Global Information Assurance Certification Paper

Small Scale PKI

Mark A Ryken

SANS Security Essentials GSEC
Practical Assignment Version 1.3
14 January 2002

Abstract

Although many may not see the need for public-key infrastructure (PKI), this paper shows not only the need for PKI but also why organizations are not currently using it. Furthermore, a discussion of the system requirements and an implementation strategy will be presented for a small-scale deployment between two entities. Finally, this paper will specifically show how this PKI deployment can be implemented using the current beta release of Java 1.4 from Sun Microsystems.

Introduction

Banks want an online banking system that their customers are confident in. In December, the Information Technology Association of America (ITAA) and Tumbleweed Communications conducted a survey that asked Americans "how worried are you that the personal information you give out over the Internet could be stolen or used for malicious purposes?"[1] Of those asked 33 percent responded they were "very worried" and another 41 percent responded they were "somewhat worried".[2] Security of their personal information is clearly a concern for the American public; however, most citizens are not as aware that "secure" sites may not truly be secure. Donna Fenn describes a case study of how the Internet-based company UltiMutt was hacked. They used what they thought was a secure shopping-cart software but this had been hacked and they did not receive notification of the patch for their system. This resulted in several of their customers' credit card information being stolen and fraudulent charges being made.[3] This example shows some of the security holes that may exist in current systems that are believed to be secure. As public awareness of these holes increases, the likelihood of the public demanding truly secure systems will increase, especially in the realm of commercial banking.

If one goes to a commercial bank's website and desires to complete an online transaction, how does that person know that they are actually connecting to their desired bank? How does that person know their financial information is secure? The bank may boast of its 128bit SSL encryption; however, how does the bank know who the user truly is? Instances have occurred where a rouge individual obtained certificates[4] from Verisign a trusted certificate authority stating that they were Microsoft Corporation when in fact they were not.[5] In this case the certificates were used to sign executable content.[6] It took almost two months for this to be realized and for the

[1] Burton, 2002.
[2] Ibid.
[3] Fenn, p. 134-139.
[4] RSA Laboratories - 4.1.3.10 What are certificates?
[5] Verisign Inc., 2002.
[6] RSA Laboratories – 2.2.2 What is a digital signature and what is authentication?"

certificates to be revoked, a process in which the certificates are expired or nullified. If these certificates had been the type used in SSL encryption, this rouge individual could have used them to set up a website spoofing or claiming to be Microsoft and through that collect confidential information, financial or otherwise, for that entire time.

More and more companies are beginning to see that the simple password approach to securing an account may no longer be sufficient. Rose Rovelto writes, "A simple password is the easiest and cheapest security method to install. But will it be good enough? Probably. But when Security Software Technologies provides L0phtcrack (http://www.securitysoftwaretech.com/l0phtcrack/), a program that will crack 90% of passwords in less than 48 hours, it makes me wonder how much longer it will be 'good enough.'"[7] This quote is from an accounting magazine and shows that even non-technical persons are recognizing the need for greater security measures. Furthermore, the Oxford Analytica discusses Internet advances and states, "the two most prevalent concerns about computer security … are computer viruses and the risk to privacy involved in online transactions." They further state "that privacy will be improved by the imminent commercial availability of new encryption algorithms."[8] In the case of e-commerce or online banking it is clear to see that SSL is not the final answer. Many would say that some form of encryption is the answer, public-key infrastructure is one such technique, but what exactly is PKI? RSA Laboratories describes a system that in one extreme is simply a web of trust based on public-key certificates, which does not involve encryption. The other extreme RSA Laboratories describes is an entire system of encryption and digital signatures provided at the application level.[9] The remainder of this paper describes a way in which to implement on a small scale, a middle ground public-key infrastructure between two parties who have a pre-existing relationship – a bank and its customers.

The task of implementing a public-key infrastructure can be daunting. Just some of the types of sites it would include are e-commerce, access to personal federal information and online banking. With this in mind, one begins to imagine all the other requirements needed to make it all possible. These include the setting of standards, certificate authorities (CA's), registration authorities (RA's) and other company specific standards. One must also look at the personnel issues – who implements, designs, oversees, decides, maintains, and who approves? At first glance these issues can seem overwhelming, but this paper shows how to begin breaking PKI down into simple and much more manageable steps for a bank to customer connection.

The Bank

In almost all cases there exists some sort of personal contact between a bank and its customers. It may only be when that customer first opens their account, but that contact exists. Also, further non-Internet based contact may be requested by the bank, such as by telephone or regular mail. If a customer calls the bank and has questions about his

[7] Rovelto, p. 37.
[8] Oxford Analytica, 2001.
[9] RSA Laboratories – 4.1.3.1 What is PKI?

account, there are multiple ways in which they verify his identity. Such techniques include the social security number, the amount of the last deposit, the mailing address or mother's maiden name. In the case of a company banking online, it is this same sort of verification that is needed, but this is not as easy as it may sound.

The Goals[10]

When one does his banking, whether in person at the bank, over the phone or in person, there are really four main goals that must be achieved besides the successful execution of the desired transaction.

The first goal is confidentiality. Customers do not want others watching their transactions and getting account numbers, balances and account activity information. This is pretty straightforward in person or even over the phone customers trust that there is no one tapping their phone line listening. But online this becomes much more difficult. Customers cannot see who may be looking and do not have a direct line when completing online transactions and therein lies the difficulty. Due to the nature of the Internet, banks and their customers do not know exactly where their exchanged information goes as it is transmitted back and forth between the customer and bank.

The second main goal is data integrity. A customer at the bank does not ask the person in front of him to give his withdrawal slip to the teller. If he did such and act, that person could add a zero to that $100.00 value and when the teller gives them the money they could give the original customer $100.00 and keep $900.00 for themselves. Banking by phone does not suffer from this, assuming the number called is actually the bank. However, banking online has this as a difficulty too as someone could change the numbers of the customer's transactions.

The third goal is authentication. At the bank this may be done just by the personal contact and possession of the proper account number. However, some cases of large withdrawals may result in a request for picture identification or a thumbprint. Phone transactions accomplish this goal usually with the customer inputting the proper pin number. Online this is most often done with username and password. The limitations of simple passwords as the main security technique has already been shown in this paper.

The fourth and final goal is non-repudiation or proving that a transaction took place. A few years ago, a small business owner (also the father-in-law of the author of this article), dropped off a large check he had received in the night deposit box at the bank. When he did not see the funds appear in his business account he called the bank. The bank said, "What check?" He had no proof that he had dropped off a check for deposit. He now always uses the ATM or goes inside where he gets confirmation of his deposit in the form of a receipt with a transaction number. How can this be done online?

[10] Cole, 5-10 November 2001.

The Problem

The final goal above ended with the question, how can this be done? One really needs to ask that question for all four of the goals. A lot of banks today would like people to think that since they use 128bit SSL encryption, that the connection is secure. However, that is not the case. If the customer is connected to the bank just as he believes he is, 128bit SSL encryption will cover goals one and two. While it is encrypting the data both ways it can neither be read nor modified. It is assumed that the user surely had to authenticate using a username and password – that covers goal three. But what about non-repudiation, how does the customer have proof of the completion of the desired transaction? If the bank fails to transfer that money and the customer has checks that bounce, what recourse does the customer have? How can the customer prove that he made the request for the transfer of funds?

Now the next question, how does the customer know that he is really connected to his bank? The average person may naively say well I went to www.mybank.com so it must be my bank. But that is not necessarily the case. Suppose the customer believes that he has logged into www.mybank.com but is actually communicating with www.thehacker.com who has a trusted SSL certificate. He spoofs the www.mybank.com site and the customer attempts to connect to it. The customer enters his username and password, which is encrypted and sent to www.thehacker.com where it is decrypted. The hacker then uses that information to log into the actual www.mybank.com site. The hacker can then continue to mirror the actual information from the bank and make whatever modifications they desire. This scenario shows failure of satisfying all four goals. What is to be done?

The Method

These are the types of questions that PKI attempts to answer. At this point a couple of assumptions need to be made, these will be explained later. The first assumption is that both the bank and the customer have a DSA key pair[11]. The second is that they both have certificates, which they trust for the other's public keys. With these assumptions made it is possible to describe a system where all four goals are satisfied. The following does this using the Diffie-Hellman[12] key exchange and DES[13] symmetric encryption. The basic schematic follows. The customer goes to www.mybank.com and connects. Through some online application the bank has put together the customer creates a Diffie-Hellman or DH key pair. The customer through this application now sends the public key of the DH key pair to the bank. But before doing this, the customer signs this public key using the private key of their DSA key pair assumed above, which the bank has the trusted certificate for. The bank then receives this information, it verifies the signature using the trusted certificate, from the customers DSA key pair, and then generates its own DH key pair, and signs the DH public key. The bank then sends the customer their signed public key. The customer verifies the banks signature, again using the DSA trusted certificate assumed at the beginning, and then both the customer and the bank can derive the shared secret by finishing the Diffie-Hellman process. Now

[11] RSA Laboratories - 3.4.1 What are DSA and DSS?"
[12] RSA Laboratories - 3.6.1 What is Diffie-Hellman?
[13] RSA Laboratories - 3.2.1 What is DES?

that the two both have the same shared secret they can use that along with the DES algorithm to encrypt communication between the two of them. The next step would then be to actually authenticate the customer using a username and password, which now cannot be read or intercepted by any outside parties. Once the customer is authenticated they can continue conducting bank transactions.

How are the four previously discussed goals achieved using PKI? The first step of success is in the realm of confidentiality and integrity of the data. These are because of a couple of different things. The first is through the use of encryption, but if this were all then this would be no better off than using SSL. The real success comes through the process of signing and verifying information sent between the customer and bank. The act of signing the data gives both the customer and the bank a way of verifying whom they are communicating with. As an added verification by the bank, the customer then authenticates using their username and password. This is also adds another level of security for the bank if the customers DSA key pair are stolen. The fourth goal is also achieved through this signing and verification process. If the customer sends a request to the bank and signs that request the bank has proof that the request was made, and if the response from the bank of success or failure is signed the customer has proof of its outcome. The customer now has a way to prove in court that the bank received the given request. Now it is time to take a closer look at the assumptions above.

The assumption was made that both the bank and the customer have a DSA key pair and also that they trust the certificates for the others public keys. It is these two assumptions that create the true complexity of PKI. It is through the fact that the two have a pre-existing relationship that some of this complexity can be removed. If the bank wants to create a new website for online banking they can also create a utility that will assist the customer in creating a DSA key pair and that will import the banks certificate, as a trusted certificate. The customer would want to be able to verify the certificate they receive from the bank. This could be done in many ways including printing the certificate's fingerprint on the customers monthly statement, confirmation by telephone or simply by going to the bank and verifying it in person. Likewise, when the customer creates their DSA key pair they can send their certificate to the bank which can in turn verify it either over the phone or by requesting the customer come to the bank with ID.

This scenario creates the needed trust between the two entities without the need for the full web of trust. It does not give the freedom that is the full vision of PKI, but it gives the security needed by online banking and other such tasks where a pre-existing relationship exists.

The Implementation

It is now necessary to move on to the implementation of such a system. The most recent version of Java was chosen for several different reasons. First it can be used for both the server and the client. For the basic implementation, servlets were used on the server side and applets were used on the client side. The second and maybe more relevant reason for using Java is that in what is to become the next release 1.4, which is

now in beta testing, the entire encryption API has been moved into the standard development kit so there is no need to add extra API's to the Java Runtime Environment (JRE). This encryption API includes functions for the Diffie-Hellman key agreement, DES and message signing and verification. Furthermore, if one chooses not to use the Sun Microsystems implementation, he can write his own implementation of each of the functions.

The following is sample code that a bank could use to implement such a scenario. Following each code section, there is a discussion of exactly what is occurring. The code that is executed on the client's computer will be in blue and the code that is executed on the server is in red.

```
FileInputStream keyIn = new FileInputStream("C:\\sanskeys\\clientkeystore");
KeyStore ks = KeyStore.getInstance("JKS");
char[] storepass = "theStorePassPhrase".toCharArray();
char[] keypass = "thePrivateKeyPassPhrase".toCharArray();
ks.load(keyIn, storepass);
```

This gives the customer access to the DSA key pair that has been created and stored on their machine. The code first specifies where the key store is located and the format it is in. In the Sun implementation of their key store they also require a pass phrase to access the store not just the private keys within it. In a production implementation of this code you would want to prompt for the two pass phrases.

```
Key theKey = ks.getKey("client", keypass);
KeyFactory signKeyFac = KeyFactory.getInstance("DSA");
KeySpec dsaKeySpec = signKeyFac.getKeySpec(theKey, DSAPrivateKeySpec.class);
PrivateKey privateKey = signKeyFac.generatePrivate(dsaKeySpec);
Signature theSigSign = Signature.getInstance("DSA");
theSigSign.initSign(privateKey);
```

The previous code actually assigns the DSA private key with an alias of "client" to a variable. The program then uses the key to generate the signing object that will be used to sign the Diffie-Hellman public key that is sent to the server.

```
DHParameterSpec dhParamSpec;
AlgorithmParameterGenerator paramGen = AlgorithmParameterGenerator.getInstance("DH");
paramGen.init(512);
AlgorithmParameters params = paramGen.generateParameters();
dhParamSpec = (DHParameterSpec)params.getParameterSpec(DHParameterSpec.class);
KeyPairGenerator clientKeyPairGen = KeyPairGenerator.getInstance("DH");
clientKeyPairGen.initialize(dhParamSpec);
KeyPair clientKeyPair = clientKeyPairGen.generateKeyPair();
KeyAgreement clientKeyAgree = KeyAgreement.getInstance("DH");
clientKeyAgree.init(clientKeyPair.getPrivate());
```

Now the program sets the number of bits used for the prime modulus and random exponent used in the DH key pair, and then generates those numbers. Once those have been created the keys are actually generated.

```
byte[] clientPubKeyEnc = clientKeyPair.getPublic().getEncoded();
theSigSign.update(clientPubKeyEnc);
byte[] theSigSignArray = null;
theSigSignArray = theSigSign.sign();
```

Once the keys have been created the public key is extracted to a simple byte array and that array is then signed with the signing object created above. After this is done both

the byte array storing the client public key and the byte array containing the signature can be sent to the server.

```
FileInputStream keyIn = new FileInputStream("C:\\sanskeys\\serverkeystore");
KeyStore ks = KeyStore.getInstance("JKS");
char[] storepass = "theStorePassPhrase".toCharArray();
char[] keypass = "theKeyPassPhrase".toCharArray();
ks.load(keyIn, storepass);
Key theKey = ks.getKey("server", keypass);
KeyFactory signKeyFac = KeyFactory.getInstance("DSA");
KeySpec dsaKeySpec = signKeyFac.getKeySpec(theKey, DSAPrivateKeySpec.class);
PrivateKey privateKey = signKeyFac.generatePrivate(dsaKeySpec);
Signature theSigSign = Signature.getInstance("DSA");
theSigSign.initSign(privateKey);
```

Here again the server is accessing the DSA key pair stored on the server and using the private key to generate the signing object that will be used to sign its response to the client. In an actual server it is more likely that the opening and access to the server's DSA key pair would take place when the server process was started and would then ask for the two pass phrases to access them.

```
java.security.cert.Certificate clientCert = ks.getCertificate(clientAlias);
Signature theSigVerify = Signature.getInstance("DSA");
theSigVerify.initVerify(clientCert);
theSigVerify.update(clientPubKeyEnc);

if (theSigVerify.verify(theSigVerifyArray))
{
        verified = true;
}
```

What is shown above is the access of the client's certificate on the server and then using the certificate to initialize a verification object. The verification object is then given the signature byte array received from the client. This is then checked against the signature the server calculates using the client's certificate.

```
KeyFactory serverKeyFac = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(clientPubKeyEnc);
PublicKey clientPubKey = serverKeyFac.generatePublic(x509KeySpec);
DHParameterSpec dhParamSpec = ((DHPublicKey)clientPubKey).getParams();
KeyPairGenerator serverKeyPairGen = KeyPairGenerator.getInstance("DH");
serverKeyPairGen.initialize(dhParamSpec);
KeyPair serverKeyPair = serverKeyPairGen.generateKeyPair();
KeyAgreement serverKeyAgree = KeyAgreement.getInstance("DH");
serverKeyAgree.init(serverKeyPair.getPrivate());
serverPubKeyEnc = serverKeyPair.getPublic().getEncoded();
```

Once the client's public key has been verified, the server can begin building its Diffie-Hellman key pair.

```
theSigSign.update(serverPubKeyEnc);
byte[] theSigSignArray = null;
theSigSignArray = theSigSign.sign();
```

Now that the server's DH key pair has been generated it can sign its DH public key and it can be sent back to the client.

```
serverKeyAgree.doPhase(clientPubKey, true);
byte[] serverSharedSecret = serverKeyAgree.generateSecret();
int serverSharedSecretLen = serverSharedSecret.length;
serverKeyAgree.doPhase(clientPubKey, true);
SecretKey serverDesKey = serverKeyAgree.generateSecret("DES");
byte[] serverDesKeyEnc = serverDesKey.getEncoded();
```

Since the server has both its DH key pair and the clients public key it can go ahead and generate the shared secret and use it to create a DES key.

```
java.security.cert.Certificate serverCert = ks.getCertificate(serverAlias);
Signature theSigVerify = Signature.getInstance("DSA");
theSigVerify.initVerify(serverCert);
theSigVerify.update(serverPubKeyEnc);

if (theSigVerify.verify(theSigVerifyArray))
{
        verified = true;
}
```

The server public key is received and again its signature is verified.

```
KeyFactory clientKeyFac = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(serverPubKeyEnc);
PublicKey serverPubKey = clientKeyFac.generatePublic(x509KeySpec);
clientKeyAgree.doPhase(serverPubKey, true);
byte[] clientSharedSecret = clientKeyAgree.generateSecret();
int clientSharedSecretLen = clientSharedSecret.length;
clientKeyAgree.doPhase(serverPubKey, true);
clientDesKey = clientKeyAgree.generateSecret("DES");
```

Now that the server's public key is verified it can be used to generate the shared secret, which is again used to generate the DES key.

```
Cipher clientCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
clientCipher.init(Cipher.ENCRYPT_MODE, clientDesKey);

byte[] cipherUser = clientCipher.doFinal(loginInfo.username.getBytes());
byte[] cipherPass = clientCipher.doFinal(pass.getBytes());
```

For the last step in setting up the initial connection the client needs to be authorized. Here we create a cipher object that is set up for encryption using the DES key created above. Both the username and password are then encrypted and can be sent off to the server.

```
Cipher serverCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
serverCipher.init(Cipher.DECRYPT_MODE, serverDesKey);
String clearUser = new String(serverCipher.doFinal(cipherUser));
String clearPass = new String(serverCipher.doFinal(cipherPass));
```

Here the server creates a cipher object initialized for decryption using the DES key created above. And then both the username and password are decrypted and can be used to authenticate the client. Once that is complete the servers reply would be sent back to the client who if verified would continue with their desired transactions and if failed would probably retry authenticating.

An additional note, in a production environment the pass phrases on both the client and server should be stored in a character array as opposed to a String object. The String object will be left in memory until the JVM destroys it, even if the String is no longer being used. In the case of a character array you would set it to "null" once you were done with it and the memory used to store it would be recycled more quickly.

The above sample clearly covers the first three goals of confidentiality, integrity of data and authentication. As far as non-repudiation both entities can confirm the connection was initialized, but in the code above it is not even possible to confirm that it was successful. In order to fully achieve non-repudiation it will be necessary to sign each transaction, both request and response. Doing this will fully achieve all four goals.

The Draw Backs

As with every system, there are problems. This implementation involves much more setup on the part of the bank and the customer. The customer does not have the freedom of logging into their bank from whatever computer they have access; they only have access from the computer that stores the DSA key pair that is trusted by the bank. It is possible to copy that key pair to other machines or to create a second DSA key pair and sign the new public key using the private key of the original DSA key pair. If someone who is trusted by the bank signs the certificate then the bank could import the new certificate as a trusted certificate. In general this is how the web of trust works, but in this case it may not be wise for the bank to work in this fashion. This is due to the fact that the bank lays the determination of trust on the customer and that could cause legal liability issues. The third possibility is for the customer to simply set up DSA key pairs on multiple machines and individually confirm these with the bank. This may take a little more time and energy, but it is safer for the bank and it gives the customer the ability to bank online from other locations where he spends considerable time.

Of the various drawbacks one significant limitation is time. Generating the Diffie-Hellman key pairs is process intensive, which would be noticeable on a clients PC but more importantly could severely impact the performance of the servers it is running on.

There are many who would say the use of DES encryption is not enough. Perhaps this is true, but there are many considerations that need to be taken into account. The use of say triple-DES would be much more secure, but it would also be much more processor intensive. It could triple the number of times the Diffie-Hellman key agreement process needs to run. On the other hand, connections to a bank for these purposes are short term, so is the use of DES really a security hazard? It could even be configured that connections that last longer than ten minutes would require the creation of a new, shared secret using Diffie-Hellman. This makes sure the use of a key is limited, but does not have the overhead of creating three shared secrets for every connection. These are just some of the questions that would need to be investigated further for an actual implementation.

The Conclusion

Due to the increased sophistication of hackers, the demand of the general public for truly secure online connections and the need to protect confidential and financial information it has been shown that the 128 bit SSL encryption currently in place is not sufficient from a security standpoint. Ideally, a complete implementation of a full public-key infrastructure in a manageable amount of time is preferred. However, this is neither likely nor really feasible due to financial, personnel and time limitations imposed on banking. The goal of this paper is to show how to implement a more feasible middle

solution. The protocol and code presented in this paper do not fulfill the ultimate goal of complete transaction security throughout the entire Internet. However, this paper illustrates the possibility of accomplishing the four described goals on an individual site-by-site basis where a pre-existing relationship exists. The future is unknown in terms of PKI implementation and new security advances; however, perhaps small-scale implementation will lead to the necessary infrastructure for full scale PKI.

Bibliography

"2.2.2 What is a digital signature and what is authentication?" RSA Laboratories. 26 December 2001. URL: http://www.rsasecurity.com/rsalabs/faq/2-2-2.html

"3.2.1 What is DES?" RSA Laboratories. 26 December 2001. URL: http://www.rsasecurity.com/rsalabs/faq/3-2-1.html

"3.4.1 What are DSA and DSS?" RSA Laboratories. 13 January 2002. URL: http://www.rsasecurity.com/rsalabs/faq/3-4-1.html

"3.6.1 What is Diffie-Hellman?" RSA Laboratories. 26 December 2001. URL: http://www.rsasecurity.com/rsalabs/faq/3-6-1.html

"4.1.3.1 What is PKI?" RSA Laboratories. 13 January 2002. URL: http://www.rsasecurity.com/rsalabs/faq/4-1-3-1.html

"4.1.3.10 What are certificates?" RSA Laboratories. 26 December 2001. URL: http://www.rsasecurity.com/rsalabs/faq/4-1-3-10.html

Burton, Tinabeth. "ITAA Poll Finds Almost Three of Four Americans Concerned about Cyber Security." ITAA Press Release. 9 January 2002. URL: http://www.itaa.org/news/pr/PressRelease.cfm?ReleaseID=1008095083

Cole, Eric. "Introduction to Encryption I." The SANS Institute. Chicago, Illinois. 5-10 November 2001.

"December 27, 2001: SCIENCE & TECHNOLOGY – Internet Advances." Oxford Analytica – Weekly Column. 9 January 2002. URL: http://www.oxan.com/columns/wkcol_27122001.html

Fenn, Donna. "Hacked! Web-based business UltiMutt's experience with computer hackers." Inc. v.23 no. 12 (September 2001): 134-139.

Rovelto, Rose. "Security – an issue that should concern you! Part 2." National Public Accountant v. 46 no. 5 (July 2001): 35 – 37.

"VeriSign Security Alert Fraud Detected in Authenticode Code Signing Certificates." VeriSign Inc. 9 January 2002. URL: http://www.verisign.com/developer/notice/authenticode/

Last Updated: May 28th, 2017

Upcoming Training

SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 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
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
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	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
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Minneapolis SEC401	Minneapolis, MN	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Colorado Springs SEC401	Colorado Springs, CO	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Charleston SEC401	Charleston, SC	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event