Global Information Assurance Certification Paper

Name: Bobby Decker
Certification: GSEC
Version: 1.4b Option 1
Date Submitted: 8/27/2003

SSH (Secure Shell) Authentication Methods and Security Control

Abstract

SSH also known as Secure Shell is a software package used to provide secure connections within an existing computer network. This document will contain information on the software and protocol history. It will also contain information on problems an environment may face if SSH is not utilized. It will list the types of authentication, and provide detail on host and user authentication. There is a "how to" used to describe public key authentication with an SSH agent. This document will also contain information on providing security control using the server configuration settings.

SSH History

SSH is both a protocol and software package. "Secure Shell's main use is to replace the *telnet, rlogin, rsh,* and *rcp* programs with secure alternatives" (Stein, p.353). A protocol is a set of rules or regulations used when data is sent from one computer to another. To eliminate confusion, when SSH is being referred to as a protocol then the word protocol will be used. If the software package is being referenced then the word protocol will not be used in conjunction with the word SSH.

The SSH protocol will ensure authentication, encryption, and integrity during data transmission. The authentication portion of the protocol will verify that a user is who they say they are. The data will be encrypted until it reaches its intended destination. The integrity of the SSH protocol will detect if any of the data has been changed or altered while being transmitted. SSH has a version one and version two protocol. The SSH product and the SSH version one protocol were developed in 1995 at the Helsinki University of Technology in Finland because of a "password-sniffing attack earlier that year" (Barrett & Silverman, p.10). SSH was available to everyone that wanted it. In 1996, version two of the SSH protocol was created using new algorithms. In 1998, a software package utilizing the SSH version two protocol was released. This new SSH2 software package was only available for educational and non-profit areas. The SSH2 software package has now been implemented, maintained, and sold as a commercial version.

SSH is often used by individuals that are unaware of the software's capabilities. This can be good or bad. It is to the advantage of the individual when they are unaware of the existence or function of the software package. It will prove to be

easier to use and understand. It will function as though the user is unaware of its existence. It may be a disadvantage when users are familiar with the functionality. They may lack the knowledge needed to use the product appropriately. They may also use the product with malicious intent.

SSH uses a client/server relationship. One computer system acting as a client will make a request to another computer system acting as the server. Through this relationship and the use of SSH the request from the client and reply from the server will be encrypted. In fact, any communication or data that travels between the client and server will be encrypted in transit. The data can reside on the client or server in plain text and it will arrive in plain text. The data will only be encrypted while it is being transmitted. This means that it is not necessary for a user to have a working knowledge of encryption or SSH in order to use the product effectively. Data can reside on a system in plain text, be sent encrypted, and arrive to another system in plain text.

Threats to an environment without SSH

If a computer network exists and is functioning without SSH, then the environment will be left exposed and prone to an attack. The installation and configuration of SSH will not eliminate the exposure to an attack. However, the product will provide a significant amount of security that will greatly reduce or eliminate the chance of an attack ever taking place. Without the existence of SSH or a similar product, a network is left wide open and becomes vulnerable. Sensitive information would be transmitted in clear text over network cables. For example, a user name and password could be intercepted on a network cable on its way to a destination system. If this information is intercepted, then the system that it was intended for has been compromised. This example refers back to the information provided earlier as to why SSH was developed back in 1995.

The evaluation and configuration of SSH on a system is the responsibility of the system administrator. In a networked environment, all of the data being transmitted has to be evaluated. The operating systems, the users, the system's physical location in a network, and the importance or confidentiality of the data all have to be taken into consideration. This evaluation or risk assessment will determine the areas that contain sensitive enough information to warrant the need for increased security. The system administrator should also be setting up security policies after a risk analysis has taken place. The security policy may dictate the requirements needed to implement SSH in their environment. One way to administer the SSH software package is to determine all of the levels of security through the risk assessment. If an environment has four levels of security then the administrator can set up four server configuration file templates. Each server in a particular security level will receive the appropriate configuration file template.

Once SSH has been installed and configured on a system, then one step has been added to improve that system's security. The level of security on a particular system can also be configured and manipulated using SSH. Some systems may prove to have a greater need for security, and such systems can be locked down tighter than others based upon the configuration of the SSH server software.

Types of Authentication

The following list of authentication types can be used while configuring SSH to fit the needs of a computer environment. Each type of authentication will state the possible values in italics.

- AllowedAuthentications *password,hostbased,publickey*
- DSAAuthentication *yes no*
- KerberosAuthentication *yes no*
- PasswordAuthentication *yes no*
- PubKeyAuthentication *yes no*
- RequiredAuthentications *password,hostbased,publickey*
- RhostsAuthentication *yes no*
- RhostsPubKeyAuthentication *yes no*
- RhostsRSAAthentication *yes no*
- RSAAuthentication *yes no*

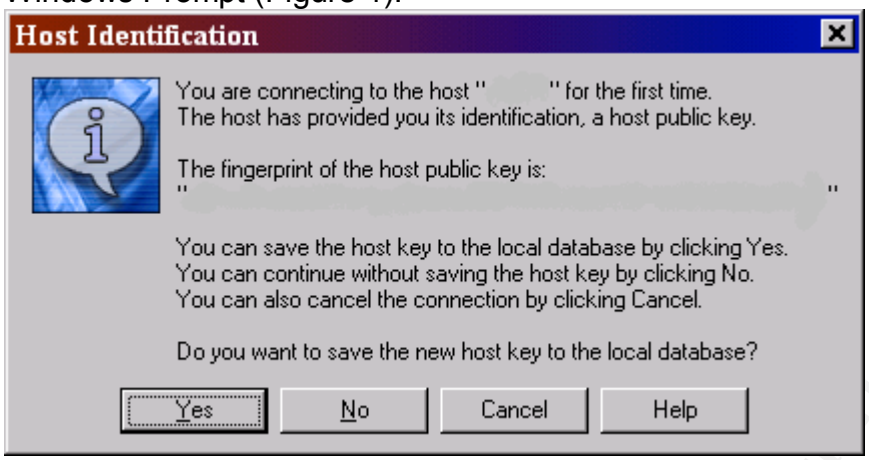
Authentication can be defined as the validation of someone or something's identity. When SSH is used to make a connection to another system there are two authentications that take place. One is host based or server based authentication and the other is user based authentication. These two types of authentication, host and user, can be controlled by both the client and the server. SSH uses its configuration files in order to clearly define all of the authentication possibilities. SSH2 has a client configuration file called ssh2_config and a server configuration file called sshd2_config.

Host Authentication

Host based authentication will validate the server information when a request is made from a client. The client piece of SSH will retain the server information to make a comparison for each subsequent connection. Each server has its own host key that is created when SSH is first installed. This host key will remain unique for each server. Host based authentication helps to prevent man-in-the-middle attacks and it helps ensure that the client is connecting to the same server.

When a user makes an initial connection to a system using SSH the user is prompted.

Windows Prompt (Figure 1):



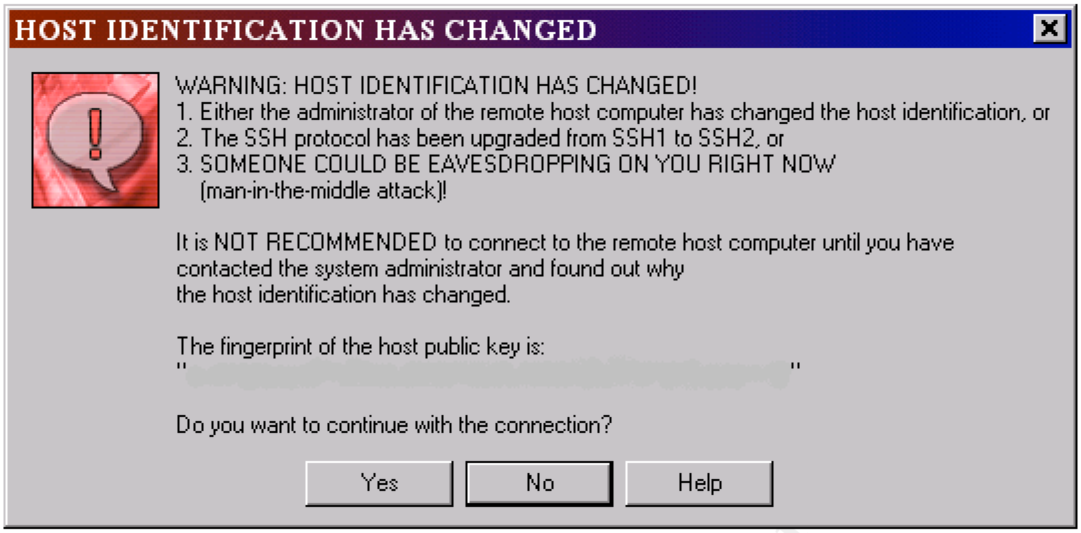
* Screen shot of SSH Inc.

UNIX Prompt (Figure 1.1):

```
user@systemA [/home/user]
$ ssh systemB
Host key not found from database.
Key fingerprint:
xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Are you sure you want to continue connecting (yes/no)? yes
Host key saved to /home/user/.ssh2/hostkeys/key_22_systemB.pub
host key for systemB, accepted by user Mon May 5 2003 12:51:23 -0600
```

This prompt will inform the user that a connection to this particular system has not been made by this client before. The user is then given the choice to continue connecting to the server. If the user continues with the connection to the system, the servers host key will be stored locally on the client machine. After the initial connection, if the user accepts the host key, the host authentication happens behind the scenes. The user will not be prompted for the host authentication again for that server unless the host key of that server does not match up with that server's key that was stored on the client during the initial connection. If the server's host key changes then the user will receive a warning stating that a man-in-the-middle attack might be taking place.

Windows Prompt (Figure 2):



* Screen shot of SSH Inc.

UNIX Prompt (Figure 2.1):

```
user@systemA [/home/user]
$ ssh systemB
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the host key has just been changed.
Please contact your system administrator.
Add correct host key to "/home/user/.ssh2/hostkeys/key_22_systemB.pub"
to get rid of this message.
Received server key's fingerprint:
xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx-xxxxx
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Agent forwarding is disabled to avoid attacks by corrupted servers.
X11 forwarding is disabled to avoid attacks by corrupted servers.
Are you sure you want to continue connecting (yes/no)? yes
Do you want to change the host key on disk (yes/no)? no
```

The user is then asked if they would like to continue connection to the server. If they answer "yes" then they are asked if they would like to replace the servers host key locally with the new key that was detected. If they answer "no" then the session will be closed.

A man-in-the-middle attack occurs when a client attempts to connect to a server and the connection is intercepted by a third system. The third system will masquerade as the destination server in the client/server connection. A true man-in-the-middle attack happens when the following events take place:

- The client will initiate contact with systemA.
- SystemX acting on behalf of systemA intercepts the transmission.
- SystemX manipulates the transmission often with malicious intent.
- SystemX sends the data to systemA.

The other vulnerability that server authentication will help prevent is a connection to the wrong server. If a network is compromised, then data can be redirected to go to the wrong server altogether. In referring to the example above, the client tries to connect to systemA. If the real systemA is taken off the network or if the real systemA is being flooded with network requests then systemX can try to become systemA. With SSH installed the user would be notified about the man-in-the-middle warning, but in this case the connection would be made only to systemX.

It is important to note that the warnings in Figure 2 and Figure 2.1 will not always prove to be a network security breach. These warnings will appear if the host keys of a server have been changed. If a user has administrative privileges on a system then they have the ability to regenerate new server host keys. A system might be experiencing one of the following: it is being renamed, it is being imaged, SSH might be uninstalled and then reinstalled, the host key may be periodically recreated, and the host key might be destroyed. Any one of the following issue or other issues may change a systems host key, thus generating a man-in-the-middle warning for the clients that have the old host key stored locally.

User Authentication

After a client makes a connection to a server and the host authentication takes place, the user has to be verified. The user authentication is generally in the form of a user password or user public key.

If a user password is utilized then the user will be prompted for their password while using the SSH client. The password that is needed to authenticate to the server is not an SSH password. The password is completely independent of the SSH protocol and the SSH software. The password that the user needs to supply is the operating system password. The SSH client provides the means to get to a system with an encrypted name and password, but the user needs to have rights and privileges on that system. Once the user provides a password it is evaluated and access is granted once operating system credentials are verified.

"The name *public key* comes from the fact that you can make the encryption key public without compromising the secrecy of the data or the decryption key" (Suominen). The public key method of authentication is more secure, but also a little more cumbersome to administer. The user will generate a key on a client system. When the key is created the user will be prompted to enter a

passphrase. The passphrase acts like a password. This passphrase can be different than most passwords, in that, the user is able to provide a true phrase that can contain special characters and even spaces. The passphrase is entered twice to avoid error. When this key generation is complete, there will be a public key and a private key. The key will reside in the users default profile on the operating system that the key was created on. In order to use the newly created key for authentication several steps have to take place.

UNIX steps to utilize a user's public key for authentication:

1. The user enters the ssh-keygen command on systemA to create their public and private keys. Once the user provides the passphrase the keys will exist in their own .ssh2 directory.
   ```
   user@systemA [/home/user]
   $ ssh-keygen key_name
   ```
2. The user has to create a file called "identification" on systemA that will contain the name of the private key that was created. This identification file will allow SSH to determine which private key to use for authentication.
   ```
   user@systemA [/home/user/.ssh2]
   $ more identification
   IdKey key_name
   ```
3. The user will place the public key on systemB to use public key authentication. The public key will reside in the user's .ssh2 directory on systemB.
4. The user will create a file on systemB called "authorization" that will contain the name of the public key that was copied over.
   ```
   user@systemB [/home/user/.ssh2]
   $ more authorization
   Key key_name.pub
   ```
5. Once the following steps have been done the user will be able to connect using the passphrase that was created during the ssh-keygen process. Note: The user will only be able to authenticate using the public key if the server is configured to accept a public key connection.
   ```
   user@systemA [/home/user]
   $ ssh systemB
   Passphrase for key "/home/user/.ssh2/key_name":
   ```

In a UNIX environment the users have the option of using an SSH agent to store their keys to make multiple connections using the public key authentication method without providing any password or the passphrase. This agent will hold multiple keys for the user, and the user will have to provide the passphrase only one time to store the key into the agent. "Before your connections can be authenticated without prompts for a pass-phrase you have to use ssh-add to add the necessary keys to memory" (Suominen). Once the user's keys have been added to the agent, any systems that have the above process set up will allow a remote SSH command without providing user credentials. Below are the steps needed to add a user key to an agent.

1. The user has to add the SSH agent into the UNIX shell.
   ```
   user@systemA [/home/user]
   $ exec ssh-agent2 /usr/bin/sh
   ```
2. The user has to add the key into the agent. This process will prompt the user to provide the passphrase for the key.
   ```
   user@systemA [/home/user]
   $ ssh-add2 /home/user/.ssh2/key_name
   Adding identity: /home/user/.ssh2/key_name
   Need passphrase for /home/user/.ssh2/key_name (1024-bit dsa,
   user@systemA).
   Enter passphrase:
   ```
3. The user can use the ssh-add command with a flag, -l for list, to verify that the agent is storing the key that the user just added.
   ```
   user@systemA [/home/user]
   $ ssh-add2 -l
   Listing identities.
   The authorization agent has one key:
   key_name: 1024-bit dsa, user@systemA
   ```
4. After the key has been added into the agent the user will be able to connect to any systems that have the combination of the public key and the "authorization" file. The user will not be prompted for a password or a passphrase because the key and passphrase are being held by the agent.
   ```
   user@systemA [/home/user]
   $ ssh systemB

   user@systemB [/home/user]
   $
   ```

Windows Client steps to utilize a user's public key for authentication:

1. Open up an SSH client.
2. Go to Edit -> Settings.
3. Go to a section within the settings that specify "User Authentication" and click on keys.
4. Click on a button that allows a user to generate a new key.
5. Follow the wizard to create a key and a passphrase and close the settings box.
6. Connect to a system with the password.
7. Once connected to a system, go back into the settings for user keys and click on the button that says upload public key. Accept the default values and click ok.
8. Exit the system.
9. At this point the user will be able to connect to the system and specify "public key" as a type of authentication. The user will be prompted to enter the passphrase that was created in step five.

Remember that the allowed authentication types are controlled by the server. So, if a system does not allow for public key authentication then this process will not work.

Security Control using sshd2_config

"The server drives the authentication by telling the client which authentication methods can be used to continue the exchange at any given time" (Ylonen). The sshd2_config file is used on a server to control all SSH connections that are made to a server. When a client makes a connection to a server using SSH, then the predefined settings in the configuration file will dictate what is allowed and not allowed during that client's connection. When referring to security controls, I am making reference to who can connect to a system and how a user can connect to a system. There are several security controls that can be utilized in the configuration file. Below is a list of some of the keywords that can be used in the configuration file. The keywords listed are provided with a brief description of how it can be used.

- Port- The default value is twenty-two. This number can be set to any available port. If this is set to an abstract number then it will be more difficult to connect to the system making it more secure. When the SSH daemon is running, it will only be listening for a connection on the port that is specified.
- ListenAddress- With this keyword the user can configure SSH to listen to a specific network address. If a system has multiple cards that are connecting to different networks, SSH will only listen to the address of the card that is specified here.
- MaxConnections- This configuration option will dictate how many SSH connections will be allowed into the system. The default is zero which will allow unlimited connections up to the system's available resources.
- LoginGraceTime- This keyword can be used to specify how much time a user has to authenticate into a system successfully. The value represents how many seconds. To disable this feature the user may enter a zero.
- PasswordGuesses- This value will limit the number of failed authentication attempts to a system. This value for SSH will only be used against failed password attempts. Therefore, a user will be able to make an unlimited number of login attempts using the public-key method for authentication.
- PermitEmptyPasswords- If this value is set to no it will ensure that a user will not be able to login to a system with an empty password.
- StrictModes- This option can be configured to look at the permissions and ownership of the user's SSH directories. The files and directories have to be owned by root or the account owner and the write permissions for the world cannot be open.
- IdleTimeOut- This keyword can be used to terminate a connection that has been idle for a specific amount of time. This option can be followed by a zero to be disabled or it can be followed by a number and a letter. The letter has to be the first letter of one of the following: seconds, minutes, hours, and weeks.

- AllowedAuthentications- This option will specify which authentication method will be allowed into the server. These options can include: publickey, password, and hostbased.
- RequiredAuthentications- This option will override the AllowedAuthentications keyword. For example if there are multiple options listed for AllowedAuthentications, but the RequiredAuthentications only lists publickey then the other options cannot be used.
- AllowHosts- This option can be used to allow only the hosts that are listed with this keyword. If this option is used then any hosts that are not explicitly listed with the keyword will not be given the option to authenticate into the server. The values can be listed with a common name or an IP address. Special characters can be used in the configuration file to specify a range of hosts.
- DenyHosts- This keyword will prevent specific hosts or ranges of hosts from being able to authenticate into a server. If this option is used then all of the hosts that are not listed will have the ability to authenticate to the server.
- SilentDeny- This keyword is used in conjunction with the AllowHosts and DenyHosts keywords. If this keyword is set to no which is the default then the message "Sorry, you are not allowed to connect" (Barrett, p.182) will be displayed. It will also be displayed in the log file. If the keyword is set to yes then the user will not see a failed connect message, and it will not be in the log file.
- AllowUsers- If this keyword is used then only the users listed with this option will be able to authenticate into the system. Any other users that are not listed with this keyword will not be able to authenticate into the system. If this option is not used then all users will be allowed to authenticate into the system.
- DenyUsers- This option will deny authentication for the users listed. It will allow all other users to authenticate into the system.
- AllowGroups- This keyword can be used to allow only specific groups into a system. Many of the users with similar roles are part of the same group. Instead of using the AllowUsers option, an administrator will be able to add an entire group. Again if this option is used and users are not part of the specified groups then they will not be able to authenticate into the system.
- DenyGroups- When this keyword is used, it will ensure that any users in the groups listed will not be able to authenticate. Any groups and users of those groups that are not listed with this option will have the ability to authenticate into the server.
- PermitRootLogin- This keyword can be used to control the root account. In order to protect the root account on a server the setting must be no. If the user tries to connect to a server as root they will not be able to authenticate into the system.

If any of the previous options are changed on the server, the SSH daemon has to be restarted for the settings to take effect.

If a network is left exposed then it can be compromised. SSH was developed for this reason. SSH is set up for host based and user based authentication. A system administrator should be responsible for completing a risk assessment. The outcome of the risk assessment should dictate a security policy that will be enforced by the system administrator. The SSH configuration file stored on the server is an ideal way to control SSH connectivity and functionality for security. There are many existing networks that have been set up prior to security being an issue. If SSH or another security product is not used then the network environment will not contain the authentication, encryption, or integrity that is needed.

Works Cited

Acheson, Steve. “The Secure Shell Frequently Asked Questions.” 9 Mar. 2001. URL: http://www.employees.org/~satch/ssh/faq/ssh-faq.html (25 Aug. 2003).

ATN Applications Support Group. “Research Application – SSH Secure Shell.” 18 Dec. 2002. URL: http://www.unc.edu/atn/asg/applications/sshsecureshell/ (19 Aug. 2003).

Barrett, Daniel J., Silverman, Richard E. SSH, the Secure Shell: The Definitive Guide. Sebastopol: O’Reilly & Associates, Inc., 2001.

Garfinkel, Simon. Web Security & Commerce. Cambridge: O’Reilly & Associates, Inc., 1997. 218, 461.

Koenig, Thomas. “Ssh (Secure Shell) FAQ- Frequently asked questions.” 6 June 1997. URL: http://www.uni-karlsruhe.de/~ig25/ssh-faq/ (24 July 2003).

Stein, Lincoln D. Web Security: A Step-By-Step Reference Guide. Reading: Addison Wesley Longman, Inc., 1998. 35, 353.

Suominen, Kimmo. “Getting started with SSH.” 6 Oct. 2002. URL: http://kimmo.suominen.com/ssh/ (25 Aug. 2003).

University of Bristol Information Services. “Using SSH secure shell client on a PC to connect to and transfer files to and from a remote site.” 16 June 2003. URL: http://www.bris.ac.uk/is/selfhelp/documentation/ssh-i1/ssh-i1.htm (24 July 2003).

University of Minnesota. “SSH secure shell.” 9 July 2003. URL: http://www.physics.umn.edu/support/software/ssh.html (25 Aug. 2003).

“Using SSH Secure Shell to Connect to Host Computers [Windows].” June 2003. URL: http://www.itd.umich.edu/itcsdocs/s4304/ (19 Aug. 2003).

Ylonen, Tatu. “SSH Authentication Protocol.” Sept. 2002. URL: http://www.ietf.org/internet-drafts/draft-ietf-secsh-userauth-17.txt (19 Aug. 2003).

Last Updated: July 28th, 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 New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
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 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
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
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
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 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
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event