Global Information Assurance Certification Paper

Linux.Slapper.Worm: Buffer overflow attacks continue to be a problem

SANS GSEC Practical ver. 1.4b, Option 1
Richard H. Fifarek
Oct. 1, 2002

Abstract

The Linux.Slapper.Worm is yet another example of the security problems that programmers must consider when creating applications and libraries that are intended to be used via the network, and on a broader scale, the Internet.

Many programs that are written today, particularly for the UNIX operating system or UNIX-like operating systems, are written in the C programming language. The design of the C programming language created a framework that allowed buffer overflows to occur. In addition, lack of proper training of programmers and bad programming practices makes the buffer overflow possible. "Buffer overflow attacks are said to have arisen because the C programming language supplied the framework, and poor programming practices supplied the vulnerability" (searchSecurity.com).

OpenSSL is a popular library of functions that is used to encrypt transmissions of data over the network. It is often used in the very popular Apache web server to provide encrypted web communications, which is the means by which this worm propagated itself. The OpenSSL package is designed to help improve security, but yet, in turn, it made systems less secure. Numerous versions of the OpenSSL library contained four buffer overflow conditions that could be exploited to compromise a system over the network. Using these buffer overflows as a point of attack, someone designed a worm (called Linux.Slapper.Worm) that attacked and compromised systems, and then used that system to propagate itself to other systems. Approximately 13,900 machines were confirmed infected, as reported by F-Secure, Inc. (F-Secure).

Until programmers begin writing from a security first approach and systems administrators and vendors learn to be proactive about patching their systems, the computing industry will continue to be plagued by worms and remote security compromises in general.

Overview

On July 30, 2002, the OpenSSL Project team released a security advisory documenting vulnerabilities in various versions of the project's OpenSSL programming library. The affected versions were 0.9.6d and earlier, or 0.9.7-beta2 and earlier. Four of these vulnerabilities were explicitly documented because they were identified as remotely exploitable via a buffer overflow attack. While the Neohapsis computer security group had internally proven that exploits were possible, there were no known exploits "in the wild" at the time the report was released.

On September 13, 2002, Ben Laurie, the author of the original security advisory, posted the first reports of a worm that exploited the buffer overflow vulnerabilities in OpenSSL programming library to the software security mailing list Bugtraq. In his message, he said "Anyone who has not patched/upgraded to [OpenSSL] 0.9.6e+ should be _seriously worried_" (Laurie). He was correct.

The Linux.Slapper.Worm specifically targets machines running the Linux operating system, and its point of entry was the popular Apache web server, which used OpenSSL to provide secure web communications using encryption via the plugin module mod_ssl. The Apache web server is responsible for more

than 60% of the 35 million plus public websites on the Internet. Less than 10% of those websites use SSL services. Furthermore, only 50% of those use OpenSSL, which equates to slightly over 1 million sites that were potentially vulnerable (F-Secure). Analysis of the worm's method of attack generated some quick fixes that obscured vulnerable systems from attack by the worm. Unfortunately, these temporary fixes bought systems administrators little time as 2 "smarter" variants of the Linux.Slapper.Worm were released shortly after the original. While certainly not the fastest and most destructive worm, it proved that the problem of buffer overflows attacks was still an ongoing issue.

Buffer overflows and worms that exploit them are nothing new. The first publicly documented Internet worm that exploited a buffer overflow occurred in 1988, and 14 years later, they still regularly plague the computing industry (Litterio). Effective methods and tools exist to prevent or detect buffer overflows before they become a problem, but these aren't widely implemented. Some tools actually monitor the program while it is running, but these often seriously degrade the performance of the system. Programming techniques and specific functions that help prevent buffer overflows are well documented and easily available. Newer versions of compilers are beginning to flag dangerous functions and generate warnings regarding the use of those functions. However, these aren't being adopted, and the computer industry continues to see problems.

Unfortunately, education tends to always lag behind the "cutting edge" in any field. In the computer security and programming, this is particularly an issue, as advancements have followed a much more aggressive curve than many other industries. Until the computer education community begins teaching programming from a "security first" approach rather than "security as an afterthought" approach, these problems will continue to persist and likely get worse as the number of Internet connected hosts grows.

Also, vulnerabilities are often discovered, repaired, and made public before an exploit is created and actively used. In the case of the OpenSSL vulnerabilities, a functioning exploit wasn't "in the wild" for nearly a month and a half, giving many systems administrator's ample time to upgrade or patch their OpenSSL libraries.

OpenSSL

A well known, successful open source project, the OpenSSL project is an effort "to develop a robust, commercial-grade, full-featured … toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library" (OpenSSL Project). In short, the OpenSSL package allows one to build SSL and TLS functionality and the inherent security provided by encryption into network applications. As security on the Internet continues to be a growing concern, SSL and TLS encryption has proven to be an effective way to protect sensitive (and sometimes non sensitive) data while in transit between two machines via the network. In addition to supporting SSL and TLS, OpenSSL now includes support for interfacing with external crypto hardware. OpenSSL is a popular library for use in the web server Apache for providing secure web

communication, and in OpenSSH, a popular Telnet replacement that allows secure remote administration. OpenSSL is often used to provide secure communications in many open source software packages that require encryption. Any security flaw in OpenSSL reduces the effectiveness of the security it was designed to provide.

Worm

A computer worm is similar to a computer virus in how it replicates, and often contains some functionality that is designed to disrupt the normal functions of a computer (Virus Definitions). However, a worm does not attach itself to a program or file like a virus does, but rather is a standalone program or a separate entity that attempts to spread without help. A worm's objective is to propagate itself as quickly as possible, and as such, a worm will attempt to do as much of the propagation work without any intervention or assistance from a user as it can.

In order to propagate itself, a worm will often take advantage of automated tasks or functions, such as file sending and receiving (Virus Definitions). Many modern operating systems provide these services. Also, certain popular programs (typically email clients) have automatic functions built into them that are intended to make the program easier for the layman to use. Another method that worms use to propagate is to take advantage of a remotely exploitable vulnerability such as a buffer overflow that can be taken advantage of via the network. The Linux.Slapper.Worm is an example of a worm that uses this technique.

The Linux.Slapper.Worm will randomly begin scanning for hosts on the Internet, and when it finds one, it will attempt to determine if the host is capable of being attacked. If the machine is able to be compromised, the worm copies itself to the newly compromised system, and then both systems begin searching for other systems to attack. Using this method, a worm can spread very quickly.

Nicolas C. Weaver of the University of California Berkeley believes that a properly designed worm using intelligent propagation techniques could infect every vulnerable host within the IPV4 address space (more than 4 billion addresses) in a minimum of 15 minutes to a maximum of one hour (Weaver). Even the most proactive systems administrators could not be notified and able to take action to protect their systems within that short time frame. The potential of this Warhol worm is enormous, "capable of doing many billions of dollars in real damage and disruption" (Weaver). Furthermore, because of the speed with which the worm spreads, "human mediated responses offer almost no hope of stopping it", as systems administrators would likely not receive security advisories in time to prevent infection (Weaver). Fortunately, this hyper-virulent worm has yet to be seen in the wild.

While detecting worm activity isn't impossible, it often can appear as normal network traffic. A connection to port 80, the web server port, on a machine is not unusual, and is, in most cases, expected. Any network monitoring device, such as an Intrusion Detection System (IDS), would have to look for patterns within the network traffic to detect worm activity. The patterns may be discovered by monitoring the data transferred between the attacker and

victim machines, or by tracking the number of connections and type of connections to multiple machines. This is resource intensive, and often requires prior knowledge of the patterns before accurate detection can occur. As worms become more sophisticated, this becomes an increasingly difficult means of protection against them. The only truly effective way to stop a worm from propagating is to prevent infection by maintaining current patch levels that repair/remove the point of entry, in this case a buffer overflow.

Buffer Overflow

According to searchSecurity.com, "a buffer overflow occurs when a program or process tries to store more data in a buffer (temporary data storage area) than it was intended to hold" (searchSecurity.com). By overflowing a buffer, a program can begin to overwrite certain areas of memory that it typically wouldn't have access to. A properly designed buffer overflow attack will overflow the buffer and write new instructions to memory that the running program will eventually execute. By tailoring these instructions, an attacker can give themselves access to the machine that he or she wouldn't normally have available. In the case of a worm, the worm would use the access to infect the host, and then use that host to aid in the further propagation of the worm.

Using buffer overflows to attack machines is nothing new. One of the most well known, and perhaps the first to be publicly documented, buffer overflow attacks occurred in November 1988 in the form of the infamous "Internet Worm" (Litterio). This worm shut down approximately 6,000 machines, and in effect the entire Internet itself (Farrow). By using a buffer overflow in the finger service, the worm was able to infect the machine and then use the newly infected host to further propagate itself.

Buffer overflows occur primarily because of the design of the C programming language, and partially because of poor programmer's habits. The C language was designed to allow easy porting of programs to many hardware architectures, and to be "tight and fast". In short, the C programming language was designed to be very close to assembly language while still providing a layer of abstraction that would facilitate porting code to other architectures easily. Unfortunately, this type of design inadvertently created a framework that allows buffer overflow conditions to occur easily.

A computer categorizes things in two ways: code and data. It treats code as a set of instructions that tell it what to do, and it treats data as the item that it executes the instructions on or with. Unfortunately, a computer can't tell the difference between the two, and relies on the programmer or user to determine which is which. In order to run a program on a computer, the operating system must load the code and data the code needs into memory for execution. For certain applications, the code will get most of its needed data from another application through some form of communication, often via a network. This design isn't inherently dangerous because the remote application can only work within the constraints of the local program's original code.

The problem arises when a remote program does something that breaks those constraints and overwrites the set of instructions stored in the memory of

the local program. Since a computer can't distinguish between code and data, it will blindly execute commands stored in the data. A malicious remote program will first try to get the code to the targeted computer by sending it as input data for the buffer over the network. An improperly written program won't check the size of the input data to see if it is larger than the destination buffer, and thus will begin overwriting other areas of memory outside the boundary of the buffer. The malicious program will tailor the data such that the data written outside of the buffer contains instructions that the program will execute. These instructions vary in type and purpose, but in general, they perform actions that will allow the worm to infect the machine and then use that machine to attempt to infect other machines. Frequently, programs such as distributed denial of service (DDoS) tools are installed as well.

Given its age and popularity, the design of the C programming language is not going to change, so that leaves the burden of change on the programmers and the educators of programmers. Unfortunately, very little progress has been made in the education of the programming community, as shown by the Linux.Slapper.Worm and the continued success of buffer overflow attacks over the years. The only effective prevention of buffer overflow attacks is properly written software that checks for an attempted buffer overflow and deals with it accordingly, written by better educated programmers who avoid writing software that is susceptible to attack. The best defense against buffer overflow attacks is maintaining proper patch levels on the system.

Linux.Slapper.Worm

In an email message to Bugtraq, Ben Laurie reported that he had seen a worm that attacked OpenSSL, taking advantage of the buffer overflow problems reported earlier by him. He went on to say that the worm was specifically targeted at Linux systems, but he felt that variants that attacked other systems could exist, or could easily be created (pg. 1).

The Linux.Slapper.Worm (also known as the Apache/mod_ssl Worm) came in three variants. All three variants took advantage of the buffer overflow that existed in older OpenSSL versions by attacking vulnerable installations of the popular open source web server Apache on Linux operating systems. Many flavors of Linux were vulnerable, including SuSE, Mandrake, RedHat, Slackware, and Debian.

Each variant attempts to connect to port 80 on the victim computer. If the connection is successful, it will send an invalid GET request (a HTTP command used by web browsers). Each web server responds differently to an invalid GET request, and typically identifies itself in the response. If the invalid GET request is responded to by an Apache server, the worm will connect to port 443, the SSL port, and send the exploit to that port. The worm tries to execute a Linux shell code exploit that is specifically targeted at x86 (Intel) hardware platforms. The code requires /bin/sh in order to function, and then sends uuencoded source code files of itself to the /tmp directory on the victim machine. Once the files are uploaded to the victim machine, the worm decodes the files, and using gcc, compiles them into binary format. It then executes the binary file that was

created in /tmp. When executed, the IP address of the attacking computer is provided as an argument to the program. This creates a virtual peer-to-peer network of infected systems that is used to coordinate and launch a Distributed Denial of Service attack. The worm listens on a UDP port (each variant uses a different port number) in order to participate in the peer-to-peer network and await instructions. In order to continue propagation, the worm scans class B-sized networks, choosing the first byte of the address randomly from a list, and the second byte randomly.

The A variant listens on UDP port 2002. The files that it places in /tmp are .uubugtraq, .bugtraq.c, and .bugtraq. .uubugtraq is the uuencoded file, .bugtraq.c is the source code, and .bugtraq is the compiled binary of the worm.

The B variant listens on UDP port 1978. The files that it places in /tmp are .cinik.uu, .cinik.c, and .cinik. .cinik.uu is the uuencoded file, .cinik.c is the source code, and .cinik is the compiled binary of the worm. The B variant also behaves differently from the A and C variant in that it collects and sends information about the infected host to a yahoo.com address. Also, the B variant takes on the role of a virus and infects certain files on the victim machine by overwriting them with a copy of itself. It then schedules them for later execution in an attempt to reactivate the worm on that machine in the event that it is discovered and removed from its standard /tmp location. Due to a bug in the software, it is only able to infect a small number of files. Upon first infecting a machine, the B variant will look for the source code in the /tmp directory and if it doesn't find it, will download the code from a (now shutdown) web site in Romania. A modification of the B variant, referred to as the C2 (cinik version 2) variant, was discovered on October 2, 2002. The C2 variant had some bug fixes from the original B variant, and uses port 1812 instead.

The C variant is unique in that it contains two source code files. The files that it places in /tmp are .unlock.uu, .unlock, .unlock.c, .update.c, httpd, and update. .unlock.uu is the uuencoded file, .unlock is an archive file (.tgz format), .unlock.c and .update.c are the source code files, and httpd and update are the compiled binaries. This variant listens on UDP port 4156. The renaming the binary httpd is an attempt to hide the process by making it look like a regular Apache process when the "ps" command is executed. Another twist on the original is that "the C variant periodically listens on TCP port 1052 and provides a shell to an attacker who is connecting to the port."

In addition, all of the filenames of the variants begin with a ".". This is an attempt to prevent detection by hiding the files from listing by the ls command. The only way to see the files would be to run "/bin/ls -a", instead of the standard "/bin/ls".

Once specific details of the worms behavior and methods of attack were posted to Bugtraq, a number of people offered temporary fixes. One idea, offered by Ajai Khattri, was to modify the "Server:" HTTP response field which obscured a vulnerable system from detection by the worm (Khattri). The worm based its attack on the response generated by the server for identification of the type of system. Another idea posted by Sandu Mihai suggested using the worm's own peer-to-peer network against itself, periodically sending commands

to the infected hosts instructing them to kill the worm process with the killall command (Mihai). This technique could have been made more effective by sending instructions prior to killall that deleted the files in /tmp, and made the filenames symlinks to /dev/null to prevent possible reinfection, and then send the kill command to the worm's process. While the idea of using the peer-to-peer network against the worm was never implemented, a similar, but simpler technique was used. As reported by F-Secure Security Information Center, there was evidence that systems administrators were running automated scripts that would kill and delete the worm. Unfortunately, many of those systems would soon get reinfected (F-Secure).

Another worm named Linux.DevNull, which has been incorrectly called Linux.Slapper.Worm variant D because it also uses the OpenSSL vulnerability, was discovered on September 30, 2002 (F-Secure). Once this worm has successfully infected a host, it downloads and executes a shell script from a website. This script downloads a compressed executable from the same website, decompresses it, and executes it. The executable is an IRC client that connects to a particular chat channel, and accepts commands to run on the infected host. The worm then downloads another compressed file, and decompresses it. This file contains both an executable and a source code file. The worm attempts to compile the source code file, and then executes the executable. This executable begins to search for more vulnerable hosts, and attempts to infect them. This worm does not create a peer-to-peer network as Linux.Slapper.Worm does. Since this behavior isn't similar to the Linux.Slapper.Worm pattern, it is considered an entirely separate worm.

Detection and Removal

Detecting infection requires very little effort and skill. The author(s) of this worm didn't put much effort into hiding this worm from systems administrators.

1. Login as root and stop the Apache process (note: if you aren't running Apache, then it is very unlikely that you have been infected). This often can be accomplished by running the command "/etc/init.d/httpd stop" or the command "apachectl stop".
2. Change into the /tmp directory, and run the command "/bin/ls -a" (note: the -a argument to /bin/ls is important because ls' default behavior is to not list files that begin with a "."). If you see any of the filenames that were mentioned before, then you are infected.
3. Run the "/bin/ps -aux| less" and look for any process names that are unusual (note: if you have already stopped Apache, and you see a httpd process still running, then it is possible that you have been infected by the C variant. However check for other evidence to be sure).

Based on the filenames in /tmp, it should be possible to determine which variant has infected the computer. If it is the A variant (.bugtraq), then one should see a .bugtraq process running and the process id it is running under in the process list generated by the "/bin/ps -ef" command. By running "/bin/netstat -t inet -an" one should also see that UDP port 2002 is open. To kill the process, run "/bin/killall -9 .bugtraq". Run the "/bin/ps -ef" command again to verify that

the process is no longer listed. If it is the C variant of the worm, then the same steps apply, just replacing ".bugtraq" with "httpd" and a UDP port of 4156. With both variants, one will want to delete the worm files from /tmp, or at the very least move the files to a different directory for later viewing and investigation. All major Linux distribution vendors have released updates for their OpenSSL packages, so the process of patching a system should be simple. For instance, the updated OpenSSL package for RedHat 7.x is openssl-0.9.6b-28.<arch>.rpm, where <arch> is the hardware architecture. To update the package on RedHat 7.x on the Intel (x86) architecture, download the openssl-0.9.6b-28.i386.rpm file from RedHat's site, and then run "/bin/rpm -Uvh openssl-0.9.6b-28.i386.rpm". Once the package is updated, Apache can safely restarted.

With the B variant, the detection and removal process is similar, however one has to consider that the B variant infects other files, and tracking those files down may be difficult and time consuming. One method of doing this on an rpm based distribution like RedHat would be to use the /bin/rpm command's package verification options. "/bin/rpm --verify --all" will scan the files and compare them to its database and flag any of them have changed from what it expects. On a cautionary note, one must be certain that the rpm database hasn't been tampered with; otherwise the information will be suspect. Another tool is Tripwire, but it needs to be installed prior to infection by the worm. Also, consider that information about the infected machine was sent to a yahoo.com email address. Given all of this, wiping the system clean and installing a fresh operating system and software would be the safest option. This applies to all of the variants, but is especially significant if infected by the B variant. As always, after reinstalling the operating system and software, make sure that the system is patched to the latest patch level provided by the vendor.

Conclusion

While the Linux.Slapper.Worm affected a small percentage of the machines, it serves as yet another example of the problems still facing the computing industry. This worm took advantage of a buffer overflow in the OpenSSL library commonly used by the Apache web server to provide secure web communications. The first recorded worm that utilized a buffer overflow attack was launched in November 1988, and 14 years later, the computer industry continues to be plagued by them.

While the C programming language created the framework for buffer overflows to occur, the burden of responsibility falls on the programmers to avoid creating buffer overflows in their programs. In addition, the educational system is also responsible for teaching proper programming techniques to students. Programmers have to start writing programs from a "security first" approach rather than building security into the application as an afterthought. Unfortunately, security "holes" are something that are always going to be a problem, and as such, systems administrators need to be proactive in maintaining current patch levels on their systems when security advisories are released.

Even though each of the Linux.Slapper.Worm variants were either improved or modified versions of the original, all variants made infection fairly easy to detect. They created a peer-to-peer virtual network running over UDP that used non-standard ports that probably wouldn't get confused with other commonly used UDP ports. They placed all of their files in /tmp. The B variant had the potential to do the most damage because it not only tried to infect other machines, but it tried to infect local files. Fortunately, due to a bug in the worm's source code, it only met with mild success. Using tools such as rpm or Tripwire to verify signatures of the systems files, it is possible to track down the changes made by the B variant, and reinstall the software packages that were affected from known good sources. The threat of the Linux.Slapper.Worm was short lived, and infected a relatively small number of hosts. A suspect was arrested on suspicion of authoring the original worm, after having been tracked back to an email address in the Ukraine (Middleton).

Works Cited

"Computer Worm (Definition)." Virus Information. University Computing Services, Ball State University. 22 September 2002 <http://www.bsu.edu/ucs/article/0,1299,6313~448~1985,00.html>

Farrow, Rik. "Blocking Buffer Overflow Attacks." Network Magazine 1 November 1999. 22 September 2002 <http://www.networkmagazine.com/article/NMG20000511S0015>

"Global Slapper Worm Information Center." F-Secure Security Information Center. 14 September 2002. 6 October 2002 <http://www.f-secure.com/slapper>

Khattri, Ajai. "Re: Linux Slapper Worm." Online posting. 18 September 2002. 6 October 2002 <http://online.securityfocus.com/archive/1/292361>

Laurie, Ben. "OpenSSL worm in the wild." Online posting. 13 September 2002. 6 October 2002 <http://online.securityfocus.com/archive/1/291782>

Litterio, Francis. The Internet Worm Of 1988. 22 September 2002 <http://world.std.com/~franl/worm.html>

Middleton, James. "Arrest for Slapper author." vnunet.com. 24 September 2002. 6 October 2002 <http://www.vnunet.com/News/1135274>

Mihai, Sandu. "Re: bugtraq.c httpd apache ssl attack." Online posting. 16 September 2002. 6 October 2002 <http://online.securityfocus.com/archive/1/292013>

The OpenSSL Project. 23 December 1998. 22 September 2002 <http://www.openssl.org>

"searchSecurity.com Definitions: buffer overflow." searchSecurity.com. 1 May 2001. 6 October 2002 <http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci549024,00.html>

Weaver, Nicholas C. Warhol Worms: The Potential for Very Fast Internet Plagues. 15 August 2001. 22 September 2002 <http://www.cs.berkeley.edu/~nweaver/warhol.html>

Last Updated: June 24th, 2018

Upcoming Training

SANS Cyber Defence Canberra 2018	Canberra, Australia	Jun 25, 2018 - Jul 07, 2018	Live Event
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
Community SANS Nashville SEC401	Nashville, TN	Jun 25, 2018 - Jun 30, 2018	Community SANS
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
SANSFIRE 2018	Washington, DC	Jul 14, 2018 - Jul 21, 2018	Live Event
SANSFIRE 2018 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
Mentor Session - SEC401	Jacksonville, FL	Jul 17, 2018 - Aug 28, 2018	Mentor
Community SANS Annapolis Junction SEC401	Annapolis Junction, MD	Jul 23, 2018 - Jul 28, 2018	Community SANS
SANS Riyadh July 2018	Riyadh, Kingdom Of Saudi Arabia	Jul 28, 2018 - Aug 02, 2018	Live Event
SANS Pittsburgh 2018	Pittsburgh, PA	Jul 30, 2018 - Aug 04, 2018	Live Event
SANS Boston Summer 2018	Boston, MA	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS August Sydney 2018	Sydney, Australia	Aug 06, 2018 - Aug 25, 2018	Live Event
San Antonio 2018 - SEC401: Security Essentials Bootcamp Style	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 Hyderabad 2018	Hyderabad, India	Aug 06, 2018 - Aug 11, 2018	Live Event
Mentor Session - SEC401	Ankara, Turkey	Aug 08, 2018 - Oct 03, 2018	Mentor
Northern Virginia- Alexandria 2018 - SEC401: Security Essentials Bootcamp Style	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
SANS Northern Virginia- Alexandria 2018	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS New York City Summer 2018	New York City, NY	Aug 13, 2018 - Aug 18, 2018	Live Event
SANS Virginia Beach 2018	Virginia Beach, VA	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS Chicago 2018	Chicago, IL	Aug 20, 2018 - Aug 25, 2018	Live Event
Mentor Session AW - SEC401	Raleigh, NC	Aug 22, 2018 - Aug 29, 2018	Mentor
SANS San Francisco Summer 2018	San Francisco, CA	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS Tokyo Autumn 2018	Tokyo, Japan	Sep 03, 2018 - Sep 15, 2018	Live Event
SANS Amsterdam September 2018	Amsterdam, Netherlands	Sep 03, 2018 - Sep 08, 2018	Live Event
SANS Tampa-Clearwater 2018	Tampa, FL	Sep 04, 2018 - Sep 09, 2018	Live Event
SANS Baltimore Fall 2018	Baltimore, MD	Sep 08, 2018 - Sep 15, 2018	Live Event
SANS vLive - SEC401: Security Essentials Bootcamp Style	SEC401 - 201809,	Sep 11, 2018 - Oct 18, 2018	vLive
SANS Munich September 2018	Munich, Germany	Sep 16, 2018 - Sep 22, 2018	Live Event