Global Information Assurance Certification Paper

Cynthia Cripe
December 15, 2003

Sudo as an Alternative to Root

Abstract

The UNIX operating system must be secured just like any other operating system. Since the operating system is the most important program on a UNIX server, the root account must be guarded to ensure availability. The root account must also be guarded to ensure data integrity. Since users often need to run commands as the root user, alternatives to disclosing the root password are needed. If the user cannot be added to the group which allows the needed access and an ACL cannot be implemented, sudo is an easy, secure, and inexpensive option.

This paper will explain UNIX security files and the significance of user accounts, file permissions, and groups. This will be a foundation for an overview of the root account and why the use of this account should be kept to a minimum. Then a detailed explanation of how to implement sudo will conclude the paper.

Body

The anthem of the network security world is “least privileged access,” and for very good reasons. Many issues face the network security professional which is solved through least privileged access. One issue is protecting the confidentiality of customer data. Another issue facing security professionals is protecting program files. Yet another issue is protecting the computer’s operating system files. Least privileged access restricts user access so that they can view or alter only the files they need to in order to do their jobs.

Computer security encompasses several aspects of computer and information management. This paper will cover the UNIX system and its files, users, and data as a foundation to the discussion of sudo as an alternative to root access. UNIX security professionals are concerned with user access, groups and file permissions. Proficient knowledge of access controls allow the UNIX security professional to ensure the integrity of the data housed on UNIX servers. Limiting the use of root access is paramount to assisting UNIX security professionals in accomplishing this directive. However, there are times when users have a legitimate need for root access. Sudo is a security tool that is easy to implement, secure, and inexpensive. It is an excellent alternative to root access. I will discuss root access and why its use should be kept to a minimum. I will also discuss sudo as an alternative to root. I will then demonstrate the ease of sudo implementation. You will see how sudo can solve an enormous UNIX security problem.

The root password of a UNIX system is a coveted possession in the UNIX world. Root access on a UNIX server can make life easier by eliminating keystrokes and knocking down the barriers of file permission restrictions. I

understand why individuals request permanent root access. Work is made easier while using the root account. Before root access is granted; however, the requester needs to understand other aspects of the root user. The requester needs to understand the significance and responsibility involved with possessing the root password. An overview of user accounts, system groups and files sets the stage for an understanding of the root account. It lends understanding to the ease and benefit of using sudo as an alternative to root access. Discourse concerning the implication of possessing the root password will also support the premise that the root password needs very little use.

UNIX system administrators receive numerous requests for permanent root access to UNIX systems. There are good reasons for possessing the root password. It is easier to accomplish many tasks with the root account. But, before permanent root access is granted the requester needs more information about the root account and the implications of working as the root user.

The root user is specifically used to administer the operating system. The importance of an operating system cannot be overstated. When a user uses a computer they are using an operating system. (I will use computer, system, and server interchangeably.) Most people are familiar with Windows, the graphical interface most people see when they turn on their computers. A computer operating system is defined by Webopedia™ as "the most important program that runs on a computer. Every general-purpose computer must have an operating system to run other programs (Webopedia™)." The operating system is a group of files working together to manage processes, manage memory, manage input and output, and manage files. In other words, it manages all activity on the computer. The operating system also acts as an interpreter between the hardware (e.g., keyboard, mouse, printer, monitor, etc.) and software (e.g., DB2, Oracle, Microsoft Word, etc.). You cannot operate a computer without an operating system.

A UNIX operating system consists of users, groups, and files. A description of each component is necessary in order to give the big picture of operating system security. In order to access files on a UNIX system a user must have a user account. Each user account has a unique name (also called a user name), a unique identification number (also called a UID), and a password. The root user (also called the superuser) is the most powerful user on a UNIX system. In Essential System Administration, AEleen Frisch defines the root user as follows: "Unlike some other operating systems, the UNIX superuser has all privileges all the time: access to all files, commands, etc. (7)." The root user is primarily used to administer various tasks at the operating system level.

Sometimes a group of users needs access to the same files. The UNIX system accommodates this need by grouping users through system groups. A UNIX system group is a set of users, all of whom need access to a given set of files. Every user name is a member of at least one group and can be a member

of several groups. For instance, if the users steve, fred, susan, and shelly are employees of the accounts payable department they will need access to accounts payable files. A system administrator is likely to name a group acct_pay and steve, fred, susan, and shelly would be placed in this group. By default, the root user is added to all groups.

On a UNIX system everything is a file. Each file has a set of permissions. The file is also owned by a user name and is associated with a system group. If you listed the attributes of a file you might see a line resembling the following:

-rwxr-xr-- 1 john staff 512 Sept. 9 14:01 johnsfile

The first string of characters (-rwxr-xr—) is the file permissions. The second string of characters (1) is the number of links. The third string of characters (john) is the file owner. The fourth string of characters (staff) is the group associated with this file. The fifth string of characters (512) is the file size. The sixth string of characters (Sept. 9 14:01) is the date and time the file was last altered. And the seventh string of characters (johnsfile) is the file name. UNIX security is primarily concerned with file permissions, the owner, and group because these components control access to files.

The first character in the permissions, the dash (-), signifies this entry is a file. The second through fourth characters in the permissions (rwx) indicate that the owner has read, write and execute permissions. The fifth through seventh characters (r-x) indicate the group has read and execute permissions. And the eighth through tenth characters (r--) indicate that all other users on the system have only read permissions. Therefore the user “john” can do anything he wants to the file. The users in the staff group can look at the file or execute the file, but they cannot change anything inside the file. All other users on the system can look at the file, but they cannot execute or change the file

Now that we understand user accounts, system groups, and files with their associated permissions, owners, and groups, I would like to discuss the considerations for granting permanent root access. Since root has full privileges to the entire file system on a UNIX server, the root user can read, modify, execute or delete any file on the system.

The first dilemma is the security of the operating system. In Essential System Administration, AEleen Frisch states, “Therefore, it is entirely too easy for a superuser to crash the system, destroy important files, and create havoc inadvertently.” (7) In the book Practical UNIX, Steve Moritsugu states, “If you are logged in as root, a simple typographical error could be disastrous for the whole system.” (80) Since the entire computer relies on the operating system to perform its job, it is critical that the operating system files are protected. This means ensuring that the system is functioning so your application doesn’t crash and you have constant access to your files. This can best be accomplished

through limited distribution of root's password. For a non-critical server, a system crash could costs a company the amount of wages paid to the system administrator who recovers the system. For a critical server, the cost could be astronomical. Expenses would include the cost of the system administrator's wages as well as the wages paid to users who cannot perform their jobs. Web servers could cost customer satisfaction would could result in the loss of business. This translates to lost revenue. The final consequences could result in millions of lost dollars. Not only am I concerned with protecting private information, I am also concerned with protecting my job, your job, and the jobs of our fellow employees by complying with these laws and helping to ensure the financial stability of our employer.

The second dilemma is the security of the data on each UNIX system. Businesses now have to protect the public's private information. Two such federal laws are the Health Insurance Portability and Accountability Act of 1996 (HIPAA) and the Gramm-Leach-Bliley Act of 1999 (GLB). HIPAA is described on the United States Department of Health and Human Services website as follows: "The new privacy regulations ensure a national floor of privacy protections for patients by limiting the ways that health plans, pharmacies, hospitals and other covered entities can use patients' personal medical information." (Fact Sheet) The privacy portion of the Gramm-Leach-Bliley Act is described on the United States Federal Reserve website as follows: "Most important, our objective is to devise disclosure requirements and consumer "opt-out" procedures that protect consumer privacy without overwhelmingly burdening financial institutions or consumers (Remarks by Governor Laurence H. Meyer)." Each law spells out stiff penalties for violations of the laws. In this litigious society, lawsuits are likely to ensue. Lawsuits normally translate to thousands, if not millions of dollars in legal fees, court costs, and settlements to the defendants. Disclosure of private information could also result in damage to the company's reputation. This will likely translate to lost business which means lost revenue

I spoke with Glenn Heermance, a 15 year UNIX system administrator. I asked Glenn what problems he has experienced from unintentional or intentional misuse of the root account. Glenn explained that the biggest problem they experience is that users do not know which directory they are in before they make file permission, ownership or group changes. He stated that other problems they have experienced are system shut downs, unauthorized file system creation, and other such administrative tasks that can cause outages or system performance problems. (Heermance)

Glenn advised that these changes often result in two to five hundred lost man hours in investigating system outages. He said that the average problem takes approximately ten to twenty hours to troubleshoot. He said he and his team experience, on average, approximately two of these outages per month. (Heermance)

I asked Glenn about alternatives to root access. He immediately responded, "Consulting." I asked him to explained, and he said that one of his team's primary roles is to consult with application support areas for changes to the UNIX servers. He said that his team will evaluate the changes and make suggestions for improvements. He said that once an agreement is reached on system changes, the deployment team can issue the changes to the servers. He recommended that the application areas have root access on the non-production servers in order to learn how the UNIX environment best interacts with their application. Glenn said that problems on non-production systems can be mitigated with education. (Heermance)

Now that you understand the importance of the files on a UNIX system and the considerations for granting root access, I would like to discuss alternatives to root access. The three alternatives I wish to discuss are system group assignments, access control lists (ACLs) and sudo.

Since each file is associated with a system group, users can be added to the group or groups associated with the files the user needs to manage. If the permissions of the files are too low, not allowing the access needed, the permissions on the files can be changed. This will allow read, write and execute permissions for all of the files that are needed to be managed without using the root account.

Since changing file permissions could stop an application from running, system administrators can also consider access control lists (ACLs). ACLs have to be used with care. If file permissions are changed using the hexadecimal method, the ACL associated with the file is no longer valid. This will cause users who are granted permission to the file via an ACL to lose their access. It will also allow anyone who was disallowed access via the ACL to be granted access. UNIX allows the application of ACLs but the operating system does not provide a way of tracking the ACLs. Therefore, careful documentation of the their existence is necessary.

Since the use of ACLs can be time consuming and troublesome to management, the other alternative is sudo. Sudo (pronounced SUE-DUE) is defined on the sudo website as follows: "Sudo (superuser do) allows a system administrator to give certain users (or groups of users) the ability to run some (or all) commands as root while logging all commands and arguments. Sudo operates on a per-command basis, it is not a replacement for the shell (Miller)." If commands need to run as the root user, sudo rules can be written to grant this access. Sudo is simple to administer. Once the commands are identified, sudo rules can be implemented within several minutes.

If you find sudo is a viable option, information gathering is the next step. Receiving the information by way of via e-mail or a memo is the best approach in order to archive the information indefinitely for future reference. If a system

crashes and you do not have a good system back up, the documentation can be used to recreate the sudoers file. You may also need the business case as proof of need in an audit.

First, it is important to obtain a business case for the access changes. The business case should be a thorough description of the user's needs. The following business case is too vague: "I need to run the rm command in order to perform my job." The business case should be full of specific information like path names, file names, the individuals needing access, on what servers they need sudo access, all of the commands (along with the command path) they need to perform using sudo, any situations where the command needs to run as a different user, and are the rules temporary or permanent. It needs to include specific reasons why the user must have sudo rules such as the file permissions do not allow them to use the command. Sometimes through the business case you will find that the user or users don't need changes to their access.

After you determine that the user(s) need sudo access, collect the names and sign on IDs of the individuals who need the sudo rules. Is it one individual, is it several different individuals with nothing in common, or is it a group of individuals with a common UNIX group who need access? Refer to the business case to verify this information.

The next piece of information you need is a list of servers where the user(s) needs the access. Refer back to the business case. Is the access needed in the testing environment, the development environment, and/or the production environment?

Next, what commands does the user(s) need to run in order to perform their jobs. Again, confirm their needs with the business case. Special care must be exercised when granting certain commands. And some commands should never been granted sudo privileges. I will discuss this later when I demonstrate sudo implementation.

Next, do the commands need to run as a user other than root. Some programs require that the script run as a specific application ID. Sudo is flexible enough to accommodate this need.

Finally, are the rules temporary or permanent? If a user needs to run a command once a month using sudo, it is possible to script the sudo rule so that it appears during the time period needed then script the rule out of the sudo file once the command is executed. Sometimes rules are needed for installation then they are not needed again. Sometimes the rules are needed on a daily basis.

Be certain to ascertain all of this information and writing the sudo rules will be simple. Keep your lines of communication open with the users. You will have

to have the users execute the sudo commands to ensure that sudo fulfills their needs.

Once you have your information in hand it is now time to implement sudo. If sudo is not packaged with your version of Linux or UNIX, go to the Main Sudo Page in order to download the software (Miller). First navigate to the Getting Sudo Via FTP link found on the left side of the webpage (Miller). FTP the sudo software to your server. Before you install the software, navigate to the Documentation Installation Notes (Miller). Read the installation notes completely to ensure a successful installation. Since there are several flavors of UNIX and Linux, I will rely on your expertise to read the instructions and install the software properly. If you plan to use sudo on all servers, I recommend that you make sudo part of your base load.

Once the software is in place, we need to know the parts of sudo. The first part is the actual command itself, sudo. The sudo command will precede the UNIX command the user wishes to execute. The location of the sudo command will vary depending on which flavor of UNIX you are running. The second part is the file that holds the sudo rules. This file is located in the /etc directory and the file name is sudoers. There are several options available for the sudoers file that I will later discuss in detail. The third part is the visudo command. This command should always be used when editing the /etc/sudoers file. On the O'Reilly Network website, Michael Lucas states that "visudo locks the file so only one person can edit the configuration file at a time. It then opens the sudo configuration file in an editor (vi by default, but it respects the $EDITOR environment variable). When you exit the editor, visudo parses the file and confirms that there are no sudo syntax errors. This is not a guarantee that the configuration will do what you want, merely a confirmation that the file is actually a valid. (Lucas)" Using visudo protects the sys admin in two ways. First, visudo locking the sudoers file prevents one sys admin from overwriting another sys admin's changes. If both individuals are in the file at the same time, the last person saving changes will overwrite the first. Second, visudo checks for syntax errors, which helps to minimize sudo errors.

Now that I have given a high level overview of sudo, I will provide a fictitious business case so that I can demonstrate the simplicity of implementing sudo rules. The business case comes from the Human Resources department of a fictitious company and reads as follows:

> The user IDs bob, carol, ted, and alice need the ability to execute /usr/local/bin/hrapp and /sbin/hrscript. They need the ability to run these commands on the following servers: hrsrv1 and hrsrv2. When running hrapp, this application's vendor requires that the application run as the hrappid user. We do not want to give bob, carol, ted, and alice the password for hrappid. bob, carol, ted, and alice also need to execute the /sbin/hrscript when performing daily

tasks. The hrscript is owned by the root user and is associated with the root group. They need root access to execute this script. bob, carol, ted, and alice have to run the hrapp and hrscript in order to perform their jobs. The user ID joseph needs to run the /usr/local/bin/hrprocessing and /sbin/hrscript scripts on server hrsrv1. Part of joseph's job requires that he run this command on a daily basis. Additionally, the UNIX group infosys needs to be allowed to run the shutdown command on the hrsrv1 and hrsrv2 servers in order to bring down the server during the maintenance window. The hrprocessing script and the shutdown command require root access.

A security conscience system administrator does not want to provide the root password to anyone. The business case also states that the Human Resources department does not want to distribute the password for hrappid. The users will be able to log on as themselves and execute hpapp and hpscript using sudo. Likewise, the joseph user will be able to run the hrprocessing script and the infosys UNIX group will be able to run the shutdown command using sudo. No additional passwords need distribution.

Before writing sudo rules, additional consideration must be given to whether the company's computer security policy dictates that users must authenticate each time they execute a command using sudo. For the purposes of this scenario and best security practices dictate that the users always authenticate when using sudo. Now, I will discuss the essential components of the sudoers file and demonstrate the ease of implementing sudo.

Within the sudoers file there are several options to consider while writing sudo rules. The first option I will discuss is setting up users aliases. If a group of users needs the same sudo rules and one or more of the users does not belong to the same UNIX group, the system administrator can group them in the sudoers file under a user alias. A user alias is comparable to a UNIX group. The user alias syntax is case sensitive and should look like the following example:

User_Alias HRUSERS=bob,carol,ted,alice

The next option is grouping servers. If bob, carol, ted, and alice need identical access on the same group of servers, the system administrator can create a host alias. The host alias syntax is case sensitive and should look like the following example:

Host_Alias HRSRVS=hrsrv1,hrsrv2

The next option is grouping commands. If bob, carol, ted, and alice need an identical list of commands, the system administrator can create a command alias. It is important to include the path when creating the list or visudo will return

an error while trying to save the sudoers file. If the commands need to span more than one line, use a backslash (\) in order to tell UNIX that the line continues on the next line. It is best to line up the commands for readability. The command alias is case sensitive and should look like the following example:

```
Cmnd_Alias HRCMNDS=/usr/local/bin/hrprocessing,\
        /sbin/hrscript
```

The final option I will discuss is the creation if a run-as alias for a user ID to run a command. An example of this option is when a user needs to open an application and the vendor of the application requires that the application *run as* a specific user ID. This specific user ID is the *run as* ID. The syntax for *run as* aliases are case sensitive and should look like the following example:

```
Runas_Alias HRAPPID=hrappid
```

Before granting sudo access to a script, a system administrator needs to carefully examine the contents of the script to ensure that no commands are executed within the script that allow the users to break out of the command and receive a shell prompt. If the user breaks out into a shell prompt, they will be the root user, since sudo runs the script as root. The user can then access any file on the server and possibly plan malicious code or drop a root kit. If the script contains a command that has a break command to a shell prompt, the system administrator should follow company policies in order to have the sudo request denied. Granting this type of access will leave a hole in security.

The Sudoers Manual Page is an excellent source of information for additional options allowed in the /etc/sudoers file. It will also give tips on how to enhance and organize your sudoers file.

Now that we know the different parts of the sudoers file, we can write rules. The first step is to switch to the root user. Once you receive the pound sign (#) indicating a root system prompt, type visudo and press the [ENTER] key. Example:

```
$ su
root's password:
# visudo
```

If the /etc/sudoers file was never configured, a blank script will appear with a series of tildes (~) along the left side of the screen. Press the 'A' key to append and begin typing the rules. Below is an example of the sudoers file for the above business case.

```
#User Aliases
User_Alias HRUSERS=bob,carol,ted,alice
#Host Aliases
```

```
Host_Alias HRSRVS=hrsrv1,hrsrv2
#Command Aliases
Cmnd_Alias HRCMNDS=/usr/local/bin/hrprocessing,\
        /sbin/hrscript
#Runas Aliases
Runas_Alias HRAPPID=hrappid
#sudo Rules
HRUSERS  HRSRVS = (hrappid) PASSWD:/usr/local/bin/hrapp
HRUSERS  HRSRVS = PASSWD:/sbin/hrscript
joseph   hrsrv1 = PASSWD: HRCMNDS
%infosys HRSRVS = PASSWD: /usr/sbin/shutdown
```

A detailed explanation of the rules will show the flexibility of sudo. The first line, HRUSERS HRSRVS = (hrappid) PASSWD:/usr/local/bin/hrapp states that the users listed in the HRUSERS user alias, on the servers listed in the HRSRVS host alias, will run as hrappid user ID, requiring user authentication through the use of the user's password, to run the command /usr/local/bin/hrapp. The second line is similar to the first, only the /sbin/hrscript will not run as a different user. It will run as root via sudo. The third line, joseph hrsrv1 = PASSWD: HRCMNDS states that the user joseph, on server hrsrv1, is required to authenticate with joseph's password and has the capability to run the commands listed under the command alias HRCMNDS. The fourth line, %infosys HRSRVS = PASSWD: /usr/sbin/shutdown states that the users in the UNIX group infosys, on the servers listed in HRSRVS, must authenticate with their individual password, in order to execute the shutdown command.

The sudo rules need to be added to each server. Be sure to only apply those rules that are needed for each individual server. Adding additional, unnecessary rules could cause security vulnerabilities by allowing users access to programs and files that they shouldn't have. For example, user ID bob needs to perform sudo /usr/bin/maintscript on servera but not on serverb. If you create the sudo rules for bob on both servers and bob has a user account on both servers, bob can run the script via sudo on both servers. This violates the rule of least privileged access.

As an added feature, sudo logs each time a user invokes a command using sudo. The sudo log is stored in the syslog. The location of syslog varies depending on which flavor of UNIX you are running. Sudo will log each successful and unsuccessful attempt.

Once the sudo rules are in place, the users need to be instructed on how to use their new access. When instructing the users, the system administrator needs to include the fact that sudo should precede the command. The system administrator should also include the fact that the command needs to include the fully qualified path. Often I have received calls stating that the sudo commands don't work. It has usually been one of two problems. Either the user forgot to

include sudo at the beginning of the command line or the user did not include the fully qualified path immediately before the command name. The users should also be advised that they could gain a list of their sudo rules by typing sudo –l at a system prompt.

I searched the Internet for sudo security vulnerabilities. I found vulnerabilities that were two to six years old. The vulnerabilities I found were addressed by Todd Miller and fixed in a subsequent sudo upgrade release. Therefore, the sudo solution to root access is very secure.

The only secure server is the server that is shut off. Since a computer is of little value when it is turned off, security professionals are tasked with identifying vulnerabilities in their systems. Once the vulnerabilities are identified, they must determine the methods and costs of mitigating the vulnerabilities. And finally, they should present their findings to corporate management for a decision on funding. When it comes to reducing the use of the root account, sudo is a freeware option. This is a price hard for any corporate manager to refuse.

In conclusion, I discussed how the operating system (OS) is the most important program on a computer. On a server running the UNIX OS, the root account is very powerful and needs to be guarded closely. I showed how sudo could be used instead of granting root access to everyone who has a legitimate need. I also demonstrated how simple it is to implement sudo. I also explained how to make sudo a secure alternative to root. Then I discussed how there are virtually no known security vulnerabilities with sudo. Lastly, I discussed the low, low price of $0 for the sudo software. Sudo is an excellent alternative to granting permanent root access due to its ease of implementation, security, and low cost.

Works Cited

Fact Sheet: Protecting the Privacy of Patients' Health Information. 14 Apr. 2003. U.S. Department of Health & Human Services. 11 Sep. 2003. <http://www.hhs.gov/news/facts/privacy.html>.

Foremost IT. Operating System. 22 Oct. 2003. <http://www.foremost.co.uk/glossary/operating_system.html>.

Frisch, AEleen. Essential System Administration. 3rd ed. Sebastopol: O'Reilly, 2002.

Heermance, Glenn R. Personal interview. 2003 Sep. 30

Lucas, Michael. O'Reilly Network. "Eliminating Root with Sudo." 22 Oct. 2003. <http://www.onlamp.com/lpt/a/2680>.

Miller, Todd C. Sudo Main Page. 28 Oct. 2003. <http://www.courtesan.com/sudo/sudo.html>.

-- Sudo in a Nutshell. 12 Sep. 2003. <http://www.courtesan.com/sudo/intro.html>.

-- Getting Sudo Via FTP. 28 Oct. 2003. <http://www.courtesan.com/sudo/ftp.html>.

-- Sudo Installation Notes. 28 Oct. 2003. <http://www.courtesan.com/sudo/install.html>.

- Sudoers Manual. 23 Oct. 2003. <http://www.courtesan.com/sudo/man/sudoers.html>.
-

Moritsugu, Steve, David Pitts, and Sanijv Guha. Practical UNIX. Indianapolis: Que, 2000.

Remarks by Governor Laurence H. Meyer. 12 Sep. 2003. Board of Governors of the Federal Reserve System. 12 Sep. 2003. <http://www.federalreserve.gov/boarddocs/speeches/2000/20000203.htm>.

(Webopedia)™. 11 Sep. 2003. <http://www.webopedia.com/TERM/o/operating_system.html>.

Last Updated: July 24th, 2017

Upcoming Training

SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 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
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor