

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

Sven Dietrich
NASA Goddard Space Flight Center
<spock@sled.gsfc.nasa.gov>

Neil Long
Oxford University
<neil.long@computing-services.oxford.ac.uk>

David Dittrich
University of Washington
<dittrich@cac.washington.edu>

Copyright 2000. All rights reserved.
March 13, 2000

-- 1. Introduction

This is an analysis of the "Shaft" distributed denial of service (DDoS) tool. Denial of service is a technique to deny access to a resource by overloading it, such as packet flooding in the network context. Denial of service tools have existed for a while, whereas distributed variants are relatively recent. The distributed nature adds the "many to one" relationship. Throughout this analysis, most actual host names have been modified or removed.

-- 2. Historical overview

"Shaft" belongs in the family of tools discussed earlier, such as Trinoo, TFN, Stacheldraht, and TFN2K. Like in those tools, there are handler (or master) and agent programs. The general concepts of these tools can be found in a Distributed Intruder Tools Workshop Report held in November 1999 at the Computer Emergency Response Team Coordination Center (CERT/CC) in Pittsburgh, Pennsylvania:

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

In chronological order, there are Trinoo, TFN, Stacheldraht, Shaft, and TFN2K. Trinoo, TFN, and Stacheldraht were analyzed in [5], [6], and [7] respectively. TFN2K was recently analyzed in [1].

In the first two months of 2000, DDoS attacks against major Internet sites (such as CNN, ZDNet, Amazon etc.) have brought these tools further into the limelight. There are a few papers covering DDoS to be found at:

<http://packetstorm.securify.com/distributed/>
<http://staff.washington.edu/dittrich/misc/ddos/>
<http://www.cert.org/advisories/CA-99-17-denial-of-service-tools.html>

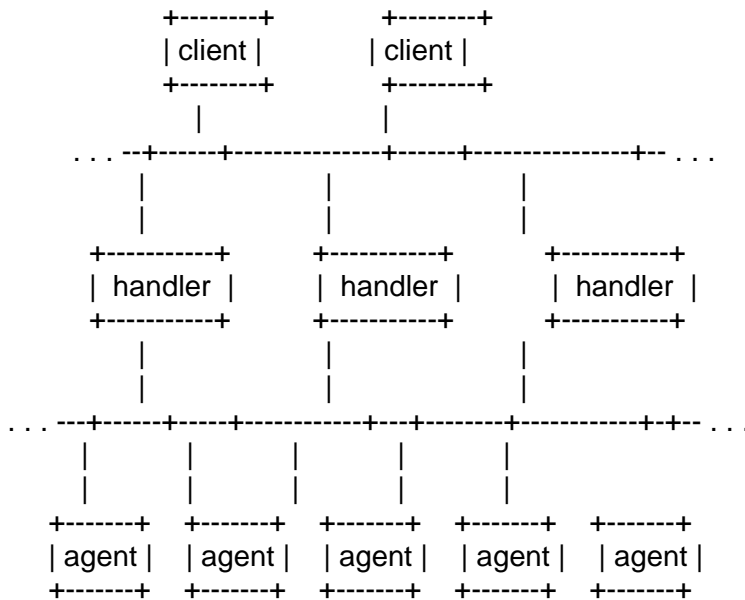
THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

-- 3. Analysis

Shaftnode was recovered, initially in binary form, in late November 1999, then in source form for the agent. Distinctive features are the ability to switch handler servers and handler ports on the fly, making detection by intrusion detection tools difficult from that perspective, a "ticket" mechanism to link transactions, and the particular interest in packet statistics.

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

The "Shaft" network is made up of one or more handler programs ("shaftmaster") and a large set of agents ("shaftnode"). The attacker uses a telnet program ("client") to connect to and communicate with the handlers. A "Shaft" network would look like this:



-- 3.2 Network Communication

Client to handler(s): 20432/tcp
Handler to agent(s): 18753/udp
Agent to handler(s): 20433/udp

"Shaft" (in the analyzed version, 1.72) is modeled after Trinoo, in that communication between handlers and agents is achieved using the unreliable IP protocol UDP. See Stevens [18] for an extensive discussion of the TCP and UDP protocols. Remote control is via a simple telnet connection to the handler. "Shaft" uses "tickets" for keeping track of 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

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

[17]) is in use.

-- 3.3 Commands

The command structure is divided into the agent and handler command syntax groups. The attacker interacts with the handler via a command line.

-- 3.3.1 Agent Command Syntax

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.

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

pktres <host>

Request packet results for that host at the end of the flood

Generates a "pktres" reply.

Sent by agent:

new <password>

Reporting for duty

pktres <password> <sock> <ticket> <packets sent>

Packets sent to the host identified by <ticket> number

-- 3.3.2 Handler (shaftmaster) Command Syntax

Little is known about the handler, but this is a speculation, pieced together from clues, of how its command structure could look like:

mdos <host list>

Start a distributed denial of service attack (mdos = massive denial of service?) directed at <host list>.

Sends out "own host" 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|BOTH>

Set the type of attack, UDP packet flooding, TCP SYN packet flooding, ICMP packet flooding, or all three (here BOTH = ICMP and IP protocols)

Sends "type <type>" to all agents.

+node <host list>

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

Add new agents

-node <host list>
Remove agents from pool

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

Inod
List all agents

ltic
List all 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?

switch
become the handler for agents

Send "switch" to all agents.

ver
show version

exit

-- 3.4 Password protection

After connecting to the handler using the telnet client, the attacker is prompted with "login:". Too little is known about the handler or its encryption method for logging in. A cleartext connection to the handler port is obviously a weakness.

-- 3.5 Detection

-- 3.5.1 Binaries and their behavior

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,

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

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 [5] for a description of possible installation methods of this type of tool.

Precautions have been taken to hide the default handler in the binary code. In the analyzed code, the default handler is defined as follows:

```
#define MASTER      "23:/33/75/28"
```

which would translate into 129.22.64.17 (electrochem1.echem.cwru.edu) using the same simple cipher mentioned above. Port numbers are munged before actual use, e.g.

```
#define MASTER_PORT  20483
```

is really port 20433.

All these techniques intend to hide the critical information from prying eyes performing forensics on the code. The program itself tries to hide itself as a legitimate Unix process (httpd in the default configuration).

Looking at strings in the shaftnode application reveals the following:

```
> strings -n 3 shaftnode
pktres
switch
alive
stat
end
own
time
type
size
httpd
23:/33/75/28
Unable to fork. (do it manually)
shift
new %s
size %s %s %s %s
type %s %s %s %s
time %s %s %s %s
owning %s %s %s %s
switched %s %s %s
done %s %s %s %s
pktstat %s %s %s %lu
alive %s %s %s blah
%d.%d.%d.%d
Error sending tcp packet from %s:%i to %lu:%i
pktres %s %i %i %lu
```

Upon launch, the "Shaft" agent (the "shaftnode") reports back to its

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

default handler (its "shaftmaster") by sending a "new <upshifted password>" command. For the default password of "shift" found in the analyzed code, this would be "tjgu". Therefore a new agent would send out "new tjgu", 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". It is unclear at the moment whether this is intentional or not.

Incoming commands arrive in the format:

```
"command <upshifted password> <command arg> <socket> <ticket> <optional args>"
```

For most commands, the password and socket/ticket need to have the right magic in order to generate a reply and the command to be executed.

Message flow diagram between handler H and agent A:

Initial phase: A -> H: "new", f(password)

Running loop: H -> A: cmd, f(password), [args], Na, Nb

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

- 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.

Source port = (rand() % (65535-1024)+1024) where % is the mathematical 'mod' operator

This will generate source ports greater than 1024 at all times.

Source IP = rand()%255.rand()%255.rand()%255.rand()%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

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

own duration, which gets divided evenly across all hosts. This is unlike TFN [2] 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 of having a different type and packet size for each set of victim hosts, this feature is not exploited in this version.

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.)

Currently, the ability to switch host IP and port for the handler exists, but the listening port for the agent remains the same. It is foreseeable that this will change in the future.

-- 3.5.2 A sample attack

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 shaftnode agent when in use, as seen by "lsof" [10]:

```
# lsof -c shaftnode
COMMAND PID  USER  FD  TYPE   DEVICE  SIZE  NODE NAME
shaftnode 13489  root  cwd  VDIR    0,0    400    2  /tmp
shaftnode 13489  root  txt  VREG    0,0   19492   10  /tmp (swap)
shaftnode 13489  root  txt  VREG   32,0  662764 182321 /usr/lib/libc.so.1
shaftnode 13489  root  txt  VREG   32,0   17480 210757 /usr/platform/sun4u/lib/libc_psr.so.1
shaftnode 13489  root  txt  VREG   32,0  566700 182335 /usr/lib/libnsl.so.1
shaftnode 13489  root  txt  VREG   32,0   39932 182348 /usr/lib/libw.so.1
shaftnode 13489  root  txt  VREG   32,0   15720 182334 /usr/lib/libmp.so.1
shaftnode 13489  root  txt  VREG   32,0   15720 182327 /usr/lib/libintl.so.1
shaftnode 13489  root  txt  VREG   32,0   68780 182342 /usr/lib/libsocket.so.1
shaftnode 13489  root  txt  VREG   32,0   2564 182324 /usr/lib/libdl.so.1
shaftnode 13489  root  txt  VREG   32,0  137160 182315 /usr/lib/ld.so.1
shaftnode 13489  root  0u  inet 0x507dc770 0t116  TCP hostname:ftp-
>electrochem1.echem.cwru.edu:53982 (CLOSE_WAIT)
shaftnode 13489  root  1u  inet 0x507dc770 0t116  TCP hostname:ftp-
>electrochem1.echem.cwru.edu:53982 (CLOSE_WAIT)
shaftnode 13489  root  2u  inet 0x507dc770 0t116  TCP hostname:ftp-
>electrochem1.echem.cwru.edu:53982 (CLOSE_WAIT)
shaftnode 13489  root  3u  inet 0x5032c7d8 0t0    UDP *:18753 (Idle)
```

As one can see, the agent is waiting to receive commands on its default UDP port number 18753. The TCP connection back to the handler remains unexplained to date.

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

Packet flows:

Date	Time	Protocol	Source IP/Port	Flow	Destination IP/Port
Sun 11/28	21:39:22	tcp	129.22.64.17.53982	<->	x.x.x.x.21
Sun 11/28	21:39:56	udp	x.x.x.x.33198	->	129.22.64.17.20433
Sun 11/28	21:45:20	udp	129.22.64.17.1765	->	x.x.x.x.18753
Sun 11/28	21:45:20	udp	x.x.x.x.33199	->	129.22.64.17.20433
Sun 11/28	21:45:59	udp	129.22.64.17.1866	->	x.x.x.x.18753
Sun 11/28	21:45:59	udp	x.x.x.x.33200	->	129.22.64.17.20433
Sun 11/28	21:45:59	udp	129.22.64.17.1968	->	x.x.x.x.18753
Sun 11/28	21:45:59	udp	129.22.64.17.1046	->	x.x.x.x.18753
Sun 11/28	21:45:59	udp	129.22.64.17.1147	->	x.x.x.x.18753
Sun 11/28	21:45:59	udp	129.22.64.17.1248	->	x.x.x.x.18753
Sun 11/28	21:45:59	udp	129.22.64.17.1451	->	x.x.x.x.18753
Sun 11/28	21:46:00	udp	x.x.x.x.33201	->	129.22.64.17.20433
Sun 11/28	21:46:00	udp	x.x.x.x.33202	->	129.22.64.17.20433
Sun 11/28	21:46:01	udp	x.x.x.x.33203	->	129.22.64.17.20433
Sun 11/28	21:48:37	udp	129.22.64.17.1037	->	x.x.x.x.18753
Sun 11/28	21:48:37	udp	129.22.64.17.1239	->	x.x.x.x.18753
Sun 11/28	21:48:37	udp	129.22.64.17.1340	->	x.x.x.x.18753
Sun 11/28	21:48:37	udp	129.22.64.17.1442	->	x.x.x.x.18753
Sun 11/28	21:48:38	udp	x.x.x.x.33204	->	129.22.64.17.20433
Sun 11/28	21:48:38	udp	x.x.x.x.33205	->	129.22.64.17.20433
Sun 11/28	21:48:38	udp	x.x.x.x.33206	->	129.22.64.17.20433
Sun 11/28	21:48:56	udp	129.22.64.17.1644	->	x.x.x.x.18753
Sun 11/28	21:48:56	udp	x.x.x.x.33207	->	129.22.64.17.20433
Sun 11/28	21:49:59	udp	x.x.x.x.33208	->	129.22.64.17.20433
Sun 11/28	21:50:00	udp	x.x.x.x.33209	->	129.22.64.17.20433
Sun 11/28	21:50:14	udp	129.22.64.17.1747	->	x.x.x.x.18753
Sun 11/28	21:50:14	udp	x.x.x.x.33210	->	129.22.64.17.20433

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 ICMP packet flooding, for which we omit the data here due to size limitations.

Let us look at the individual phases from a later attack.

Setup and configuration phase:

date	time	src	dest	dest-port	command
4 Dec 1999	18:06:40	129.22.64.17	x.x.x.x	18753	alive tijgu hi 5 8170
4 Dec 1999	18:09:14	129.22.64.17	x.x.x.x	18753	time tijgu 700 5 6437
4 Dec 1999	18:09:14	x.x.x.x	129.22.64.17	20433	time tijgu 5 6437 700
4 Dec 1999	18:09:16	129.22.64.17	x.x.x.x	18753	size tijgu 4096 5 8717
4 Dec 1999	18:09:16	x.x.x.x	129.22.64.17	20433	size tijgu 5 8717 4096
4 Dec 1999	18:09:23	129.22.64.17	x.x.x.x	18753	type tijgu 2 5 9003

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

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

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:

date	time	src	dest	dest-port	command
4 Dec 1999	18:09:24	129.22.64.17	x.x.x.x	18753	own tijgu 207.229.143.6 5 5256
4 Dec 1999	18:09:24	x.x.x.x	129.22.64.17	20433	owning tijgu 5 5256 207.229.143.6
4 Dec 1999	18:09:24	129.22.64.17	x.x.x.x	18753	pktres tijgu 207.229.143.6 5 1993
4 Dec 1999	18:09:24	129.22.64.17	x.x.x.x	18753	own tijgu 24.7.231.128 5 78
4 Dec 1999	18:09:24	129.22.64.17	x.x.x.x	18753	pktres tijgu 24.218.58.101 5 8845
4 Dec 1999	18:09:24	129.22.64.17	x.x.x.x	18753	own tijgu 18.85.13.107 5 6247
4 Dec 1999	18:09:25	129.22.64.17	x.x.x.x	18753	own tijgu 24.218.52.44 5 4190
4 Dec 1999	18:09:25	129.22.64.17	x.x.x.x	18753	own tijgu 207.175.72.15 5 2376
4 Dec 1999	18:09:25	x.x.x.x	129.22.64.17	20433	owning tijgu 5 78 24.7.231.128
4 Dec 1999	18:09:26	x.x.x.x	129.22.64.17	20433	owning tijgu 5 6247 18.85.13.107
4 Dec 1999	18:09:27	x.x.x.x	129.22.64.17	20433	owning tijgu 5 4190 24.218.52.44
4 Dec 1999	18:09:28	x.x.x.x	129.22.64.17	20433	owning tijgu 5 2376 207.175.72.15
4 Dec 1999	18:21:04	x.x.x.x	129.22.64.17	20433	pktres rghes 5 1993 51600
4 Dec 1999	18:21:04	x.x.x.x	129.22.64.17	20433	pktres rghes 0 0 51400
4 Dec 1999	18:21:07	x.x.x.x	129.22.64.17	20433	pktres rghes 0 0 51500
4 Dec 1999	18:21:07	x.x.x.x	129.22.64.17	20433	pktres rghes 0 0 51400
4 Dec 1999	18:21:07	x.x.x.x	129.22.64.17	20433	pktres rghes 0 0 51400

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 207.229.143.6 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.

Note the reverse shift ("shift" becomes "rghes", rather than "tijgu") for the password on the packet statistics.

-- 3.5.3 Detection at the network level

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

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

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.

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 [5]).

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" [11] for the keywords mentioned in the syntax sections (3.3.1 and 3.3.2), will locate the control traffic, and looking for TCP packets with sequence numbers of 0x28374839 may locate the TCP SYN packet flood traffic. Source ports 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 [5], [6], and [7]. For example,

```
# ngrep -i -x "alive tijgu" udp
```

```
# ngrep -i -x "pktres|pktstat" udp
```

will locate the control traffic between the handler and the agent, independently of the port number used.

There are also two excellent scanners for detecting DDoS agents on the network: Dittrich's "dds" [8] and Brumley's "rid" [2].

"dds" was written to provide a more portable and less dependant means of scanning for various DDoS tools. (Many people encountered problems with Perl and the Net::RawIP library [15] on their systems, which prevented them from using the scripts provided in [5], [6], and [7].) Due to time constraints during coding, "dds" does not have the flexibility necessary to specify arbitrary protocols, ports, and payloads. A modified version of "dds", geared towards detecting only "Shaft" agents, is included in the Appendix.

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 1918"
  recv udp sport=20433 data="alive" nmatch=1
end shaft
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

-- 3.6 Defenses

To protect against the effects of the multiple types of denial of service, we suggest that you review the other papers (see [1, 3, 5, 6, 7]) and other methods of dealing with DDoS attacks being discussed and promoted (see [9]).

For example, rate-limiting is considered effective against ICMP packet flooding attacks, while anti-spoof filters and egress filters at the border routers can limit the problems caused by attacking agents faking source addresses.

-- 4. Further evolution

While the author(s) of this tool did not pursue the use of encryption of its control traffic, such an evolution is conceivable, since a Caesar cipher is used to obfuscate the password. A transition to Blowfish or other stream ciphers is realistic, and changing the communication protocol to ICMP, much like TFN, is conceivable. The use of multicast protocols for both communication or packet flooding is also possible.

To date, no source for the "Shaft" handler ("shaftmaster") has been obtained or analyzed.

At this stage, the code is believed to be private. This would mean that the authors could likely change defaults and the probability of detecting "script kiddie" copycats using default values as analyzed here is low. This would argue for rapid and widespread detection efforts to identify agents before this change.

-- 5. 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.

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

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.

-- APPENDIX A: References

-
- [1] Barlow, Jason and Woody Thrower. TFN2K - An Analysis
http://www2.axent.com/swat/News/TFN2k_Analysis.htm
 - [2] Brumley, David. Remote Intrusion Detector.
<http://theorygroup.com/Software/RID>
 - [3] CERT Distributed System Intruder Tools Workshop report
http://www.cert.org/reports/dsit_workshop.pdf
 - [4] CERT Advisory CA-99-17 Denial-of-Service Tools
<http://www.cert.org/advisories/CA-99-17-denial-of-service-tools.html>
 - [5] Dittrich, David. The DoS Project's "trinoo" distributed denial of service attack tool
<http://staff.washington.edu/dittrich/misc/trinoo.analysis>
 - [6] Dittrich, David. The "Tribe Flood Network" distributed denial of service attack tool
<http://staff.washington.edu/dittrich/misc/tfn.analysis>
 - [7] Dittrich, David. The "Stacheldraht" distributed denial of service attack tool
<http://staff.washington.edu/dittrich/misc/stacheldraht.analysis>
 - [8] Dittrich, David, Marcus Ranum, George Weaver, David Brumley et al.
<http://staff.washington.edu/dittrich/dds>
 - [9] Dittrich, David, Distributed Denial of Service (DDoS) Attacks/Tools
<http://staff.washington.edu/dittrich/misc/ddos/>
 - [10] Isof:
<http://vic.cc.purdue.edu/>
 - [11] ngrep:
<http://www.packetfactory.net/ngrep/>
 - [12] Packet Storm Security, Distributed denial of service attack tools
<http://packetstorm.securify.com/distributed/>
 - [13] Phrack Magazine, Volume Seven, Issue Forty-Nine,
File 06 of 16, [Project Loki]
<http://www.phrack.com/search.phtml?view&article=p49-6>
 - [14] Phrack Magazine Volume 7, Issue 51 September 01, 1997,
article 06 of 17 [L O K I 2 (the implementation)]
<http://www.phrack.com/search.phtml?view&article=p51-6>

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

[15] Net::RawIP:
<http://quake.skif.net/RawIP>

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

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

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

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

-- APPENDIX B: dds ("Shaft" only variant)

/*
* dds \$Revision: 1.6s \$ - a distributed DoS tool scanner - Shaft only
*

* Based on the gag scanner, written by David Dittrich, University
* of Washington, Marcus Ranum, Network Flight Recorder, with
* code contributed by others, and based on an idea stolen from
* George Weaver, Pennsylvania State University.
*

* Dave Dittrich <dittrich@cac.washington.edu>
* Marcus Ranum <mjr@nfr.net>
* George Weaver <gmw@psu.edu>
* David Brumley <dbrumley@rtfm.stanford.edu>
*/

/* Shaft only version, modified to that effect by
* Sven Dietrich <spock@sled.gsfc.nasa.gov>
*/

#if YOU_HAVE_NOT_READ_THIS_YET

This software should only be used in compliance with all applicable laws and the policies and preferences of the owners of any networks, systems, or hosts scanned with the software

The developers and licensors of the software provide the software on an "as is" basis, excluding all express or implied warranties, and will not be liable for any damages arising out of or relating to use of the software.

THIS SOFTWARE IS MADE AVAILABLE "AS IS", AND THE UNIVERSITY OF WASHINGTON DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, WITH REGARD TO THIS SOFTWARE, INCLUDING WITHOUT LIMITATION ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND IN NO EVENT SHALL THE UNIVERSITY OF WASHINGTON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, TORT (INCLUDING NEGLIGENCE) OR STRICT LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

```
#endif
```

```
#define VERSION "$Revision: 1.6s $"
```

```
#include <stdlib.h>
#include <ctype.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/in_sysm.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/ip_icmp.h>
```

```
#define BS 1024
#define __FAVOR_BSD
```

```
/* The two arrays below are for address range calculations. They
   should have been automatically generated, but
   1) I am lazy.
   2) There are a few special cases in them.
```

I will not scan more than a /16. When we do scan a CIDR block, we assume that it actually is a CIDR block, and do not scan the network or broadcast address.

```
*/
```

```
static unsigned long MaskBits[] = {
    0x00000000,      /* /0 */
    0x00000000,      /* /1 */
    0x00000000,      /* /2 */
    0x00000000,      /* /3 */
    0x00000000,      /* /4 */
    0x00000000,      /* /5 */
    0x00000000,      /* /6 */
    0x00000000,      /* /7 */
    0x00000000,      /* /8 */
    0x00000000,      /* /9 */
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
0x00000000,      /* /10 */
0x00000000,      /* /11 */
0x00000000,      /* /12 */
0x00000000,      /* /13 */
0x00000000,      /* /14 */
0x00000000,      /* /15 */
0xffff0000,      /* /16, Class B */
0xffff8000,      /* /17, 128 * Class C */
0xffffc000,      /* /18, 64 * Class C */
0xffffe000,      /* /19, 32 * Class C */
0xfffff000,      /* /20, 16 * Class C */
0xfffff800,      /* /21, 8 * Class C */
0xfffffc00,      /* /22, 4 * Class C */
0xfffffe00,      /* /23, 2* Class C */
0xffffff00,      /* /24, Class C */
0xffffff80,      /* /25, 128 hosts */
0xffffffc0,      /* /26, 64 hosts */
0xffffffe0,      /* /27, 32 hosts */
0xfffffff0,      /* /28, 16 hosts */
0xfffffff8,      /* /29, 8 hosts */
0xffffffc,       /* /30, 4 hosts (PPP link) */
0xffffffe,       /* /31, invalid */
0xfffffff,       /* /32, host */
};

static int NumHosts[] = {
    0, 0, 0, 0,
    0, 0, 0, 0,
    0, 0, 0, 0,
    0, 0, 0, 0,      /* don't scan more than a /16 */
    65534,           /* These are all -2 so that we don't
                       scan the broadcast addr or the
                       network addr */

    32766,
    16382,
    8190,
    4094,
    2046,
    1022,
    510,
    254,
    126,
    62,
    30,
    14,
    6,
    2,
    0,
    1,
};

extern char      *optarg;
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
struct udppkt_t {
    struct ip    ipi;
    struct udphdr udpi;
    char        buffer[BS];
} udppkt;

static void      listener();
static int       usage();

static int       vflg = 0; /* verbosity */
static int       dflg = 0; /* debugging */

/* shaft variables */
static short     shaft_dstport = 18753; /* handler listen port */
static short     shaft_rctport = 20433; /* agent listen port */
char             shaft_scmd[] = "alive";
char             shaft_spass[] = "tijgu";
char             shaft_echostr[] = "alive";

int
main(int argc, char **argv)
{
    int          pid, host;
    char         target[128];
    unsigned long target_host;
    struct in_addr target_ip;
    int          mask;
    char *       mask_ptr;
    int          result;
    int          usock;
    char         buf[BS];
    struct sockaddr_in
        usa;
    int          i;
    char         *jnk1;
    char         *jnk2;
    int          sleepytime = 500;
    int          bigsleep = 30;
    int          num_hosts;
    char         scmd[BS], spass[BS], sbuf[BS];

    while((i = getopt(argc,argv,"ds:S:v")) != -1) {
        switch(i) {
            case 'd':
                dflg++;
                break;
            case 's':
                sleepytime = atoi(optarg);
                if(sleepytime <= 0) {
                    fprintf(stderr,"WARNING: zero interping sleep time will probably overflow your sy
stem's transmit buffers and yield poor results\n");
                    sleepytime = 1;
                }
        }
    }
}
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
        break;
    case 'S':
        bigsleep = atoi(optarg);
        if(bigsleep <= 0) {
            fprintf(stderr, "WARNING: negative sleep value - staying with default of %d\n", bi
gsleep);
        }
        break;
    case 'v':
        vflg++;
        break;
    default:
        exit(usage());
    }
}

if(optind >= argc || argc - optind > 1)
    exit(usage());

mask_ptr = strchr(argv[optind], '/');

/* if a CIDR block is passed in */
if (mask_ptr) {
    *mask_ptr = '\0';
    mask_ptr ++;

    sscanf(mask_ptr, "%d", &mask);

} else {
    printf("No mask passed, assuming host scan (/32)\n");
    mask = 32;
}

result = inet_aton(argv[optind], &target_ip);

if (result == 0) {
    fprintf(stderr, "%s: Bad IP address: %s\n", argv[0],
        argv[optind]);
    exit(-1);
}

if (mask < 16) {
    fprintf(stderr, "Bad Network Admin! Bad! Do not scan more than a /16 at once!\n");
    exit(-1);
}

num_hosts = NumHosts[mask];

if (num_hosts == 0) {
    fprintf(stderr, "Cannot scan a /%d. Exiting...\n", mask);
    exit(-1);
}
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
if(vflg) {
    printf("Mask: %d\n", mask);
    printf("Target: %s\n", inet_ntoa(target_ip));
    printf("dds %s - scanning...\n\n", VERSION);
}

sprintf(sbuf,"%s %s hi 5 1918",shaft_scmd,shaft_spass);

target_host = ntohl(target_ip.s_addr);
target_host &= MaskBits[mask];

target_ip.s_addr = htonl(target_host);

if((pid = fork()) < 0) {
    perror("cannot fork");
    exit(1);
}

/* child side listens for return packets */
if (pid == 0)
    listener();

sleep(1);

/* main sweep loop - COULD be expanded to whole Internet but... */
/* but that would be _very_ bad.... */
while (num_hosts) {
    if (mask != 32) {
        target_host++;
    }
    target_ip.s_addr = htonl(target_host);

    num_hosts--;

    /* we really need to skip the network and broadcast addresses */
    if ((target_host & 0xff) == 0 || (target_host & 0xff) == 0xff) {
        if(vflg)
            printf("Skipping special address %s\n", inet_ntoa(target_ip));
        continue;
    }

    if(vflg)
        printf("Probing address %s\n", inet_ntoa(target_ip));

    /* shaft check */
    bzero((char *) &usa, sizeof(usa));
    usa.sin_family = AF_INET;
    usa.sin_addr.s_addr = target_ip.s_addr;
    usa.sin_port = htons(shaft_dstport);

    if (dflg)
        fprintf(stderr,"Sending UDP to: %s\n",
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
        inet_ntoa(usa.sin_addr));
    if ((usock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("cannot open UDP socket");
        exit(1);
    }

    i = sendto(usock,sbuf,strlen(sbuf), 0,
        (struct sockaddr *)&usa,
        sizeof(usa));

    if (i < 0) {
        char ebuf[BS];
        sprintf(ebuf,"sendto: udp %s",
            inet_ntoa(usa.sin_addr));
        perror(ebuf);
        break;
    }
    close(usock);

    usleep(sleeptime);
}

/* wait for any late responses */
if (dflg)
    fprintf(stderr,"Waiting %d seconds for late responses.\n",
        bigsleep);
sleep(bigsleep);

/* shut listener. if this fails the listener exits on its own */
(void)kill(pid, SIGHUP);
exit(0);
}

static void listener()
{
    int      usock;
    int      i, len;
    fd_set   fdset;
    char     buf[BS];
    char     rcmd[BS], filler[BS], rpass[BS];
    struct timeval  timi;
    struct udppkt_t
        upacket;
    struct sockaddr_in
        sa, from;

    /* child becomes a listener process */

    if ((usock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
        perror("cannot open raw UDP listen socket");
        exit(1);
    }
}
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
}

bzero((char *) &sa, sizeof(sa));
sa.sin_family = AF_INET;
sa.sin_addr.s_addr = INADDR_ANY;
sa.sin_port = htons(shaft_rctport);

if (bind(usock, (struct sockaddr *)&sa, sizeof(sa)) < 0) {
    perror("cannot bind to socket");
    exit(-1);
}

while (1) {
    /* if parent has exited, die */
    if(getppid() == 1)
        exit(0);

    FD_ZERO(&fdset);
    FD_SET(usock, &fdset);
    timi.tv_sec = 1;
    timi.tv_usec = 0;
    select(FD_SETSIZE, &fdset, NULL, NULL, &timi);
    usleep(100);
    if (FD_ISSET (usock, &fdset)) {
        /* read data from UDP listen socket */
        memset((void *) &upacket, 0, sizeof(struct udppkt_t));
        len = sizeof(from);
#if 1
        if ((i = recvfrom(usock, buf, BS, 0,
            (struct sockaddr *) &from, &len)) < 0) {
            perror("recvfrom");
            continue;
        }
#else
        i = read (usock, (char *) buf, BS) -
            (sizeof (struct ip) + sizeof (struct udphdr));
#endif
        sa.sin_addr.s_addr = upacket.ipi.ip_src.s_addr;
        if(dflg)
            fprintf(stderr,
                "Listener got a UDP packet on port %s\n",
                shaft_rctport);

        /* shaft check */
        if (strstr(buf,shaft_echostr)) {
            printf("Received '%s' from %s",
                shaft_echostr,
                inet_ntoa(from.sin_addr));
            printf(" - probable shaft agent\n");
        }
        else {
            printf("Unexpected UDP packet received on port %d from %s\n",
                shaft_rctport, inet_ntoa(from.sin_addr));
        }
    }
}
```

THE "SHAFT" DISTRIBUTED DENIAL OF SERVICE ATTACK TOOL

```
    }  
  }  
}  
  
static int  
usage()  
{  
    fprintf(stderr,"usage: dds [options] <target>\n");  
    fprintf(stderr,"target is CIDR block to scan in form:\n");  
    fprintf(stderr,"\tA.B.C.D/mask\n");  
    fprintf(stderr,"Options:\n");  
    fprintf(stderr,"\t[-v] turns on verbosity\n");  
    fprintf(stderr,"\t[-d] turns on debugging\n");  
    fprintf(stderr,"\t[-s] interpacket sleep in microseconds\n");  
    fprintf(stderr,"\t[-S] delay for late packets\n");  
  
    return(1);  
}
```

Dr. Sven Dietrich Raytheon ITSS | spock@sled.gsfc.nasa.gov
ESDIS Project, Code 586, Bldg 32 Rm N231 | +1-301-614-5119 | 614-5270 Fax
NASA Goddard Space Flight Center | Greenbelt, MD 20771, USA