between 40 and 100Mbytes. Analyzing such large data files takes significant effort and resources (hence the hourly rotation) but it was possible to determine the start time of the rapid rise in connections and then tracing the external connection (handler) and internal flooder (agent) becomes a matter for trial and error and some measure of good fortune. In this instance the first guess was used to contact the host administrator who was able to locate the process still left running (although inactive), obtain an lsof [18] output and recover residual files and logs.

Reducing the data for the hourly flows down to something suitable for graphical display was complicated by the very large number of data points which overwhelms most of the standard graphing or statistical packages.

Other traffic monitoring applications which might have indicated that an unusually large network flow had occurred would not usually have an accurate time nor could have been used to trace the external → internal communications channel correspondence. For example snmp monitoring of packet numbers is a popular method. Accumulated byte counts per sub-net (host, port, etc) could also have been obtained using NeTraMet [5] but again this would have been inadequate for the post event analysis.

Impact of Victim Hosts/networks

The effect of the combined outpouring of packets has already been considered from the point of view of the target victim but it should be noted that very little legitimate traffic was able to move over the Internet gateway while the flood took place. This secondary denial of service would be of major significance during normal working hours and when combined with several such agents distributed around one site can lead to saturation of the essential backbone infrastructure and routers.

Impact on network – given the time of day it had little or no impact apart from slowing external port scanners (!) – maybe it is worth noting that on typical asynchronous ATM external connections there would be an impact on outgoing verus incoming. The impact of one host running flat out will be a lot less than several hosts running as agents. Deliberately limiting agents to one per site would have considerable benefits when it comes to avoiding detections while still retaining effective DoS of the target(s).

Secondary Effects

Poor DNS response (if any) – even problems managing network devices during the peaks. The flurry of ‘‘response’’ packets caused by the floods can create additional complications within the network. Due to the ‘‘inband-signalling’’ nature of TCP/IP, the control messages related to network management must travel over the very same congested network.

NIDS vs. Active Scanning

Network Intrusion Detection Systems (NIDS)

During very high (near saturation) flows almost no event of any kind would be logged by an IDS system – they would either have to drop packets at a very high rate or require multi-CPU architectures in order to combine packet collection and packet state analysis. As pipe capacities continue to grow (Gigabit, etc) there will be serious difficulties for network flow monitors such as Argus to keep up based on the typical PC architecture (there is seldom a budget available for a top end machine which will, hopefully, be wasting cycles 99 % of the time waiting for such events).

Passive Scanning

For the purpose of detecting malicious activity, certain features of the whole DDoS package have to be considered and provide clues for passive scanning of such events. This program does not provide for code updates (like TFN or Stacheldraht). This may imply ‘‘rcp’’ or ‘‘ftp’’ connections during the initial intrusion phase (see also [11]). As found in [32], the intruders used ‘‘rcp’’ in their distribution scripts, but this could easily be altered.

The program uses UDP traffic for its communication between the handlers and the agents. Considering that the traffic is not encrypted, it can easily be detected based on certain keywords. Performing an ‘‘ngrep’’ [20] for the keywords mentioned in the syntax sections (Appendix 1 and 2), will locate the control traffic, and looking for TCP packets with sequence numbers of 0x28374839 (decimal 674711609) may locate the TCP SYN packet flood traffic. The latter traffic can be detected through its secondary effect of causing SYN|ACK and RST|ACK traffic with sequence numbers of 0x2837483a (decimal 674711610), as pointed out by Richard Bejtlich (who has been witnessing these effects – with this same sequence number – for well over a year [3]). Source ports of the flood traffic are always above 1024, and source IP numbers can include zeroes in the leading octet.

Strings in this control traffic can be detected with the ‘‘ngrep’’ program using the same technique shown in [11, 12, 13]. Here are some examples that will locate the control traffic between the handler and the agent, independently of the port number used.

```
# ngrep -i -x "alive tijgu" udp
U 192.168.10.1:4001 -> 192.168.10.2:18753
61 6c 69 76 65 20 74 69    alive ti
6a 67 75 20 68 69 20 35    jgu hi 5
20 38 36 34 31 0a          8641.

U 192.168.0.2:1494 -> 192.168.0.1:20433
61 6c 69 76 65 20 74 69    alive ti
6a 67 75 20 35 20 38 36    jgu 5 86
34 31 20 62 6c 61 68       41 blah
```

The above will show the "alive" messages exchanged between handler and agents.

```
# ngrep -i -x "pktres|pktstat" udp
U 192.168.10.2:1499 -> 192.168.10.1:20433
70 6b 74 73 74 61 74 20    pktstat
74 69 6a 67 75 20 35 20    tijgu 5
31 32 35 37 20 30          1257 0
```

The above shows the request for packet statistics and the flood results.

```
# ngrep -i -x "switch tijgu" udp
U 192.168.10.1:4001 -> 192.168.10.2:18753
73 77 69 74 63 68 20 74    switch t
69 6a 67 75 20 32 30 34    ijgu 204
38 33 20 35 20 32 39 36    83 5 296

U 192.168.10.2:1522 -> 192.168.10.1:20433
73 77 69 74 63 68 65 64    switched
20 74 69 6a 67 75 20 35     tijgu 5
20 32 39 36                296
```

This previous example shows the directive from the handler to "switch" to this handler.

For specific signature detection, one could also use Snort [25]. See the caveats below.

Active Scanning

Scanning the network for open port 20432 will reveal the presence of a handler on your local area network.

For detecting idle agents, one could write a program similar to George Weaver's trinoo detector. Sending out "alive" messages with the default password to all nodes on a network on the default UDP port 18753 will generate traffic back to the detector, making the agent believe the detector is a handler.

There are also two excellent scanners for detecting DDoS agents on the network: Dittrich's "dds" [15] and Brumley's "rid" [6].

"dds" was written to provide a more portable and less dependent means of scanning for various DDoS tools. (Many people encountered problems with Perl and the Net::RawIP library [24] on their systems, which prevented them from using the scripts provided in [11, 12, 13].) Due to time contraints during coding, "dds" does not have the flexibility necessary to specify arbitrary protocols, ports, and payloads. One would need to modify the source slightly to detect shaft agents or handlers. A modified version of "dds", geared towards detecting only "Shaft" agents, is available [9, 16].

A better means of detecting shaft handlers and agents would be to use a program like "rid", which uses a more flexible configuration file mechanism to define ports, protocols, and payloads.

A sample configuration for "rid" to detect the Shaft control traffic as described:

```
start shaft
    send udp dport=18753
           data="alive tijgu hi 5 1984"
    recv udp sport=20433
           data="alive" nmatch=1
end shaft
```

Caveats

It should be emphasized again and again that the passive and active detection triggers rely on "old" numbers, strings, etc. and that they are often trivial to modify. Selection and use of such tools and commercial NIDS (in particular) should bear this in mind along with their flexibility to insert the "latest" trigger information that may be provided by security teams and organizations.

Related work

We present a brief overview, in chronological order to the best of our knowledge, of DDoS tools that have been mentioned publically.

Early tools

The early tools appeared in early summer 1998. They were clumsy attempts to naturally evolve beyond coordinated attacks [17], but nevertheless laid the foundation to the subsequent tools. The first of them, fapi, featured UDP, TCP (SYN and ACK), and ICMP Echo floods. Its handler to agent communication was UDP-based. It did not provide easy controls for setting up the DDoS network, and did not handle networks over 10 hosts very well. The second one, fuck_them, was a distributed ICMP Echo Reply flooder, where the attacker either supplied the source address to spoof or randomized source addresses were generated (all 32 bits of the IP address).

Trinoo and variants

Trinoo surfaced in the early summer 1999. It has been extensively scrutinized, and we refer to [11] for a thorough analysis. The tool is capable of only generating UDP packet floods without source address forgery, but has full control features. It was capable of crippling the network of the University of Minnesota [10, 11] for three days, leading to a workshop on the subject [7]. Trinoo has mutated at least twice over the last year.

TFN and variants

TFN, a.k.a. Tribe Flood Network, was introduced in late summer 1999. With its limited control features, it still provided UDP packet flood attacks (it gave homage to Trinoo by calling it "trinoo emulation"), TCP SYN flood attacks, ICMP Echo flood attacks, and Smurf attacks in a distributed fashion. It is capable of spoofing either all 32 bits of the IP source address, or just the last 8 bits. As with Trinoo, this tool has been analyzed thoroughly [12].

TFN2K, or TFN2000, is a further development effort on the basis of TFN. It provides the same attacks as TFN, but can randomly do them all at once. Encryption of the control traffic was added to improve the security of the DDoS network and evade signature detection. Control traffic uses a superposition of UDP, TCP and ICMP, using oblivious transfers, i.e., the receipt is not acknowledged. For a brief review, see [2].

Stacheldraht and Variants

Stacheldraht, German for "barbed wire," apparently evolved out of Trinoo and TFN. Analyzed in [13], it has full control features, the same basic attacks and source address forgery as TFN, and as a twist, a Blowfish-encrypted control channel for the attacker. Mutated into StacheldrahtV4 in early 2000, it further mutated into Stacheldraht v1.666, which adds TCP ACK and TCP NUL packet flood attacks, and preconfigured Smurf attacks.

Mstream

As the name suggests, Mstream is a "multiple stream" tool, in reference to the very efficient point-to-point stream TCP ACK flooding tool. It has very limited control features and randomizes all 32 bits of the source IP address. For a review of this tool that appeared in the spring of 2000, please see [14].

Omega

Omega, which appeared in early summer 2000, features TCP ACK packet flooding, UDP packet flooding, ICMP flooding, IGMP packet flooding, and a mix of all four floods. Similar to Shaft, it provides statistics on the floods it produces. It randomizes all 32 bits of the source IP address, and introduces a chat function for communication between attackers.

Trinity and Derivatives

Trinity, and its closely related mutation Entitee, take a new approach on the DDoS model. Rather than relying on a handler network, it takes advantage of an existing Internet Relay Chat (IRC) network for its handler-to-agent communications, making a channel on IRC the "handler." Besides the up to now well-known UDP, TCP SYN, TCP ACK, TCP NUL packet floods, it introduces TCP fragment floods, TCP RST packet floods, TCP random flag packet floods, and TCP established floods, while randomizing all 32 bits of the source IP address.

myServer

In contrast to the sophistication of Trinity, yet released around the same time in summer 2000, myServer is a simplistic DDoS tool. It relies on external programs to provide the denial of service.

Plague

As a third tool in the same generation as Trinity and myServer, it has become obvious Plague was designed by attackers who are reading these reviews and incorporating new improvements based on them. This tool provides TCP ACK and TCP SYN flooding, with what are claimed as fixes over previous TCP ACK flooding tools.

Defenses and Countermeasures

There is no simple solution that would offer one hundred percent protection against these types of tools. There are, however, a number of steps that can be taken to minimize the impact [16]. Several proposed schemes are emerging for adding traceability to TCP/IP packets.

What is necessary to defend? One needs to defend the hosts, the local net, and the backbone infrastructure. As per the recommendations in [7], certain ingress and egress filtering can minimize the impact of denial of service attacks that use spoofed IP source addresses by eliminating illegitimate IP source or destination addresses. In practicality, this will not reduce the impact of DDoS tools that do not spoof their address or only spoof the last 8 bits of the IP address, making it appear to be originating from the local network that the agent resides on.

One also tries to track floods and identify their source in order to shut them down one way or another, or minimize the impact. In general, identifying the source of a spoofed IP address requires the collusion of the intermediate hosts in the path between the actual source and the victim suffering from the denial of service attack. Bellovin's ICMP traceback message scheme [4] addresses that problem by forwarding a signed copy of the transient packet traversing the router in a probabilistic manner. In the proposed version, one in every 20000 packets triggers such behavior, in order to avoid an additional denial of service. Savage et al. take a different approach [26] in inserting partial network path markings into the packet traversing the router in a probabilistic fashion, rather than creating an entirely new packet. Their Fragment Marking Scheme, as pointed out by Song [28], lacks the scalability of dealing with large DDoS networks, causing a large number of false positives. Song's marking schemes provide more efficient traceback under large scale (say 1500 agents) attacks. Stone suggests an IP overlay network to achieve the tracking and forwarding of interesting packets in his CenterTrack [30] scheme.

As mentioned above, anomaly detection can pinpoint the presence of a flood in the first place. Signature detection can either locate known control traffic (either attacker to handler, or handler to agent) or responses from victims.

Future Trends and Evolution

Ever since the introduction of Trinoo, at least eight new tools with varying degrees of sophistication and aimed at creating distributed denial of service have been discovered in a time span less than one calendar year. It is difficult to predict the trends of these tools without ending up being the trendsetter or sparking a new idea for the attacker. These tools have destructive potential and one should remain cautious as to the future directions. It is safe to assume, however, that the trends described in the section on related work will continue, namely in creating distributed variants of existing point-to-point tools.

Conclusion

"Shaft" is another DDoS variant with independent origins. The code recovered did appear to be still in development. Several key features indicate evolutionary trends as the genre develops. Of significance is the priority placed on packet generation statistics which would allow host selection to be refined. The analysis of the code and binary was greatly enhanced by the capture of attack preparation and command packets. The captured packets made it possible to assess the impact of a single agent that managed to saturate the network pipe.

The version analyzed had hooks which would allow for dynamic changes to the master host and control port but not the agent control port. However such items are trivially incorporated and must not be taken to be indicative of any current versions which may be in active use. The obfuscation of master IP, ports and passwords used a relatively simple form of encryption but this could easily be strengthened. Evolutionary findings confirm that information flows back to the authors and cause incorporation of counter-countermeasures as the spiral continues.

The detection of DDoS installations will become very much more difficult as such metamorphosis techniques progress, the presence of such agents will still be more readily determined by analysis of traffic anomalies with a consequent pressure on time and resources for site administrators and security teams.

Acknowledgements

We would like to thank the anonymous referees for their valuable feedback, the CERT Coordination Center for fostering cooperation on DDoS, and also the "last of the quartet," who wishes to remain anonymous, for participating in numerous discussions on DDoS.

Author Information

Sven Dietrich is a senior security architect for Raytheon ITSS at the NASA Goddard Space Flight Center. His focus is computer security, intrusion detection, the building of a PKI for NASA, and the security of IP communications in space. He can be reached at spock@netsec.gsfc.nasa.gov .

Neil Long is a senior systems administrator at the University of Oxford with an academic background in the physical sciences. He has become increasingly interested in network and computer security matters since the early 1990's. He can be reached at neil.long@computing-services.oxford.ac.uk .

Dave Dittrich is a senior security engineer at the University of Washington, supporting Unix workstation administrators on campus for over ten years. His credits include technical analyses (alone or in teams) of the Trinoo, Tribe Flood Network, Stacheldraht, shaft, and mstream distributed denial of service (DDoS) attack tools. He can be reached at dittrich@cac.washington.edu .

References

[1] Argus, ftp://ftp.sei.cmu.edu/pub/argus/ .

[2] Barlow, Jason and Woody Thrower, *TFN2K – An Analysis,* http://www2.axent.com/swat/News/TFN2k_Analysis.htm .

[3] Bejtlich, Richard, Public communication, http://www.sans.org/y2k/032900.htm .

[4] Bellovin, Steve, *ICMP Traceback Messages*, http://www.research.att.com/~smb/papers/draft-bellovin-itrace-00.txt .

[5] Brownlee, Nevil, *NeTraMet, A Network Traffic Accounting Meter*, http://www.auckland.ac.nz/net/NeTraMet/ .

[6] Brumley, David, *Remote Intrusion Detector,* http:// theorygroup.com/Software/RID .

[7] *CERT Distributed System Intruder Tools Workshop report*, http://www.cert.org/reports/dsit_workshop.pdf , Pittsburgh, PA, December 1999.

[8] *CERT Advisory CA-99-17 Denial-of-Service Tools*, http://www.cert.org/advisories/CA-99-17-denial-of-service-tools.html .

[9] Dietrich, Sven, *Distributed Denial of Service Page*, http://netsec.gsfc.nasa.gov/~spock/ddos.html .

[10] Dietrich, Sven, "Scalpel, Gauze, and Decompilers: Dissecting Denial of Service (DDoS)," USENIX *;login:* Magazine, Special Issue on Security, November 2000.

[11] Dittrich, David, *The DoS Project's "trinoo" distributed denial of service attack tool,* http://staff.washington.edu/dittrich/misc/trinoo.analysis .

[12] Dittrich, David, *The "Tribe Flood Network" distributed denial of service attack tool,* http://staff.washington.edu/dittrich/misc/tfn.analysis .

[13] Dittrich, David, *The "Stacheldraht" distributed denial of service attack tool,* http://staff.washington.edu/dittrich/misc/stacheldraht.analysis .

[14] Dittrich, David, George Weaver, Sven Dietrich, and Neil Long, *The "mstream" distributed denial of service attack tool*, http://staff.washington.edu/dittrich/misc/mstream.analysis.txt .

[15] Dittrich, David, Marcus Ranum, George Weaver, David Brumley, et al., http://staff.washington.edu/dittrich/dds .

[16] Dittrich, David, *Distributed Denial of Service (DDoS) Attacks/Tools Page,* http://staff.washington.edu/dittrich/misc/ddos/ .

[17] Green, John, David Marchette, Stephen Northcutt, Bill Ralph, "Analysis Techniques for Detecting Coordinated Attacks and Probes," *Proceedings of USENIX Workshop on Intrusion Detection and Network Monitoring,* Santa Clara, California, April 9-12, 1999.

[18] *lsof*, http://vic.cc.purdue.edu/ .

[19] *netcat,* http://www.l0pht.com/~weld/netcat/ .

[20] *ngrep,* http://www.packetfactory.net/Projects/ngrep/ .

[21] *Packet Storm Security, Distributed denial of service attack tools*, http://packetstorm.securify.com/distributed/ .

[22] *Phrack Magazine*, Volume Seven, Issue Forty-Nine, File 06 of 16, [Project Loki], http://www.phrack.com/search.phtml?view&article=p49-6 .

[23] *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 .

[24] *Net::RawIP:*, http://quake.skif.net/RawIP .

[25] Roesch, Martin, "Snort – Lightweight Intrusion Detection for Networks," *Proceedings of USENIX LISA 1999*, Seattle, Washington, December 1999.

[26] Savage, Stefan, David Wetherall, Anna Karlin, and Tom Anderson, "Practical network support for IP traceback," *Proceedings of the 2000 ACM SIGCOMM Conference*, http://www.cs.washington.edu/homes/savage/papers/Sigcomm00.pdf . August 2000.

[27] Schneier, Bruce, *Applied Cryptography, 2nd edition*, Wiley.

[28] Song, Dawn, and Adrian Perrig, *Advanced and Authenticated Marking Schemes for IP traceback,* Report No. UCB/CSD-00-1107, University of California, Berkeley, June 2000.

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

[30] Stone, Robert, "CenterTrack: An IP-Overlay Network for Tracking DoS floods," *Proceedings to the 9th USENIX Security Symposium*, Denver, Colorado, August 2000.

[31] *tcpdump*, http://www.tcpdump.org/ .

[32] Wash, Richard and Jose Nazario, *Analysis of a Shaft Node and Master*, http://biocserver.cwru.edu/~jose/shaft_analysis/node-analysis.txt .

[33] Zuckerman, M. J., "Net hackers develop destructive new tools", *USA Today,* http://www.usatoday.com/life/cyber/tech/review/crg681.htm , 7 December 1999.

Appendix 1: Agent Commands

Accepted by agent and replies generated back to the handler:

size <*size*> Size of the flood packets. Generates a "size" reply.

type <*0|1|2|3*> Type of DoS to run 0 UDP, 1 TCP, 2 UDP/TCP/ICMP, 3 ICMP. Generates a "type" reply.

time <*length*> Length of DoS in seconds. Generates a "time" reply.

own <*victim*> Add victim to list of hosts to perform denial of service on. Generates a "owning" reply.

end <*victim*> Removes victim from list of hosts (see "own" above). Generates a "done" reply.

stat Requests packet statistics from agent. Generates a "pktstat" reply.

alive Are you alive? Generates a "alive blah" reply.

switch <*handler*> <*port*> Switch the agent to a new handler and handler port. Generates a "switching" reply.

pktres <*host*> Request packet results for that host at the end of the flood. Generates a "pktres" reply.

Sent by agent:

new <*password*> Registering with the handler

pktres <*password*> <*sock*> <*ticket*> <*packets sent*> Packets sents to the host identified by <*ticket*> number.

Appendix 2: Handler commands

This is an overview of the command structure:

mdos <*host list*> Start a distributed denial of service attack (mdos = massive denial of service?) directed at <*host list*>. Sends out "own host" and "pktres" messages to all agents.

edos <*host list*> End the above attack on <*host list*>. Sends out "end host" messages to all agents.

time <*length*> Set the duration of the attack. Sends out "time <*length*>" to all agents.

size <*packetsize*> Set the packetsize for the attack (8K maximum as seen in source). Sends out "size <*packetsize*>" to all agents.

type <*UDP|TCP|ICMP|ALL*> Set the type of attack, UDP packet flooding, TCP SYN packet flooding, ICMP packet flooding, or all three. Sends "type <*type*>" to all agents.

+node <*host list*> Add new agents.

-node <*host list*> Remove agents from pool.

ns <*host list*> Perform a DNS lookup on <*host list*>.

lnod List all agents.

ltic List all pending tickets (transactions).

pkstat Show total packet statistics for agents. Sends out "stat" request to all agents.

alive Send an "alive" to all agents. A possible argument to alive is "hi"

stat show status values (length, type of DoS, packet size).

switch become the handler for agents. Send "switch" to all agents.

ver show version.

whoami returns "God".

exit self-explanatory.