Global Information Assurance Certification Paper

Name: Kam Ng
Date: 27 March 2001

Exploit Name: Naptha
Type: Remote DoS/RS
Operating System: All Unix and Windows system favors
Protocols/Service: TCP/IP

Naptha, a remote DoS/RS (Denial of Service/ Resources Starvation) attack, exploits the features inherent to the TCP protocol. It runs very efficient at the attacking system, having a potential to cause significant DoS to the victim. Its efficiency is gained by using raw packet handling technique rather relying on the system's native TCP/IP stack. Its program components make it very easy to put together different attack receipt to fit different situations.

Transmission Control Protocol

TCP is the most widely used protocol. It is sitting at the fourth layer of the OSI (Open System Interconnection) reference model, right above the IP (Internet Protocol) layer. The most common term used is TCP/IP. But they are independent layers in the OSI model. As such, UDP/IP is equally valid but is less mentioned.

Relying on the IP layer and below, TCP was designed to provide reliable end-to-end data byte stream over an unreliable Internet network. For the most part, Internet is a heterogeneous network structure comprising of different topologies, technologies, bandwidth, delay, packet size and other network parameters. In view of this, TCP was designed with network adaptability features that applications could pick and choose to dynamically optimize the transfer character of the data and to minimize the impact of different kinds of potential failure in the Internet.

Most systems today run TCP transport entity. The actual implementation can vary from a piece of program or an integrated part of the system kernel. From OSI model perspective, TCP interfaces with IP underneath and Session layer above. When an application in the system wants to transmit a load of data to the remote machine through network, it sends data through the different OSI model layer. TCP is just one of the layer it will go through. Different layer plays a different role and service.

In general, network protocol dictates how and what data will be sent between the communicating networked parties. The number of party is not limited only to two. Ethernet protocol is popular example that allows a larger number of

communicating parties to share a common media. The “how” specifies how all the necessary communicating activities should be conducted. And the “what” specifies the communication data format.

TCP service, running at the local system, formats data to transmit to the remote system using the pre-defined TCP Segment Header. The header is shown as below.

16 bits							16 bits
Source Port							Destination port
Sequence number							
Acknowledgement number							
TCP Header Length	U R G	A C K	P S H	R S T	S Y N	F I N	Window size
Checksum							Urgent pointer
Options (0 or more 32-bits words)							
Data (optional)							

URG: Urgent Data Flag
ACK: Ack flag
PSH: Pushed data without buffering flag
RST: Reset connection flag
SYN: Syn flag
FIN: Finish flag

TCP payload is in the “Data” field and the header above it is for flow control purposes. The operation detail is not the intent of this paper. Please refer to reference #1 listed in the last section of this document.

Before transmitting data payload across the network, TCP services running at systems on both end of a network connection, the sender and receiver, will have to go through a three ways hand-shaking to establish end points, called sockets. Each socket is identified by the 32-bit IP address of the hosts and a 16-bit number local to the host, called a port. Port number less than 1024 are reserved and called “well-known” ports. For example, port 23 and 25 are used for telnet and SMTP respectively.

TCP is a byte stream connection, not a message stream. The boundaries of the message are not preserved end to end. The TCP data payload can be split or combined along the network path by the intermediary network systems due to various practical reasons. For example, if the sending process sends four 512 bytes to the TCP data stream, they can arrive the receiving process at the other end of the network as four 512 bytes, two 1024 bytes or one 2048 bytes data

chunk.

As an intermediary delivery service, when an application passes data to TCP, it may choose to send the data immediately or buffer the data before sending. This may be necessary when it's more efficient to send data chunk larger than the size of data that the application passes to it. But there is provision that the application can force TCP to send data immediately using the PUSH flag.

TCP Connection Management

Connection Establishment Phase

TCP uses a three-way hand-shaking mechanism to establish a connection as shown in the diagram below.

```
Step #1   [Host1]  --------Syn (seq# = x) --------------------------------------------------> [Host2]
Step #2   [Host1] < ------ if accepted: Syn/Ack (seq# = y; ack seq# = x+1) --------  [Host2]
                   < ------ if no accepted: RST----------------------------------------------
Step #3   [Host1]  -------- Ack ( seq# = X+1; Ack seq# = y+1) ---------------------- > [Host2]
```

Below is a snort output of a FTP service connection establishment between clients to server.

Step#1: The client send a TCP Syn to port 21(ftp) port of server with seq# 346875 hex, ack seq# 0 hex

```
03/11-09:10:20.691235 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:47617 IpLen:20 DgmLen:44 DF
******S* Seq: 0x346975  Ack: 0x0  Win: 0x2000  TcpLen: 24
TCP Options (1) => MSS: 1460
```

Step#2: The server accepted, sending back TCP Syc/Ack back to the client with seq# 137F4 hex and ack seq# (346875 + 1) to indicate accepting the request.

```
03/11-09:10:20.691490 192.168.0.1:21 -> 192.168.0.2:1056
TCP TTL:128 TOS:0x0 ID:26112 IpLen:20 DgmLen:44 DF
***A**S* Seq: 0x137F4  Ack: 0x346976  Win: 0x2238  TcpLen: 24
TCP Options (1) => MSS: 1460

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Step#3: The client signaled back TCP Ack with same seq# but with ack seq# increased by one. At this point the TCP connection was established.

```
03/11-09:10:20.701260 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:47873 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x346976  Ack: 0x137F5  Win: 0x2238  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

TCP Data Exchange Phase

Once the connection establishment is completed, data exchange can be started. Below are snort output of the user id and password exchange of the ftp connection between the client and server.

Packet#1

The ftp service running on the server prompted the user at the client side asking for user id. See that the "PUSH" flag was set, indicating that the server is requesting the TCP service running at the client side not to buffer the data but passes it to the ftp client application right away upon arrival. This is needed to deliver quick response to the user.

```
03/11-09:10:20.721275 192.168.0.1:21 -> 192.168.0.2:1056
TCP TTL:128 TOS:0x0 ID:26368 IpLen:20 DgmLen:128 DF
***AP*** Seq: 0x137F5  Ack: 0x346976  Win: 0x2238  TcpLen: 20
32 32 30 xx xx xx xx xx xx xx xx xx xx xx xx xx         220- xxxxxxxxx
43 6C 75 62 20 46 54 50 20 53 65 72 76 69 63 65  Club FTP Service
xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx         xxxxxxxxxxxxx  R
65 61 64 79 0D 0A 32 32 30 20 50 6C 65 61 73 65  eady..220 Please
20 65 6E 74 65 72 20 79 6F 75 72 20 75 73 65 72   enter your user
20 6E 61 6D 65 2E 0D 0A                           name...
```

=+

Packet# 2

The clinet station sent an ACK TCP packet to acknowledge receipt of the last packet.

```
03/11-09:10:20.881509 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:48129 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x346976  Ack: 0x1384D  Win: 0x21E0  TcpLen: 20
```

=+

Packet# 3

The ftp client received the user name request passed to it from the network stack. It prompted the user for the user name. It got the user name and passed it back down the network stack to send the reply packet back to the server.

```
03/11-09:10:24.116165 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:48385 IpLen:20 DgmLen:52 DF
***AP*** Seq: 0x346976  Ack: 0x1384D  Win: 0x21E0  TcpLen: 20
55 53 45 52 20 74 65 73 74 0D 0A                USER test..
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Packet# 4

Knowing who the user was and checked to be existing, the server came back to ask for the password to authenticate the user.

```
03/11-09:10:24.136192 192.168.0.1:21 -> 192.168.0.2:1056
```

```
TCP TTL:128 TOS:0x0 ID:26624 IpLen:20 DgmLen:76 DF
***AP*** Seq: 0x1384D  Ack: 0x346982  Win: 0x222C  TcpLen: 20
33 33 31 20 55 73 65 72 20 6E 61 6D 65 20 6F 6B  331 User name ok
61 79 2C 20 4E 65 65 64 20 70 61 73 73 77 6F 72  ay, Need passwor
64 2E 0D 0A                                      d...
```

=+

Packet# 5

The client station sent an TCP ack packet to acknowledge the request.

```
03/11-09:10:24.286402 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:48641 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x346982  Ack: 0x13871  Win: 0x21BC  TcpLen: 20
```

=+

Packet# 6

Upon receipt of the password request, the ftp client asked the user for it and passed it back to the server.

```
03/11-09:10:27.631238 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:48897 IpLen:20 DgmLen:53 DF
***AP*** Seq: 0x346982  Ack: 0x13871  Win: 0x21BC  TcpLen: 20
50 41 53 53 20 73 6B 61 74 65 34 78 34 0D 0A     PASS skate4x4..
```

=+

Packet# 7

Now the user was authenticated successfully and the ftp client was informed.

```
03/11-09:10:27.641236 192.168.0.1:21 -> 192.168.0.2:1056
TCP TTL:128 TOS:0x0 ID:26880 IpLen:20 DgmLen:70 DF
***AP*** Seq: 0x13871  Ack: 0x34698F  Win: 0x221F  TcpLen: 20
32 33 30 20 55 73 65 72 20 6C 6F 67 67 65 64 20  230 User logged
69 6E 2C 20 70 72 6F 63 65 65 64 2E 0D 0A        in, proceed...
```

=+

packet# 8

Again, the client sent an ACK TCP packet to acknowledge the message.

```
03/11-09:10:27.791440 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:49153 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x34698F  Ack: 0x1388F  Win: 0x219E  TcpLen: 20
```

=+

The user is now able to do what's allowed on the server.

Packet# 9

As an illustration, the connection was torn down right after it's been established.

The user entered quit to the ftp client, which in turn sent the quit request to the server.

```
03/11-09:10:29.513926 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:49409 IpLen:20 DgmLen:46 DF
***AP*** Seq: 0x34698F  Ack: 0x1388F  Win: 0x219E  TcpLen: 20
51 55 49 54 0D 0A                                 QUIT..
```

=+

Packet # 10

The server agreed with that and replied with the done message back to the client.

```
03/11-09:10:29.523964 192.168.0.1:21 -> 192.168.0.2:1056
TCP TTL:128 TOS:0x0 ID:27136 IpLen:20 DgmLen:81 DF
***AP*** Seq: 0x1388F  Ack: 0x346995  Win: 0x2219  TcpLen: 20
32 32 31 20 47 6F 6F 64 62 79 65 2E 20 43 6F 6E  221 Goodbye. Con
74 72 6F 6C 20 63 6F 6E 6E 65 63 74 69 6F 6E 20  trol connection
63 6C 6F 73 65 64 2E 0D 0A                       closed...
```

=+

Packet# 11

The client station replied with an ACK TCP packet to acknowledge receipt of the message.

```
03/11-09:10:29.533972 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:49665 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x346995  Ack: 0x138B9  Win: 0x2175  TcpLen: 20
```

=+

Disconnection Phase

Packet number 9 to 11 above in the data exchange phase is the message between the ftp server and client application. The TCP connection is still active. TCP server on both client and server ends will have to tear down the network connection.

Step#1

The ftp server sent a TCP packet with “FIN” flag set to signal the client station that the connection is done and will be disconnected.

```
03/11-09:10:29.534102 192.168.0.1:21 -> 192.168.0.2:1056
TCP TTL:128 TOS:0x0 ID:27392 IpLen:20 DgmLen:40 DF
***A***F Seq: 0x138B8  Ack: 0x346995  Win: 0x2219  TcpLen: 20
```

=+

Step# 2

The client station acknowledge with TCP packet with ACK and FIN flag set.

```
03/11-09:10:29.563987 192.168.0.2:1056 -> 192.168.0.1:21
TCP TTL:32 TOS:0x0 ID:49921 IpLen:20 DgmLen:40 DF
***A***F Seq: 0x346995  Ack: 0x138B9  Win: 0x2175  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Step# 3

The server sent back an ACK TCP packet and the connection was finally completed.

```
03/11-09:10:29.564234 192.168.0.1:21 -> 192.168.0.2:1056
TCP TTL:128 TOS:0x0 ID:27648 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x138B9  Ack: 0x346996  Win: 0x2219  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

DoS/RS (Denial of Service/Resources Starvation) Attack

A denial of service (DoS) attacks is an action with a malicious intent to significantly degrade the quality of a service that the target can normally be providing.

Resources starvation (RS) attack is a type of DoS which main purpose is to consume as much as possible the system resources on the target system/s with relative light load on the attacking system. To make this happen, the DoS/RS tools usually rely on raw packet handling technique rather than the system native TCP/IP stack on the attacking system. Naptha and Netkill are two DoS exploit examples using this technique.

TCP Vulnerability to DoS/RS

TCP is a stateful process with finite states. In normal operation over network, the time between triggers from one state to the other can vary significantly depending on the actual situation. It’s in general worse in the Internet than intranet. The Internet does not guarantee the arrival time from one packet to the next one and sometimes the packet may be arriving the destination out of sequence. TCP needs to deal with all these problems to provide a reliable end-to-end connection service.

To handle the data transfer properly, the TCP process has to:

1. Set with a relatively long time out for the connection in Internet so that the slow connections will not be pre-maturely disconnected. The kernel will have to keep a record of every TCP connection for a relatively long time. The more

connection started/established, the more system memory will be required to hold all the necessary connection information.

2. Buffer the incoming data to make sure the data packet can be re-assembled in the proper order before passing to the application running above. This is required because TCP provides reliable end-to-end connection service based on an unreliable lower network layers.

Combinations of the above factors make TCP vulnerable to DoS/RS attacks. It requires system memory to hold data to reassemble the data payload in the proper sequence. It also needs to keep the run time states of all TCP connections. Thus, by creating a large number of open TCP connections and leaving them in a certain states, the operating system itself can be starved of resources to the point of failure to maintain its normal service level.

Naptha - Denial of Service attack, Resources Starvation type

Naptha attack was documented around November 2000 by Robert Keyers. Its information can be found at web site:
http://razor.bindview.com/publish/advisories/adv_NAPTHA.html

The main function of Naptha is to exhaust TCP connection of the victim systems with very minimum load on the attacking system. The end result is that the victim system will eventually fail to maintain its service level.

Naptha is a proof of concept experimental DoS/RS exploit. To make the attack efficient, Naptha crafts its own raw packets. These programs do not use the on system TCP/IP stack. The purpose is to minimize the resource consumption on the attacking system. It contains three programs: bogusarp, synsend and srvr.

Codes Description

These programs use libpcap and libnet libraries to capture incoming packet and send packet respectively.

Bogusarp

It is used on a LAN to tie the forged IP address to the MAC address of the attacking station. It basically arps every 6 seconds trying to cheat the router to add the IP/MAC pair to the ARP table in it.

Synsend

It sends sync packet to the victim systems. Based on the user input, it uses libnet library function calls to send SYN back out on a regular basis as determined by the user. The user can set in the source IP, destination IP, destination port and the time between each send.

Srvr

It monitors the incoming packets for user pre-defined flag patterns and responses with user pre-defined flag patterns. It uses libpcap library function calls to capture incoming packets and libnet library for sending packets. User can define the source IP and the in/out flag patterns.

By running a copy or multiple copies of synsend and srvr, attackers can craft different attacking patterns to determine the best strategy to DoS the target systems.

Exploit Evaluation

The purpose of this evaluation is to understand:

- If Naptha is able to launch DoS successfully.
- System/network behavior by running different attacking patterns using different TCP flags.
- Level of system resources required on the attacking.

Preparation

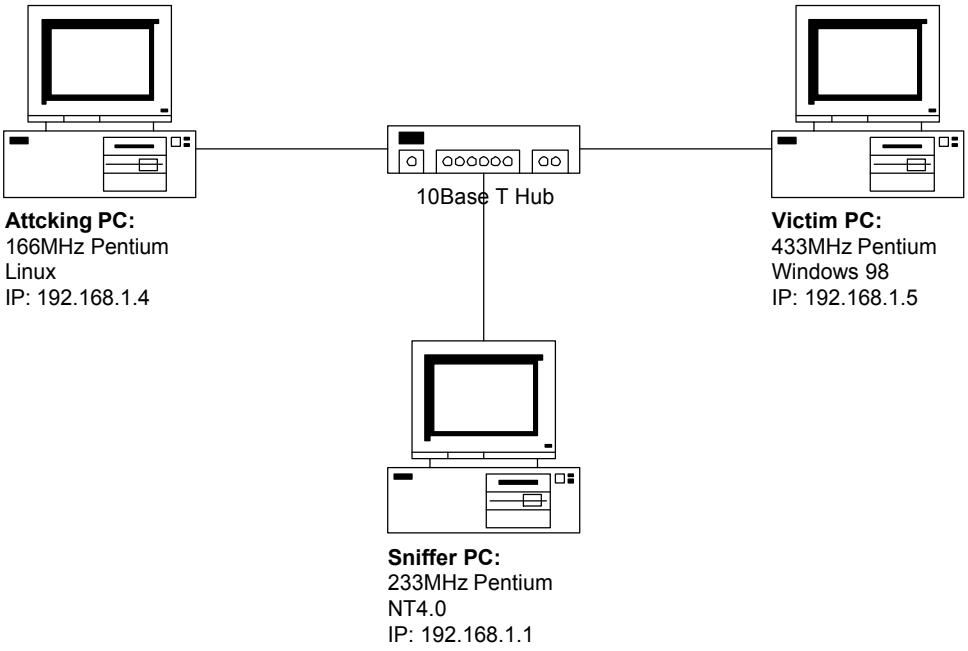
To make the exploit effective, the following is required to be setup:

1. The local TCP/IP stack must be be shielded from the incoming packets to the attacking system. Otherwise, the system's TCP/IP stack may reset the incoming packets, making the exploit defeated. Some sort of firewall setup will be an easy way out.

2. The attacking system must be able to run multiple programs effectively. Linux is a good choice.

Network Setup

To experiment with the Naptha exploit, I setup the computer/network systems as following:



Computer Setup

Attacking PC:
- Running one synsend
- Running one srvr
- All above at the same time
- Ipchains

Victim PC:
- AnalogX simple www server to keep the http connection valid
- A small batch file, cn.bat, to show the netstat output for http connection

Sniffer PC:
- snort
- IE Web-browser

Program Setup

Exploit:
- ./synsend 192.168.1.5 80 192.168.1.4 [time_sleep]
- ./srvr –[rcvr/send flag] 192.168.1.5

User needs to fill in parameters inside square brackets.

Firewall on attacking PC:
- ipchains –A input –I eth0 –s 192.168.1.5/32 –d 192.168.1.4/32 –j REJECT

Snort:
- network sniffing: "snort –N –dvi 2", "2" is the eth0 interface on the sniffer PC.
- IDS for Naptha: "snort –Afull –c snort.conf –i 2", Naptha signature included in snort.conf

AnalogX www server:
- simple start with a local webpage

Netstat update:
- Windows batch file, called cn.bat
  - :again
    netstat –an | find /N ":80"
    goto again

Testing Procedure

1. Activate AnalogX web server on victim system
2. Run cn.bat on victim system
3. Without rejecting incoming packet at the attacking PC
   Run "ipchains –F input" to flush the input control to study the native TCP/IP stack behavior.
   Run srvr with rcvr/send flag as needed.
   –SAsf: receive Syn/Ack, reply Syn/Fin
   Run synsend with time as needed. 1000 is a good standard point. The program will sleep one millisecond in between sync packet send.
4. Apply firewall to reject incoming packet to the attacking workstation
   Run "ipchains –A input –I eth0 –s 192.168.1.5/32 –d 192.168.1.4/32 –j REJECT"
   Run srvr with other flag combination to see the effect
   –SAsf: receive Syn/Ack, reply Syn/Fin
   -SAs: receive Syn/Ack, reply sync
   -SAsa: receive Syn/Ack, reply Sync/Ack
   Run synsend with different sleep time to see the effect

Test Result

Without Rejecting Incoming packet to attacking workstation

1. The exploit is not effective at all as the Syn/Ack packet was reset by the attacking system itself . I believe it's rejected by the native TCP/IP stack

because it knows it hasn't initiated the connection.

Below is the snort capture of typical traffic for this situation.

```
03/25-16:52:59.546797 192.168.1.4:35589 -> 192.168.1.5:80
TCP TTL:67 TOS:0x10 ID:413 IpLen:20 DgmLen:40
******S* Seq: 0x5C7B2A  Ack: 0x0  Win: 0x200  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

03/25-16:52:59.546841 192.168.1.5:80 -> 192.168.1.4:35589
TCP TTL:128 TOS:0x0 ID:32361 IpLen:20 DgmLen:44 DF
***A**S* Seq: 0xE8EB76  Ack: 0x5C7B2B  Win: 0x2180  TcpLen: 24
TCP Options (1) => MSS: 1460

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

03/25-16:52:59.546879 192.168.1.4:35589 -> 192.168.1.5:80
TCP TTL:255 TOS:0x0 ID:38800 IpLen:20 DgmLen:40
*****R** Seq: 0x5C7B2B  Ack: 0x0  Win: 0x0  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

See that connection always ends up with a reset packet. This did not create any resource starvation effect to the victim system. The cn.bat showed no new connection created in the netstat output.

Therefore, this test result suggests that the native TCP/IP stack will defeat the effect of naptha. The incoming packing to the attacking system must be hidden from it.

Rejecting Incoming Packet to Attacking Workstation

1. Running the same test setup but with ipchains to reject packet to attacking system

Below is a typical snort output.

```
03/25-17:09:26.076194 192.168.1.4:29496 -> 192.168.1.5:80
TCP TTL:67 TOS:0x10 ID:413 IpLen:20 DgmLen:40
******S* Seq: 0x5C7B2A  Ack: 0x0  Win: 0x200  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

03/25-17:09:26.076237 192.168.1.5:80 -> 192.168.1.4:29496
TCP TTL:128 TOS:0x0 ID:58624 IpLen:20 DgmLen:44 DF
***A**S* Seq: 0x23C4D  Ack: 0x5C7B2B  Win: 0x2180  TcpLen: 24
TCP Options (1) => MSS: 1460

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

03/25-17:09:26.086212 192.168.1.4:29496 -> 192.168.1.5:80
TCP TTL:67 TOS:0x10 ID:413 IpLen:20 DgmLen:40
***A***F Seq: 0x5C7B2B  Ack: 0x23C4E  Win: 0x200  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The cn.bat showed that there're 94 connections at CLOSE_WAIT state. This number is the same as the count shown at the srvr output.

Snort output shows that Windows98 killed further connection requests by responding with an Ack/Rst instead of Ack/Syn.

```
03/25-17:15:13.645507 192.168.1.4:41565 -> 192.168.1.5:80
TCP TTL:67 TOS:0x10 ID:413 IpLen:20 DgmLen:40
******S* Seq: 0x5C7B2A  Ack: 0x0  Win: 0x200  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

03/25-17:15:13.645560 192.168.1.5:80 -> 192.168.1.4:41565
TCP TTL:128 TOS:0x0 ID:5132 IpLen:20 DgmLen:40
***A*R** Seq: 0x0  Ack: 0x5C7B2B  Win: 0x0  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

As observed using cn.bat, five to ten minutes after the attack started, the connection state changed from CLOSE_WAIT to LAST_ACK. And then about the same amount of time later, all the connections were torn down. And the http service became available again.

Testing the Effect of different Attack Rate and Reply Flag Combination

To gain a better understanding of what can be done with this exploit, testing using a combination of time and flag combination were done. Test result is tabulated below.

Reply flag used	**Time between syn, in microsec**	**Connection status, netstat**	**Http Access from Sniffer System**	**Remark** Red: from attacking system
SYN	1 - 1000	Not showed up	The http access was incorrectly blocked by srvr because it only checks the source IP that is this web server.	The flag seq: S -> SA -> S -> SR
FIN	1000 –10,000,000	5 Syn_received	When the sleep time is from 5 to 10 seconds, the access is most of the time okay.	First 5: S -> SA -> F Rest: S -> AR

SYN/FIN	1000	No new connection	Same as SYN case above. This is mainly due to the situation that srvr corrupted the connections.	S -> SA -> SF -> AR
ACK/FIN	1000	94 new connections with CLOSE_WAIT state. Then changed to LAST_ACK and disappeared.	No access allowed even when synsend and srvr were turned off. DoS succeeded. But the system was not crashed as it sent RST when there were 94 http connections. System took about 15 minutes to recover even the attack was turned off.	First 94 connections: S -> SA -> AF Rest: S -> AR

Computing Resources Needed on Attacking System

Running the Linux “top” tool indicates that the computing resources consumption on the attacking system is very low.

Under the following running condition, the CPU % was under 0.04% of a 166MHz CPU (as the output is 0.0%) and the memory % was a total of 2.8% of 48MB physical memory.

Conditions:
- synsend running at the highest rate with sleeping time of 1 microsecond.
- Srvr was configured to run every time synsend run.

Naptha Signature

Network

This is only a proof of concept experimental exploit. The following signatures are

known:

IP:	TOS	=	Low delay
	ID	=	413
TCP:	Flags	=	SYN
	Seq ID	=	6060842
	Window	=	512

These signatures are not very reliable for IDS purpose. It is very easy for attackers to make minor change to these signatures to defeat the intrusion detection based on them. But it's not an uncommon problem to most if all the exploits.

The following rule can be set to snort to alert the presence of Naptha:

Alert tcp any any <> any any (flags:S;seq:6060842;id:413;msg:"Naptha DoS Attack)

Below is one of the alerts generated by running "snort -Afull -c snort.conf -i 2". "2" is the interface number as I run snort on NT. The rule above was included in myrule-lib, which is defined to snort.conf.

```
[**] Naptha DoS Attack [**]
03/25-19:27:26.872096 192.168.1.4:44740 -> 192.168.1.5:80
TCP TTL:67 TOS:0x10 ID:413 IpLen:20 DgmLen:40
******S* Seq: 0x5C7B2A  Ack: 0x0  Win: 0x200  TcpLen: 20
```

Victim System Signatures

1. The server under attack is having a large number of open connections For my test case, it's http connection at port 80 in FIN_WAIT_1 state. The actual number of ports opened is depending on the system on which the Web server is running. It is about 95 open connections allowed in my test Windows 98 system at CLOSE_WAIT/LAST_ACK state.

2. One very obvious symptom is that users are complaining about sluggish performance if the server runs at all. For my case, I was having http access problem from the sniffer PC to the Windows 98 system.

Protection Against It

As with other DoS type of attacks, there's no easy way to prevent the happening

of Naptha attack. But the following strategies can be used to minimize the security exposure to Naptha.

From preparation point of view:

1. Limit the number of services running on any systems.

2. Try not to run multiple applications on a single server so as to make the control and intrusion detection more effective.

3. Tighten the perimeter defense a much as possible.

4. Make sure all the border equipment are properly configured, patched and audited on a regular basis.

5. Fine-tune the servers' TCP timeout and keep-alive parameters to best protect the system from potential resources starvation attacks.

6. If possible, put in as much physical memory and other required system resource as possible as their costs are in general quite affordable now. This is justifiable as the potential services outage can potentially cost a lot more.

From detection point of view:

1. Unusual high number of repetitive access from a small number of sites to the target server requesting the same service is suspicious. To further verify if it's Naptha related, the traffic pattern should be reviewed against the signature mentioned above.

2. The subnet of the attacking sources should be closely monitored and the network activities from all the hosts from the same subnet should be logged and reviewed to make sure the system is safe.

3. If server-monitoring agent is running on these servers to monitor the resources usage, setup the alert threshold properly so that alert will be sent before the system is resources exhausted.

4. Indication of Naptha or similar kind of DoS attack taking place on one system may just be the beginning. With proper monitoring setup, the system administrators will be alerted to be able to act proactively before the problem is widely spread to more servers.

From reaction point of view:

1. The incident handler should identify the attacking sources to determine if it's appropriate to block them and how soon. Once the decision to block is made, the rule on the firewall can be changed to stop the offense.

2. It may be worthwhile to consider contacting the up-steam ISP to discuss the situation to see if they could stop the traffic at their end. It may be beneficial to them by making them be aware of the situation before the attack turns to them.

From lesson learned point of view:

1. The whole incident should be reviewed and properly documented.

2. The vulnerability that discovered from this incident should be corrected to prevent it from happening again.

3. The incident and the experience gained from handling it could be converted into good training material for the others in the organization.

Observation

Denial of service attack against Windows98 Web hosting system was proven to be effectively possible with Naptha. The DoS is in the form of blocking the web server to accept more new http connection. The test did not crash the system, which seems to be able to block the attack flooding.

The most effective attacking pattern in my test setup is to initiate http connections at the fastest rate and use Ack/Fin to reply to Syn/Ack from the victim system to clog the web service. Once the connection quota on victim system was fill up, it took about 15 minutes for the system to recover even attacking had been stopped.

Resource starvation attack is considered successful as test result demonstrates that a very light system resources running on the attacking system was able to stop the web server from responding new service requests.

The programs are easy to use. Attacker can easily run multiple pieces of each to craft a tailor made attack patterns according to the behavior of the victim system.

This exploit concept can be applied very well to Internet attack. Test data collected above indicates that a low-end system is able to launch a DoS attack against more powerful servers exist in the Internet.

Discussion

Given the widely available network library functions for most of the popular programming languages, writing DoS or other types of exploiting scripts are quite easy. Therefore, the situation will only be getting worse. Naptha demonstrates this point quite well. It's simple and requires very low computing resources on the attacking system. The ever-increasing network bandwidth that Internet can provide will not make the situation better.

So, how are we going to deal with this problem? One thing, from individual organization perspective, I can think of is to implement IP and subnet base network traffic throttle control at the appropriate network choke point device and/or on the server so that no single IP or subnet will be able to dominate the service supplies on the protected servers. Also, the Internet community should be acting together to fight against these attacks by implementing ingress and egress filtering technique to alleviate the situation.

Reference

1. Computer Networks, third edition, written by Andrew S. Tanenbaum, published by Prentice Hall.

2. Razor, Naptha DoS vulnerability, written by Robert Keyes, November 2000.
http://razor.bindview.com/publish/advisories/adv_NAPTHA.html

3. Strategies for Defeating Distributed Attacks, Written by Simple Nomad, January 2000.
http://razor.bindview.com/publish/papers/strategies.html

4. RFC2267, Network Ingress Filtering: Defeating Denial of Service Attack

5. CERT Advisory CA-2000-21 Denial of Service Vulnerabilities in TCP/IP stacks
http://www.cert.org/advisories/CA-2000-21.html

6. Vulnerability Discovered In Microsoft IIS Version 5.0, Microsoft, March 2000.
http://www.microsoft.com/technet/support/kb.asp?ID=241520
http://www.microsoft.com/technet/security/bulletin/MS01-016.asp

7. BugTraq, netkill, generic remote DoS attack, written Stanislav Shalunov, Apr 2000. http://www.securityfocus.com/templates/archive.pike?list=1&mid=56462&_ref=65481769