Global Information Assurance Certification Paper

NETWORK TRAFFIC INTERCEPTION ON A SWITCHED LAN ENVIRONMENT

PRACTICAL ASSIGNMENT
FOR
JOHN BURKE

GIAC Advanced Incident Handling And Hacker Exploits, SANS Network Security 2000, Monterey, California, October 2000.

EXPLOIT DETAILS

Name: Address Resolution Protocol (ARP) cache poisoning on a switched Local Area Network (LAN)

Variants: also known as ARP spoofing

Operating Systems: all

Protocols/Services: ARP

Brief Description: All hosts connected to a LAN use ARP to map IP addresses to network hardware addresses. Because ARP trusts all information it receives, an attacker on a switched LAN can intercept traffic destined for other users by sending spoofed ARP replies to other hosts on the switch. As a consequence, user authentication information (e.g. user login names and passwords) can be sifted from the traffic.

PROTOCOL DESCRIPTION

This exploit is capable of intercepting traffic from other directly connected switched LAN segments to yours. To better understand what this means, a short discourse into Ethernet technology, and the differences between shared and switched environments is required.

Ethernet Technology

Ethernet is the most prevalent LAN technology. There are several forms of Ethernet with the most prevalent today being 10BASE-T. 10BASE-10 is a 10 Megabits per second baseband LAN transmission over twisted pair copper wires. Baseband signals are transmitted by directly inserting voltage pulses onto the wire taking the entire bandwidth of the wire.

Two major Ethernet properties are:

- it is a broadcast bus technology
- it has a distributed access control mechanism

All attached interfaces share the common communications channel (bus) and all interfaces see every transmission on the bus (broadcast). The access control mechanism or who has the right to transmit on the bus at any time is not centrally controlled. All interfaces share a common access scheme called Carrier Sense Multiple Access with Collision Detect (CSMA/CD). When a host wants to transmit, its interface first checks for the presence of a carrier wave on the wire. If it senses there is a carrier wave, then it means another interface is currently using the wire and that it must wait before transmitting. When the interface senses there is no carrier then it will begin transmitting. This access scheme attempts to avoid collisions, a condition when two interfaces transmit simultaneously.

However, when increasing numbers of interfaces compete for access, the possibility of a collision increases. Collisions cause data errors. When the Ethernet CSMA/CD mechanism determines that a collision has occurred during its transmission, it will automatically abort the transmission, wait for the activity to subside, and then attempt to retransmit. In fact, Ethernet has a rather complicated backoff strategy that includes random delays

before retransmission. As you can imagine, a lot of collisions can cause for network throughput to suffer.

Another factor in collision avoidance is that each transmission by an interface is of a limited duration. Transmission will only take place to put one Ethernet protocol data unit (PDU) on the wire. PDU is a generic atomic data structure described by the protocol. Figure 1 describes the Ethernet PDU or frame.

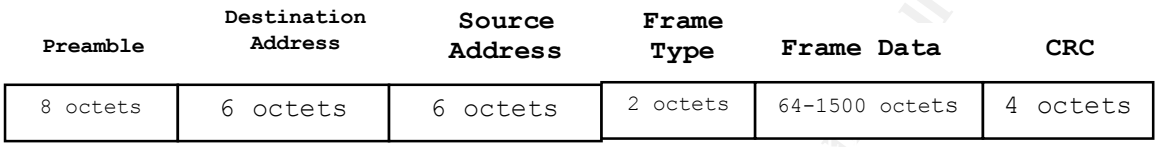

Figure 1. The Ethernet frame format. The preamble is the first part of the frame to be transmitted on the wire. Network interfaces sense frames on the wire by keying on the preamble. It was good design to have the destination address be the next frame element. This allows the interface to make a quick decision on the relevance of the frame. The final element, CRC, is an acronym for Cyclic Redundancy Check. The CRC helps the interface detect errors.

Of particular interest is the form of the source and destination addresses. They are Ethernet hardware addresses and are 6 octets or bytes long. Each network interface is assigned a unique hardware address by its vendor.

Network interfaces in standard operation only process and pass up the data portion of Ethernet frames that are hardware addressed to it. However, when put into a special promiscuous mode they will process and pass any valid frames they see regardless of the hardware address. One limiting factor in the ability to sniff network traffic is that the root user is the only user who is allowed to put the network interface into promiscuous mode. An attacker will therefore need to have special user privileges on a box to sniff network traffic. This can be accomplished if the user was able to first elevate his or her own user privilege to root level or has brought in a personal host onto the network where root access has already been established.

Shared Ethernet LAN Environment

Most legacy LANs are shared segments where all attached network interfaces have equal access to all

network traffic. Hubs are a mature and inexpensive network infrastructure that support shared LAN segments.

The security impact of using shared Ethernet is that its common communications channel allows any user on that LAN to monitor the entire LAN traffic if so desired. This is a fundamental property of Ethernet, and in the shared LAN segment it is unavoidable.

The shared LAN environment also has its performance limitation. One shared LAN segment can become saturated when the number of competing interfaces reaches a certain point. The maximum feasible number of users on one shared LAN segment is related to the duty cycle of the average user. Duty cycle is a ratio of the time an interface transmits and the total time.

Switched Ethernet LAN Environment

The switched Ethernet LAN eliminates almost all contention for the wire by establishing separate LAN segments on each port on the switch. Each segment gets the entire Ethernet bandwidth of the wire. Throughput performance does not take a hit as more interfaces populate the ports at the switch. Each switch can support a limited number of ports.

Most switches operate on the data link layer of the 7-layer Open System Interconnection model. They make switching decisions based on the destination hardware address of the Ethernet frame. There are also some network layer varieties that switch based on IP address, but they aren't as common and it is unknown whether or not arpredirect would work on them. A switch has electronics that monitors each port and keeps a table of all the hardware addresses it sees on each port. It is not unusual for a switch to see more than one hardware address active on each port. Network designers can put an entire shared LAN segment on each switched port. It is a common way to better distribute network traffic loads among multiple shared LAN segments. There is memory in the switch that is typically capable of storing thousands of hardware addresses it learns through tracking the ARP message traffic.

The security benefit of the switched LAN segment is that the network interface on a segment doesn't get traffic that is not hardware addressed to that interface. Therefore, it is thought of as more secure than using a shared LAN.

But, what if it was possible for you to trick other hosts on your switched LAN into thinking your hardware address was the hardware address for other hosts they want to communicate with? Dumb switches don't care if hardware addresses change on its ports or even if some ports share the same hardware address. The switch would just redirect frames originally intended for other hosts to yours. Spoofing ARP reply messages does this. Read the following section for background on ARP.

Address Resolution Protocol

Before a host can communicate with another host, it must determine the hardware address for that other host through the ARP request and reply process. The sending host will build the ARP request message as in Figure 2. It also shows how the ARP message is encapsulated within the frame data area of an Ethernet frame. ARP request messages

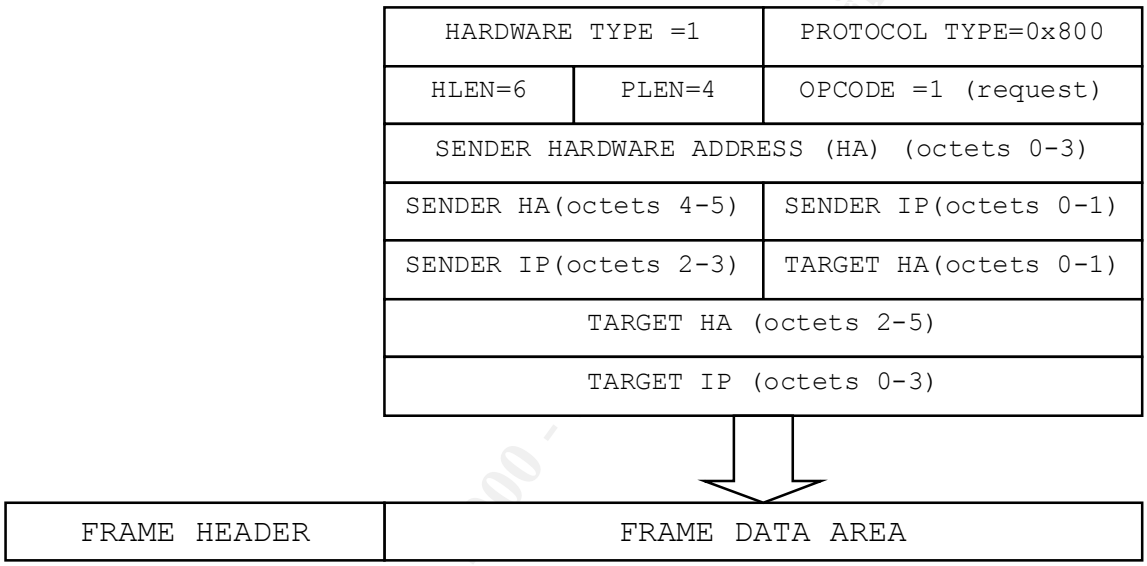

Figure 2. The encapsulated ARP request

are sent to the hardware address FF:FF:FF:FF:FF:FF. This is a special broadcast address that causes all live interfaces to process the frame and pass the ARP request to the host OS. All interfaces that receive the ARP request make a note of the sender's hardware and IP addresses in its ARP cache. If a host's interface has the IP address asked for in the ARP request it and only it should send an ARP reply. The ARP reply message has the same format as the ARP request, except that the OPCODE field in the ARP header is equal to 2. The ARP reply is sent back only to the sender's hardware address and contains the requested replier's hardware address.

Each host maintains an ARP cache of all recently learned IP and hardware address pairs to minimize the number of broadcast ARP requests transmitted. Before

transmitting a packet a host always checks its ARP cache to prevent unnecessary ARPing.

The security weakness that the ARP cache poisoning exploit uses is ARP's total lack of any security measures. ARP fully trusts all ARP replies that fit the protocol header form and will update its cache immediately when it receives new information.

Because it was designed with no security requirements in mind, ARP fully trusts all ARP messages it sees. Of course, the ARP packet must be correctly framed and conform to the protocol, but this is easy for an exploit to overcome. Whenever a host running ARP sees an ARP message it checks it cache for the sender's IP address. If it's already in its cache then the host will overwrite the corresponding hardware address with the hardware address from the ARP packet. It doesn't care if the hardware address has changed and it doesn't check it for authenticity. This is the security shortcoming that this exploit utilizes.

DESCRIPTION OF VARIANTS

ARP cache poisoning is also know as ARP spoofing. The attacker's objective is to modify or "poison" the current ARP cache of other hosts and tell the switch that your hardware address has changed. The attacker's means to this end is ARP spoofing or masquerading as another network interface.

Several tools exist to implement this exploit:

ARP0c2 (Linux)
dsniff, or more accurantly, arpredirect (Unix)
smit (Linux)
THK-Parasite (Linux)
WC1 (Windows)

HOW THE EXPLOIT WORKS

In order to verify operation of the exploit, a small switched LAN was set up as shown in Figure 3. Three Linux hosts were set up on a common switch. The three Linux hosts are:

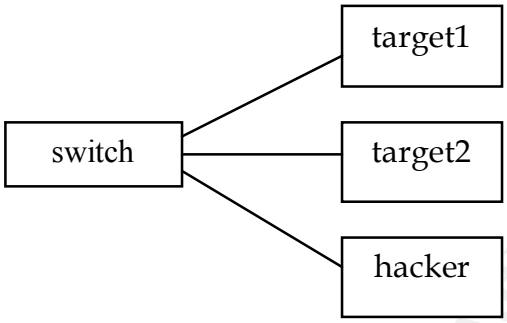

Figure 3. The test switched LAN

• target1

```
eth0      Link encap:Ethernet  HWaddr 00:10:4B:D9:D6:3D
          inet addr:192.168.1.1  Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

• target2

```
eth0      Link encap:Ethernet  HWaddr 00:A0:C9:56:43:66
          inet addr:192.168.1.2  Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

• hacker

```
eth0      Link encap:Ethernet  HWaddr 00:50:04:8B:C8:A5
          inet addr:192.168.1.3  Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

The switch used was a Bay Networks NETGEAR Model FS105 dual 10/100 Mbps fast Ethernet switch.

Target2 was set up to repeatedly send a file to target1 using netcat in a crontab so some network traffic would be continuously running in the background. As expected on the switch, hacker could normally not see the data transfer between target1 and target2. The hacker box

was used to run the exploit in an attempt to sniff the data transfer. Tcpdump was run on hacker to sniff traffic seen by hacker.

Here is a display of the packets from hacker's viewpoint showing the exploit in action:

1. 0:50:4:8b:c8:a5 Broadcast 42: arp who-has 192.168.1.1 tell hacker
2. 0:10:4b:d9:d6:3d 0:50:4:8b:c8:a5 60: arp reply 192.168.1.1 is-at 0:10:4b:d9:d6:3d
3. 0:50:4:8b:c8:a5 0:10:4b:d9:d6:3d 42: hacker.1024 > target1.67: udp 0
4. 0:10:4b:d9:d6:3d 0:50:4:8b:c8:a5 70: target1 > hacker: icmp: 192.168.1.1 udp port 67 unreachable [tos 0xc0]
5. 0:50:4:8b:c8:a5 Broadcast 60: arp reply 192.168.1.1 is-at 0:50:4:8b:c8:a5
6. 0:a0:c9:56:43:66 0:50:4:8b:c8:a5 1514: target2.1234 > target1.4766: tcp 1448 (DF)
7. 0:50:4:8b:c8:a5 0:10:4b:d9:d6:3d 1514: target2.1234 > target1.4766: tcp 1448 (DF)
8. 0:a0:c9:56:43:66 0:50:4:8b:c8:a5 1514: target2.1234 > target1.4766: tcp 1448 (DF)
9. 0:50:4:8b:c8:a5 0:10:4b:d9:d6:3d 1514: target2.1234 > target1.4766: tcp 1448 (DF)
10. 0:a0:c9:56:43:66 0:50:4:8b:c8:a5 1514: target2.1234 > target1.4766: tcp 1448 (DF)

•
•
•

11. 0:a0:c9:56:43:66 0:50:4:8b:c8:a5 1514: target2.1234 > target1.4766: tcp 1448 (DF)
12. 0:50:4:8b:c8:a5 0:10:4b:d9:d6:3d 1514: target2.1234 > target1.4766: tcp 1448 (DF)
13. 0:10:4b:d9:d6:3d Broadcast 60: arp reply 192.168.1.1 is-at 0:10:4b:d9:d6:3d
14. 0:10:4b:d9:d6:3d Broadcast 60: arp reply 192.168.1.1 is-at 0:10:4b:d9:d6:3d
15. 0:10:4b:d9:d6:3d Broadcast 60: arp reply 192.168.1.1 is-at 0:10:4b:d9:d6:3d
16. 0:50:4:8b:c8:a5 0:10:4b:d9:d6:3d 42: arp who-has 192.168.1.1 tell hacker
17. 0:10:4b:d9:d6:3d 0:50:4:8b:c8:a5 60: arp reply 192.168.1.1 is-at 0:10:4b:d9:d6:3d

Frame 1 shows hacker's broadcast ARP request for target1's hardware address. Frame 2 is target1's reply. Since hacker's ARP cache did not include target1's IP and hardware address pair, frames 3 and 4 are a result of code conditionally compiled into the exploit if you are compiling for Linux. The exploit needs to force the Linux kernel to ARP. So, the program sends a BOOTP packet to target1. This will force the kernel to first ARP for target1 as shown in frame 1. In frame 4, target1 sends an

ICMP destination unreachable back to hacker since it is not running BOOTP.

Now for the subversion, frame 5 shows the hack in action: it is a bogus ARP reply broadcast to the LAN falsely stating that target1's hardware address is the same as hacker. Frame 2 and frame 5 tell a different story, but the switch, and all other interfaces read both. From that point until the exploit restores the original configuration, the switch will send frames addressed for target1 to hacker.

Frames 6 and 7 show hacker's IP forwarding working to get one IP packet it is intercepting to its intended target. Frame 6 is from target2's hardware address and it is destination hardware addressed for hacker. Hacker's IP forwarding will repackage the IP packet, decrementing the IP Time To Live field, and put it in frame 7 hardware addressed for target1.

When the user has had enough sniffing a Control-C signal will cause the exploit program to clean up after it. It will send three ARP replies with the correct hardware address for target1. Notice these three ARP replies (frames 13-15) are spoofing target1's hardware address. Then, to be sure the switch is back to its original configuration, the exploit will ARP request for target1's hardware address (frame 16). Finally, target1 is nice enough to oblige with an ARP reply, and all is right again.

HOW TO USE THE EXPLOIT

Arpredirect is the program used to execute this exploit.

A major prerequisite for using arpredirect effectively is having IP forwarding turned on at the host on which you are using it. That "attack" host will be seeing, if successful, a lot of traffic that is not IP addressed to it. That traffic will not make it to its intended target unless your IP stack recognizes that the packets are not for you and stuffs them in another Ethernet frame hardware addressed for the real destination host. Another prerequisite would be to turn off Internet Control Message Protocol (ICMP) redirects at your "attack" host. I believe this can be done with most versions of Linux with the command:
"echo 0 > /proc/sys/net/ipv4/conf/eth0/send_redirects".

The /proc tree is not standard for all distributions of Linux out there so this command is not guaranteed to work for you, but you should have a similar path under /proc. Sending ICMP redirects when you receive IP packets not addressed to you is good practice normally, but you are trying to be as stealthy as possible in this hack.

The program itself is very easy to use with just a few command line options. Let's look at the proper usage as reported by arpredirect:

```
# ./arpredirect -?
Version: 2.2
Usage: arpredirect [-i interface] [-t target] host
#
```

Use the optional interface switch if you have more than one live network interface, otherwise the program will use your default interface.

The optional target switch is rather important to understand so you use it properly. If left off the command line, arpredirect will broadcast its bogus ARP reply to every interface listening on the LAN. This can be very dangerous since it potentially alerts someone on the LAN that you are fooling around. Remember that the consequence of using arpredirect is that some host or its operating system (OS) could recognize an anomalous condition on the LAN. It will appear, depending on the OS, that either two

IP addresses share the same hardware address or worse, that two hosts share the same IP address. The latter case is shown by FreeBSD hosts that treat its interface's hardware addresses as permanent and when a bogus ARP reply is seen it writes a entry in its syslog warning that two interfaces share the same IP address.

The mandatory host field specifies from which host you want to intercept packets. It could be the same as the target host, or if you feel ambitious, arpredirect's author says you could put the local gateway here and get the maximum amount of data.

Let's give arpredirect a typical run command to demonstrate what arpredirect looks like when executed. Say that a host named target1 is expected to use FTP to move a file to target2, then the least intrusive way to sniff the user's name and password on target2 would be:

```
# ./arpredirect –i eth0 –t target1 target2
arpredirect: intercepting traffic from 192.168.1.1 to 192.168.1.2 (^C
to exit )...

After sniffing is complete press control-c:

arpredirect: restoring original ARP mapping for 192.168.1.2
```

SIGNATURE OF THE ATTACK

Any means of ARP cache poisoning will be detectable by either host or network based intrusion detection systems (IDS). An IDS should track each local network interface by its hardware and IP address and monitor the ARP traffic for ARP replies where the SENDER HA field has changed for a given IP address. When an ARP reply with a changed SENDER HA field is detected look at the source hardware address of the frame. This hardware address points to the machine used by arpredirect to snoop on the switch.

HOW TO PROTECT AGAINST IT?

One way to protect LANs against ARP cache poisoning would be to lock them down into static network configurations. Load each host with a static ARP cache and avoid using ARP. This has a serious drawback that if you had to change the network interface in any host on the LAN, or put a new host on the LAN, you would also have to replace the ARP cache on every host. This would be administratively impossible except for the smallest LANs.

A better solution is to use a switch with security features. This paper shows successful ARP cache poisoning using an inexpensive 5-port switch. Without mentioning individual product names, there are many vendors who provide "secure ports" on their switches. Some common security features on the "secure ports" are:

- Restricting a port to a user-defined set of hosts
- Alerting when a port security address violation occurs
- Shutting down a port when a port security address violation occurs.

Of course, proper use of strong encryption on the LAN can make any sniffing attempts moot.

SOURCE CODE/PSEUDO CODE

Arpredirect source code compiles easily under Linux using the configure script supplied. However, arpredirect source code is only found with a set of other interesting tools by the same author called dsniff. Source code can be downloaded from the author's website http://www.monkey.org/~dugsong/dsniff. Instead of pulling out just the code for arpredirect and building just it; it's easier to build the entire dsniff package than to build arpredirect alone. Besides you will get a lot of other cool tools with arpredirect. Building dsniff requires:

- libpcap – http://www.tcpdump.org/
- libnet – http://www.packetfactory.net/Projects/Libnet/
- libnids – http://www.packetfactory.net/Projects/Libnids
- libdb – http://www.sleepycat.com/

To build dsniff on a Linux box:

1. untar the source code tar file: tar zxvf dsniff-2.2.tar.gz
2. cd dsniff-2.2/
3. ./configure
4. make
5. make install

Pseudo code Analysis

Execution chain:

1. Execution starts at main
   main() – passes in command line variables
   getopt() - command line variable handling
   libnet_name_resolve() – libnet function to handle name resolution
   pcap_lookupdev() – libpcap function for network interface
   libnet_open_link_inerface() – libnet function to open link-layer interface

2. ARP request for target
   arp_find() – ARP cache lookup

```
     arp_send() – uses libnet functions to build and send
ARP request message

3.  Spoofed ARP reply for target
     arp_send() – uses libnet functions to build and send
ARP reply message with local hardware address

4.  Signal handling code
     If SIGHUP, SIGINT, or SIGTERM caught, go to Cleanup
5.  Infinite loop
     arp_send() - uses libnet functions to build and send
ARP reply message with local hardware address
     sleep 2 seconds
6.  Cleanup code
     arp_send() – restores original ARP mapping
```

ADDITIONAL INFORMATION

REFERENCES

COMER, DOUGLAS E. [1995], Internetworking with TCP/IP, Volume 1 Principles, Protocols, and Architecture.

STALLINGS, WILLIAM [1996], Data and Computer Communications, 5th Edition.

Last Updated: May 26th, 2017

Upcoming Training

SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Thailand 2017	Bangkok, Thailand	Jun 12, 2017 - Jun 30, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS ICS & Energy-Houston 2017	Houston, TX	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
Community SANS Ottawa SEC504	Ottawa, ON	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Sacramento SEC504	Sacramento, CA	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
Community SANS Annapolis SEC504	Annapolis, MD	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Des Moines SEC504	Des Moines, IA	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Phoenix SEC504	Phoenix, AZ	Jul 24, 2017 - Jul 29, 2017	Community SANS
Security Awareness Summit & Training 2017	Nashville, TN	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
Community SANS Raleigh SEC504	Raleigh, NC	Aug 07, 2017 - Aug 12, 2017	Community SANS
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event