Global Information Assurance Certification Paper

Thomas Jonson
GSEC Practical Requirements (v.1.3)

Public-key Cryptography: PGP, SSL, and SSH

Abstract
I originally was going to discuss the merits of PGP, SSL, and SSH for this paper. Not long after I started researching, I realized they all had one commonality: public-key cryptography. This paper discusses the basics of how public-key cryptography works, then goes on the discuss PGP, SSL, and SSH. A brief history of each protocol will be discussed, and well as how to use the protocols and how they work.

Introduction
Communicating securely is becoming ever more important as more transactions are taking place over increasingly less secure networks. In the early days of the internet, everyone knew everyone on the net and all were considered trustworthy. Times have changed however, and the need for encrypting and authenticating communications is paramount. Fortunately, protocols have evolved to allow the safe exchange of information. In fact, a new branch of cryptography was created out of this need.

Public-key cryptography allows for secure communications, strong authentication, and message integrity. Depending on what form of communication is to be employed, three protocols stand out: PGP for email and file encryption, SSL for web service encryption, and SSH for remote administration.

Introduction to Public-key Cryptography

There are basically two kinds of cryptography today, private and public key. PGP, SSL, and SSH, all use, to some extent, public-key cryptography. While public-key cryptography has only been around since the 1970s, private-key cryptography has been used for thousands of years. For example, Julius Caesar supposedly used a substitution cipher to send messages to Gaul. Each letter in the alphabet was shifted by three:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

Using this technique, the word “secret” becomes “vhfuhw.”

“George Washington used a more sophisticated cipher in which each word was assigned a number.”[1] This list would be written down in a codebook, and given to each person needing it. When encrypting the message, the word would be looked up and the corresponding number would be written down. To decrypt, the same list would be used to convert the numbers back into words.

[1] Garfinkel, p 36

However, all private-key based ciphers assume both people communicating know the secret key, for instance, how many letters to shift, and that they will keep it secret. What happens if one of them accidentally discloses the number of letters to shift, or has the codebook stolen? Also, given enough messages, a person looking at the encrypted text will be able to decode them, similar to a popular game in many of today's newspapers. Another shortfall of private-key encryption is the secret key has to be exchanged before communicating securely. If two parties want to communicate, they must first meet somewhere to exchange the secret key, which may be difficult in many situations.

Public-key cryptography addresses many of these issues, and others. This time, two keys are created, one secret and one to be distributed openly. These keys are related mathematically to each other so that a message encoded with one can only be decoded by the other, and vice-versa.

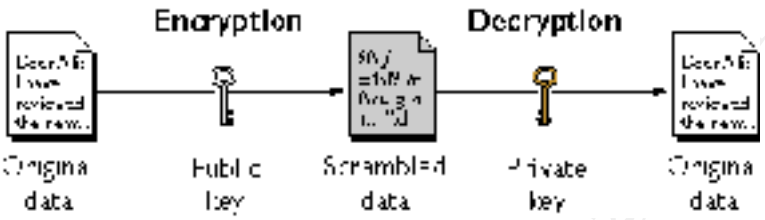

Figure 1 Public-key encryption [2]

For one person to send another a message, they get a copy of their public key, possibly posted on an internet key server, or web site, and use that key to send the message. Anyone can use the public key to send a message, which only the person with the secret key can decrypt.

Unfortunately, this comes at a cost, as public-key encryption is much slower, by orders of magnitude, than private-key encryption. A popular way to circumvent this is to create a random private-key, used only for one transaction, and send that key, encrypted using the slower public-key encryption. The recipient can then decode the session key, and use that speedier private-key to decode the rest of the conversation.

PGP – Pretty Good Privacy

History

Pretty Good Privacy was written by Phil Zimmermann during the late 1980s and was first released in 1994. At the time, good cryptography was really only available to governments and large corporations. Zimmermann was interested in making good cryptography available to everyone. Shortly after the Senate's 1991 anti-crime bill, S.266, which dealt with government wiretapping, Zimmerman thought the time was right to introduce his project. Version 1.0 was released in 1991, as a command line based interface to encrypt email and files. It used the patented Rivest-Shamir-Adleman (RSA)

[2] Netscape, Introduction to Public-key Cryptography

public-key cryptosystem and Zimmermann's Bass-O-Matic algorithm for conventional encryption.

Unfortunately, Zimmermann and RSA Data Security, Inc. hadn't worked out the details of allowing PGP to legally use the RSA cryptosystem, in fact PGP was in direct competition with RSA's MailSafe product, which aimed to also provide secure email. Nevertheless, while the arguing was going on, a following was beginning to appreciate and use PGP.

Version 2.0 was released in 1992. Two important changes in this version was the replacement of Zimmermann's symmetric key Bass-O-Matic algorithm, which had some serious failings, with the International Data Encryption Algorithm (IDEA), and support for multilingual prompts. Looking to further spread the popularity of his program, Zimmermann worked with ViaCrypt, which had a license to use the RSA algorithm, to distribute PGP commercially. With the proper licensing worked out, and commercial support available, it was hoped that PGP would better appeal to companies, and people willing to pay.

The U.S. Government opened an investigation in 1993 into PGP and Zimmermann. Some of the concerns of the government were whether Zimmermann had inappropriately gotten code for PGP, and if Zimmermann had violated any ITAR restrictions, which prevents certain grade cryptosystems from being exported out of the U.S. Fortunately for PGP, the case was dropped, and PGP's popularity continued to spread, quite possibly as a result of all the attention of the investigation.

In 1994, a group at the Massachusetts Institute of Technology, working with Zimmermann, released version 2.6 of PGP using the RSAREF 2.0 cryptographic toolkit. This toolkit specifically allowed non-commercial use of the RSA algorithm. This finally got PGP freeware out of the patent issues.

In December of 1998, McAfee Associates, which later became Network Associates Inc. (NAI), purchased PGP from ViaCrypt. About this time as well, an international freeware version became popular, based on the source code legally exported from the US in the form of a book. The book was carefully scanned and OCR'd to compile the program.

In March 2002, NAI officially put PGP on ice, halting development and sale of PGP. Fortunately, several freeware versions still exist, and more importantly is the formation of the OpenPGP alliance, which "works to facilitate technical interoperability and marketing synergy between OpenPGP implementations."[3]

How it Works

PGP is a great tool to encrypt and/or sign email, files, disks, and also to securely delete these. The original and most widely available versions are command line based.

[3] OpenPGP Alliance

However, several GUI interfaces exist including the commercial product from NAI, PGPi, and several front-end interfaces for GnuPG.

The first thing to do after installing the product is to generate the key pair for the user. This will create both the public and private keys. The private key will be protected with a pass phrase, in case it falls into the hands of the wrong person. However, it is still important to keep the private key protected. If the private key is obtained, a password cracking program can be run against it, and given enough time, the key may be recovered. If possible, the secret key should be kept off of networked file shares, or shared computers.

The public key can then be shared with friends, posted on a web site, or published on a key server. The trick is getting an unmodified copy of the public key to everyone who wants to use it. One possible attack against PGP is if the public key is compromised through a 'man in the middle' attack. The man in the middle could intercept the public key, and replace it with one they create. When someone sends a message to the person, thinking they have the correct public key, the man in the middle can decrypt it, read or modify it, and then pass it on to the recipient, without anyone the wiser. It is therefore good practice to make sure the correct pubic key is obtained for someone before sending him or her a message.

After key generation, PGP can be used to encrypt emails and files. To encrypt an email to someone, PGP needs to have a copy of their public key. Using this key, the message is encrypted so that only they can decrypt it. The message can then safely be sent to the recipient. To encrypt files, the public key of the user encrypting the files would be used. Actually any key can be used, so long as the corresponding private key is used to decrypt it. Usually the user would encrypt it to themselves, so they can use their own private key to decode it.

Another important feature of PGP is the ability to sign a message or file. The person signing the message uses their private key to assure anyone reading the file that it hasn't been modified. This happens in two steps. The first step is to use a message digest function to translate the long message to a certain length string. PGP uses a 128-bit number produced from the MD5 digest function. There is almost statistical certainty than no message will produce the same 128-bit string, and it is a property of the function that the same message will always produce the same string. To see MD5 in action, consider the following:

MD5(message!)= cab2dad1afae064f18e7da4d381f514c
MD5(message.)= 6e908b553eefb7f388590ab76b6e077e

Notice how just by changing one character dramatically changes the 128-bit string. This 128-bit string is then encrypted with the signer's private key. The signer's public key can then be used to decrypt the sting, and compared with a digest of the same message, using the same function.

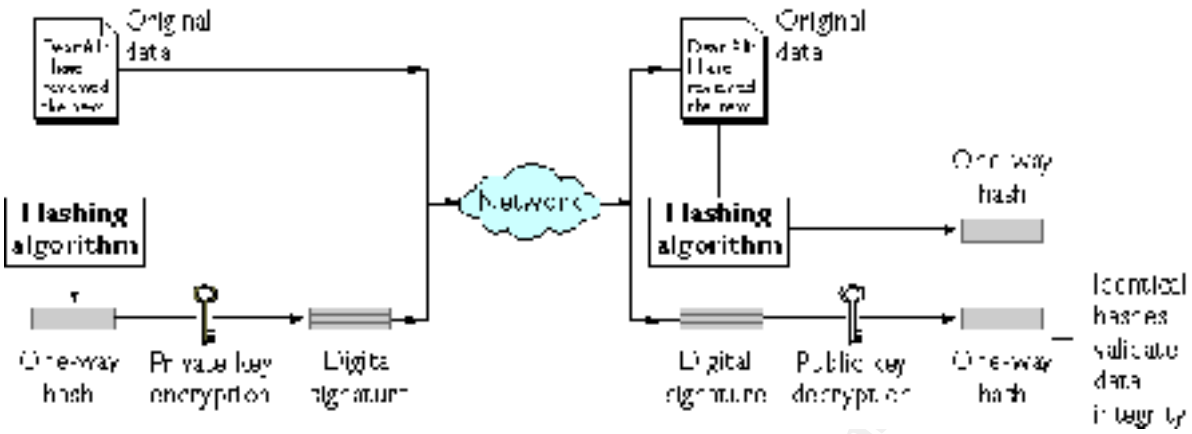

Figure 2 Using a digital signature to validate data integrity[4]

E-mail messages can now not only be encrypted, but also signed, assuring that the information is received exactly how the sender intended. Signing is also often used without encryption, i.e. postings to news groups. The intent is to let everyone read the message, but make sure no one alters the original content.

SSL – Secure Sockets Layer

SSL is a protocol designed to bring encryption and authentication to TCP/IP communication. "The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery."[5]

History
"Originally developed by Netscape, SSL has been universally accepted on the World Wide Web for authenticated and encrypted communication between clients and servers."[6] Today it is used by browsers, and is usually what people identify when they think of web security.

Initially designed in July 1994, SSL started taking off when version 2.0 was released later that year in December. Independent applications and toolkits made the protocol popular in many products beside just Netscape Communicator. SSLv3.0 was released in November 1995 offering performance improvements, more cipher support, and more power for the server to select the level of encryption being used. Also made available was the possibility to use separate authentication and encryption keys.

At that time, due to governmental restrictions, there were two basic strengths of the encryption, a 40 bit key length for international use, and a 128 bit key for use in the United States. Generally, the larger the key length, the more secure the encryption. "128-bits is about 309 septillion times (309,485,000,000,000,000,000,000,000) larger than 40-

[4] Netscape, Introduction to Public-key Cryptography
[5] Freier, et al., p 1
[6] Netscape, Introduction to SSL

bits."[7] The restriction limited the usefulness of SSL in international transactions, as a 40 bit is not considered safe enough for banking and monetary transactions. Fortunately, in January 2000, many of the restrictions were lifted, except to U.S. embargoed destinations.

A newer protocol, Transport Layer Protocol (TLS), stands to carry on the security features that SSL started. Defined in Request for Comments (RFC) 2246, TLS v1.0 goals are the same: cryptographic security, while also improving interoperability and extensibility to allow greater acceptance and use in a wide variety of applications.

How it Works

SSL is designed to provide authentication, encryption, and data integrity as transparently as possible. Authentication assures that the server (and optionally the client) is really who they say they are. For example, before submitting a credit card number to Amazon.com, some assurance must be given that the site really is Amazon.com and not an impersonator waiting to steal credit card numbers.

Encryption is the process of preventing eavesdroppers from listening to the conversation between Amazon.com and the buyer. If the eavesdropper cannot make sense of the transaction, they cannot steal the credit card number in transit.

Lastly, data integrity assures the transition has not been tampered with. This would prevent the books ordered from being changed to the hacker's favorite books, and also prevent the replay of the transition. If the transaction was replayed a hundred times, for instance, the buyer would be quite surprised to receive a hundred copies of each book, and a bill one hundred times larger.

Here is a short summary of how it works:

1. A browser requests a secure page (usually https://).
2. The web server sends its public key with its certificate.
3. The browser checks that the certificate was issued by a trusted party (usually a trusted root CA), that the certificate is still valid and that the certificate is related to the site contacted.
4. The browser then use the public key, to encrypt a random symmetric encryption key and sends it to the server with the encrypted URL required and other encrypted http data.
5. The web server decrypts the symmetric encryption key using its private key, uses the symmetric key to decrypt the URL and http data.
6. The web server sends back the requested html document and http data encrypted with the symmetric key.
7. The browser decrypts the http data and html document using the symmetric key and displays the information.[8]

[7] RSA
[8] Martin

A certificate is like a digital ID. Most e-commerce web sites get a certificate from a Certificate Authority, like Verisign or Twhate. It is the certificate authority's responsibility to verify the ID of the requestor, and to assign them a public and private key pair. The user than can then decide if they want to trust the new web site's certificate based on the authority that has issued it. If they do trust it, they can then use the public key contained in the certificate to establish a secure connection with the server. Optionally, if the requestor has obtained a certificate, the server can then verify the requestor is who they are.

SSL uses the server's public key to encrypt the session key from the client. The server can use its private key to decrypt the session key, and then encrypt the rest of the traffic with the faster session key algorithm.

A popular certificate specification is the X.509 v3 specification. This defines exactly what is in a certificate. Information such as the certificate owner's name, the valid dates for the certificate, the certificate owner's public key, and the certificate authority are contained within the certificate. The date range is used to set expiration on the certificate, so after a given time period the certificate is no longer valid. Below is a sample certificate:

```
Certificate:
    Data:
        Version: v3 (0x2)
        Serial Number: 3 (0x3)
        Signature Algorithm: PKCS #1 MD5 With RSA Encryption
        Issuer: OU=Ace Certificate Authority, O=Ace Industry, C=US
        Validity:
            Not Before: Fri Oct 17 18:36:25 1997
            Not  After: Sun Oct 17 18:36:25 1999
        Subject: CN=Jane Doe, OU=Finance, O=Ace Industry, C=US
        Subject Public Key Info:
            Algorithm: PKCS #1 RSA Encryption
            Public Key:
                Modulus:
                    00:ca:fa:79:98:8f:19:f8:d7:de:e4:49:80:48:e6:2a:2a:86:
                    ed:27:40:4d:86:b3:05:c0:01:bb:50:15:c9:de:dc:85:19:22:
                    43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:a1:00:
                    98:ce:7f:47:50:2c:93:36:7c:01:6e:cb:89:06:41:72:b5:e9:
                    73:49:38:76:ef:b6:8f:ac:49:bb:63:0f:9b:ff:16:2a:e3:0e:
                    9d:3b:af:ce:9a:3e:48:65:de:96:61:d5:0a:11:2a:a2:80:b0:
                    7d:d8:99:cb:0c:99:34:c9:ab:25:06:a8:31:ad:8c:4b:aa:54:
                    91:f4:15
                Public Exponent: 65537 (0x10001)
        Extensions:
            Identifier: Certificate Type
                Critical: no
                Certified Usage:
                    SSL Client
            Identifier: Authority Key Identifier
                Critical: no
                Key Identifier:
                    f2:f2:06:59:90:18:47:51:f5:89:33:5a:31:7a:e6:5c:fb:36:
                    26:c9
    Signature:
        Algorithm: PKCS #1 MD5 With RSA Encryption
        Signature:
            6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c:01:69:8e:54:65:fc:06:
            30:43:34:d1:63:1f:06:7d:c3:40:a8:2a:82:c1:a4:83:2a:fb:2e:8f:fb:
            f0:6d:ff:75:a3:78:f7:52:47:46:62:97:1d:d9:c6:11:0a:02:a2:e0:cc:
            2a:75:6c:8b:b6:9b:87:00:7d:7c:84:76:79:ba:f8:b4:d2:62:58:c3:c5:
            b6:c1:43:ac:63:44:42:fd:af:c8:0f:2f:38:85:6d:d6:59:e8:41:42:a5:
```

```
4a:e5:26:38:ff:32:78:a1:38:f1:ed:dc:0d:31:d1:b0:6d:67:e9:46:a8:
 dd:c4
```

Figure 3 Sample X.509 v3 Certificate[9]

SSL can also be used to send encrypted email, similar to PGP, using the protocol known as Secure Multipurpose Internet Mail Extension (S/MIME). Many of the same functions in PGP can be applied using S/MIME, such as encryption and signing, however S/MIME requires the use of certificates.

SSL can also be used to sign objects. For example, a company can use their certificate to sign a software package, assuring end users that the software hasn't been tampered with during transit.

Vulnerabilities

One of the main vulnerabilities with certificate based authentication and encryption, as used by SSL, is the issue of trusting the Certificate Authority. As long as the Certificate Authority is trusted to assign new certificates, the user must completely trust the authority. If the issuer is hacked, or lapses in verifying the identification its applicants, 'valid' certificates could be issued for unscrupulous purposes. Also, not everyone can afford to buy certificates from a Certificate Authority, although many CA's sell different priced certificates, mainly depending on the level of trust and encryption allowed, users often look to implement their own Public Key Infrastructure (PKI). However, a good PKI is oftentimes difficult to correctly implement, and many vulnerabilities may be exposed if the PKI is not carefully managed.

Another security feature with SSL is the ability to revoke a certificate. If a certificate is compromised, the owner of the certificate can invalidate it by issuing an alert for everyone to stop using that certificate. The owner would then be issued a new certificate to use. However, many implementations fail to easily check for these revocations. It is up to the user to manually check for these, which often rarely gets done.

SSH – Secure Shell

Secure shell (SSH) was developed as a secure replacement for the Unix 'r utilities': rsh, rcp, rlogin, and telnet, to allow secure remote administration. One of the best features with SSH, is after its initial setup, is that it operates virtually transparent to the users.

History

Tatu Ylonen, a researcher at the Helsinki University of Technology in Finland, developed SSH1 in 1995 after his university had been the victim of a password sniffing attack. In July of that same year, Ylonen released the source code to the public. "By the end of the year, an estimated 20,000 users in 50 countries had adopted SSH1."[10] Soon after, Ylonen founded SSH Communications Security, Ltd. to help deal with the flood of support

[9] Netscape, Introduction to Public-key Cryptography
[10] Barrett & Silverman, p 11

questions he was receiving, and to commercialize the product. He also began working with the Internet Engineering Task Force (IETF) to document the SSH-1 protocol.

The SSH-2 protocol was developed to fix a number of issues with the original protocol. However, SSH-2 couldn't be backwards compatible, and due to restrictive licensing, the protocol was slow to spread. Fortunately, as the licensing restrictions eased, and the protocol appeared in a broad range of products, such as Linux and FreeBSD, it has quickly grown in prominence. Also important to the development of SSH-2 was the creation of OpenSSH, which implemented both SSH-1 and SSH-2 under the OpenBSD style licensing.

How it Works

When using the telnet protocol to connect to a remote computer, the user is prompted for their username and password. This information is transmitted in plain text back to the server, and if the credentials match, the user is presented with a prompt. All further communication is sent plain text until the connection ends. All this information is susceptible to eavesdropping through the use of a network sniffer. While not all people necessarily care if their session information is listened to, most people would object to having their password sniffed. SSH solves this problem by using public-key encryption to encrypt not only the data, but also the user name and password.

In addition, SSH can use the principle of certificates to authenticate a user based not only on their password, but also on the key they have on their computer. In fact SSH can even use PGP keys, S/Keys, Kerberos (SSH-1 only), and host based authentication. For example, after creating a public and private key pair, the public key can then be loaded on the server. When attempting to logon to the server the private key will be used in addition to a password to verify a user. This is better that editing the .rhosts file in that this can not be subverted by DNS spoofing. SSH can also be used to tunnel other applications, like Xwindows and ftp, to secure almost any TCP/IP based communications.

SSH-1 establishes a secure connection by the following:

1. The client contacts the server.
2. The client and server disclose the SSH protocol versions they support.
3. The client and server switch to a packet-based protocol.
4. The server identifies itself to the client and provides session parameters.
5. The client sends the server a secret (session) key.
6. Both sides turn on encryption and complete server authentication.
7. The secure connection is established.[11]

The first time connecting to a new host the user is prompted with a message indicating the user has never connected to the server. If the user has already visited the server, the client's computer will have a copy of the server's host key. Once assured of the server's

[11] Barrett & Silverman, p 53

identity, this helps prevent man in the middle attacks, as the user will know whenever it is not talking to the server it thinks.

The client then creates a random session key. It double encrypts this key with the server's host key and the server key. The server key changes every hour to prevent an attacker from collecting too much data from one key, which can assist in cracking a key. The client and server then communicate using the same session key, known only to them.

SSH-2 on the other hand uses a more secure form of session establishment. Using the diffie-hellamn-group1-sha1 method for key exchange, each side contributes to the creation of the session key. This prevents the one side from choosing a specific key to aid in hacking the other's key. Another feature with SSH-2 is the ability for either side to request a re-keying of the session. In this way, a single session could use multiple session keys, further reducing the chance of the key being cracked. SSH-2 also provides a strong means of integrity checking using Message Authentication Code algorithm (MAC).

While not part of the SHH-2 protocol, most implementations of SSH include two other utilities to securely copy files: scp and sftp. Scp is a secure implementation of the Unix cp, and sftp, is a secure ftp agent. These two utilities interact with the user on one end, and call the SSH program on the other. Thus, these utilities make use of all the SSH keys, authentication, and encryption features.

SSH can also be used to secure almost any application using port forwarding. After initial setup, the SSH connection process is virtually transparent to the application. For example, suppose IMAP (port 143), a popular email protocol, is to be secured using SSH. The local host can set up SSH to forward all connections to the localhost on port 2001, to the server's port 143. The email client would then be configured to use the localhost port 2001 as the server. When SSH, listening on port 2001, detects incoming traffic, it encrypts it and sends it over the secure tunnel it created with the server, and instructs the server to forward the data, once unencrypted, to the server's port 143. Although a little bit of work is required to set up the SSH connection, and configure the email client, but the communication now has all the security benefits of SSH, and operates transparently.

Bibliography

1. Garfinkel, Simson. PGP: Pretty Good Privacy Cambridge:O'Reilly & Associates, Inc., 1995. 36

2,4,9. Netscape Communications Corporation. "Introduction to Public-Key Cryptography." 09 Oct 1998. URL: http://developer.netscape.com/docs/manuals/security/pkin/contents.htm (25 Mar. 2002).

3. OpenPGP Alliance. "Welcome to The OpenPGP Alliance." URL: http://www.openpgp.org (03 Apr. 2002)

5. Freier, Alan O., Karlton, Phillip, and Kocher, Paul C. "The SSL Protocol Version 3.0." 18 Nov. 1996. URL: http://www.netscape.com/eng/ssl3/draft302.txt (25 Mar. 2002).

6. Netscape Communications Corporation. "Introduction to SSL." 09 Oct. 1998. URL: http://developer.netscape.com/docs/manuals/security/sslin/contents.htm (25 Mar. 2002).

7. RSA Security Inc. "SSL Basics for Internet Users." URL: http://www.rsasecurity.com/standards/ssl/basics.html (25 Mar. 2002).

8. Martin, Franck. "What is SSL and what are certificates." SSL Certificates HOWTO. 18 Nov. 2001. URL: http://www.linuxdoc.org/HOWTO/SSL-Certificates-HOWTO/x46.html (25 Mar. 2002).

10,11. Barrett, Daniel J., and Silverman, Richard E. SSH The Secure Shell: The Definitive Guide Cambridge:O'Reilly & Associates, Inc., Feb. 2001. 11

Last Updated: March 24th, 2017

Upcoming Training

SANS Pen Test Austin 2017	Austin, TX	Mar 27, 2017 - Apr 01, 2017	Live Event
Mentor Session - SEC401	Milwaukee, WI	Mar 29, 2017 - May 31, 2017	Mentor
Mentor Session AW - SEC401	Grand Rapids, MI	Apr 07, 2017 - May 19, 2017	Mentor
Mentor Session - SEC401	Hollywood, CA	Apr 07, 2017 - May 05, 2017	Mentor
SANS vLive - SEC401: Security Essentials Bootcamp Style	SEC401 - 201704,	Apr 11, 2017 - May 18, 2017	vLive
Community SANS Cleveland SEC401	Cleveland, OH	Apr 17, 2017 - Apr 22, 2017	Community SANS
Community SANS Virginia Beach SEC401*	Virginia Beach, VA	Apr 24, 2017 - Apr 29, 2017	Community SANS
SANS Baltimore Spring 2017	Baltimore, MD	Apr 24, 2017 - Apr 29, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Apr 26, 2017 - Jun 07, 2017	Mentor
Community SANS Salt Lake City SEC401	Salt Lake City, UT	May 01, 2017 - May 06, 2017	Community SANS
SANS Riyadh 2017	Riyadh, Saudi Arabia	May 06, 2017 - May 11, 2017	Live Event
Community SANS Las Vegas SEC401	Las Vegas, NV	May 08, 2017 - May 13, 2017	Community SANS
SANS Security West 2017	San Diego, CA	May 09, 2017 - May 18, 2017	Live Event
Community SANS Columbia SEC401	Columbia, MD	May 15, 2017 - May 20, 2017	Community SANS
Community SANS Baton Rouge SEC401	Baton Rouge, LA	May 15, 2017 - May 20, 2017	Community SANS
SANS Northern Virginia - Reston 2017	Reston, VA	May 21, 2017 - May 26, 2017	Live Event
SANS London May 2017	London, United Kingdom	May 22, 2017 - May 27, 2017	Live Event
SANS Melbourne 2017	Melbourne, Australia	May 22, 2017 - May 27, 2017	Live Event
SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Ottawa SEC401	Ottawa, ON	Jun 05, 2017 - Jun 10, 2017	Community SANS
Community SANS Boise SEC401	Boise, ID	Jun 05, 2017 - Jun 10, 2017	Community SANS
Community SANS Portland SEC401	Portland, OR	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event