Interested in learning more about security?

SANS Institute InfoSec Reading Room

Distributed Systems Security: Java, CORBA, and COM+

Security can have varying levels of difficulty for implementation. One factor in determining the difficulty is the number and distribution of the systems. With distributed systems architecture, there are different nodes and resources. One major issue with distributed systems is application security. There is the question of how security is handled in distributed applications, and how the client handles applications coming from an unknown source. The purpose of this paper is to examine three popular architectures for di...

Distributed Systems Security: Java, CORBA, and COM+

April L. Moreno

GSEC v.1.4b

Abstract

Security can have varying levels of difficulty for implementation. One factor in determining the difficulty is the number and distribution of the systems. With distributed systems architecture, there are different nodes and resources. One major issue with distributed systems is application security. There is the question of how security is handled in distributed applications, and how the client handles applications coming from an unknown source. The purpose of this paper is to examine three popular architectures for distributed systems applications and their security implications. The architectures analyzed are Java by Sun, CORBA by the OMG, and COM+ from Microsoft. Outstanding issues and future areas for research are considered.

Distributed Systems Security

Security can have varying levels of difficulty for implementation. One factor in determining the difficulty is the number and distribution of the systems. When only individual systems need to be protected, such as one computer with all files residing locally and with no need to connect to any outside resources, security is not as complex as with distributed systems. With distributed systems architecture, there are different nodes and resources. One major issue with distributed systems is application security. There is the question of how security is handled in distributed applications, and how the client handles applications coming from an unknown source. The purpose of this paper is to examine three popular architectures for distributed systems applications and their security implications. The architectures analyzed are Java by Sun, CORBA by the OMG, and COM+ from Microsoft. It is extremely important for developers to consider the security implications when designing distributed applications, as many of these applications offer access to crucial resources: financial, medical, and military information, just to name a few. This paper will not address authentication controls, physical protection of the systems, patches, firewalls, network protocols, etc, as they are beyond the scope of the paper.

Java

The Java architecture for distributed systems computing was designed taking security requirements into consideration. Developers need to create programs that are executed on remote distributed systems. An architecture needed to be put in place, however, that would not leave these systems vulnerable to malicious code. This was accomplished through the Java architecture. The source code is written and then converted to byte code and is stored as a class file, which is interpreted by the Java Virtual Machine (JVM) on the client. Class loaders then load any additional classes that are needed by the applications.

Several security checks are put between the remote server distributing the program, and the client executing it, such as the “sandbox” security model, the byte code verifier, the applet class loader, the security manager, and through other security measures that can be implemented through Java’s security APIs.

Sandbox Security Model

In a distributed architecture, the end users would ultimately be responsible for determining which applets to run on their systems. Most of these users would not be able to determine whether a particular applet should be trusted or not. In order to have all applets run in a protected environment, the sandbox security model was developed. Applets that run from a remote site would be permitted only limited access to the system, while code run locally would have full access.

If the applet is signed and trusted, then it can run with full local system access. Permissions can be set by a security policy that allow the administrator to define how the applets should be run.

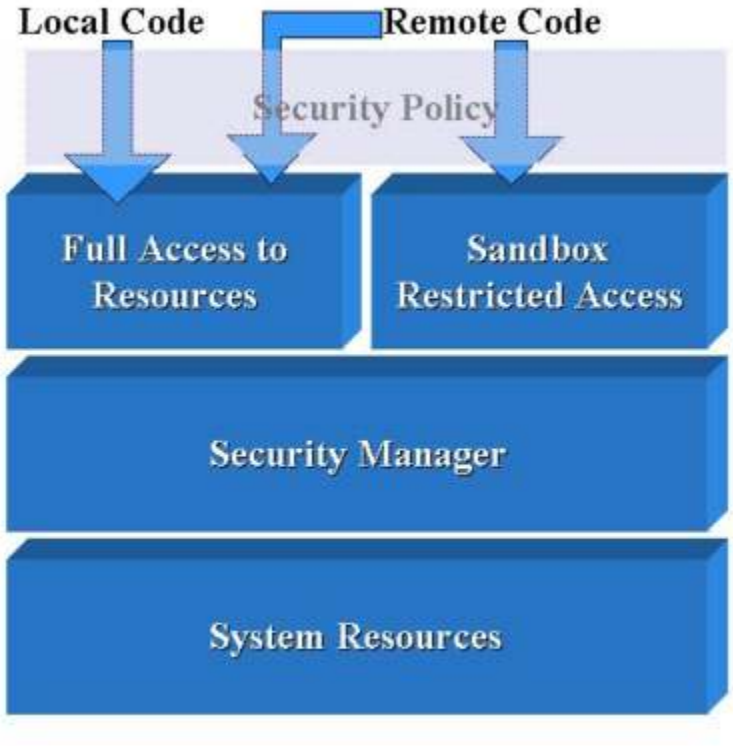

Figure 1 - Java model

Byte Code Verifier

The byte code verifier looks at the class files that are to be executed and analyzes them based on specific checks. The code will be verified by three or four passes (MageLang Institute, 1998) depending on whether or not any methods are invoked. Gollmann (2001) states that some of the checks performed are to ensure that the proper format is used for the class, to prevent stack overflow, to maintain type integrity, to verify that the data does not change between types, and that no illegal references to other classes are made. Hartel and Moreau (2001) further state that the byte code verifier ensures that jumps do not lead to illegal instructions, that method signatures are valid, access control, initialization of objects, and that "subroutines used to implement exceptions and synchronized statements are used in FIFO order" (p. 520).

Applet Class Loader

As a Java application is executed, additional classes may be called. These classes are not loaded until they are needed. When they are called the applet class loader is responsible for loading the specified applets. Classes in Java are organized by name spaces, and each class loader is responsible for one name space. The class loaders are therefore responsible to protect the integrity of the classes in its name space (Gollmann, 2001). Java has built-in classes that reside locally, however, that are loaded automatically without any security checks. The path to these classes is indicated by the CLASSPATH environment variable.

Security Manager

When writing applications, developers often wish to protect variables and methods from being modified by classes that do not belong to the group of classes they have written. In order to create this division, classes are grouped into packages. When a variable or method is declared in a class, it can be private (access only through same class), protected (access through class or subclass), public (any class can access), or they may chose not to use any of the former, in which case only classes within the same package will have access. Depending on the package that a class belongs to, the class will have different access to the other classes in the package, so security could be compromised if an unauthorized class attaches itself to the package. The security manager makes sure that only classes that actually belong to the package in question are able to declare themselves in this package. The security settings are configured through a security policy.

Browsers and applet viewers have a security manager, but by default Java applications do not (Sun Microsystems, n/d). Java has provided developers the means to create their own security manager. To create it, the developer must create a subclass of the SecurityManager class, and override whichever methods are necessary to implement the required security. For example, the developer may decide to impose a stricter policy for reading and writing files. This could be attained through overriding the read and write methods already defined in the superclass.

API Security

Java offers further security through several security APIs. Among the different APIs provided, the developer can make use of signed applets, digital signatures, message digests, and key management. When an applet is signed it is given full access to the system as if it were run locally. As mentioned in the section on the security manager, the security policy defines what permissions are given to an application or applet when executed. The default Java Runtime Environment provides digital signatures, message digests, and key management, and encryption can be implemented through the Java Cryptography Extension (JCE).

Outstanding Issues

As with any system, whether it has been designed around security or not, the Java distributed architecture contains several outstanding security problems. One problem is with the CLASSPATH system environment variable. As mentioned previously, the CLASSPATH variable is used to determine the location of the built-in Java system classes. If the CLASSPATH variable is altered, it could point to a set of altered classes that may execute what the original classes intended, but also insert malicious code. The code would be executed, and the user may not notice any difference in the behavior of the application (Golmann, 2001).

Wheeler, Conyers, Luo, and Xiong (2001) found that there are several Java vulnerabilities if a computer serving Java applications is either compromised from the inside, or if an attacker is able to compromise an account on the server. They note that many of these vulnerabilities exist either because of code that provides backwards compatibility, or because of decisions made to increase the ease of implementation. In other words, the vulnerabilities are due to design choices rather than software defects. First they found that "many critical components of the Java environment are only protected by the underlying operating system's access control mechanisms" (p. 65). System administrators may not be aware of the loose access controls, and critical components could be compromised, such as the keystore and system classes. If the keystore is compromised then signed files could be spoofed, and if the classes are modified, malicious code could be inserted. Wheeler et al. further note the ease of reverse-engineering of class files, which would allow an attacker to obtain the original source code. They note that there are tools for obfuscation, but suggest in their work that further obfuscation would be necessary for a higher level of security.

As discussed earlier, a security policy can be set to limit the access of applications or applets to the local system. Wheeler et al. discuss that the permissions, although fine-grained, can only be applied to a directory or JAR file. They state, "this is insufficient, except for the most rudimentary system" (p. 66). Permissions applied to the entire directory or JAR file, which violates the principle of least-privilege. They suggest finer permissions that could extend down to the class level. The security policy can also be either modified or overwritten completely through the use of the "java.security.policy" option from the command line, negating any work put into the creation of the security policy. This behavior can be turned off, but is not by default – an example of vulnerabilities being introduced for the sake of ease of implementation. They suggest that the class loader should verify that an extended security manager is loaded prior to loading any classes.

Hassler and Then (1998) discuss the possibility of using applets to perform "a degradation of service attack" (p. 120). Security policies can be created, and are usually part of the browser, to limit the access given to Java applets. They show in their research, however, that this does not prevent the applet from consuming sensitive resources such as CPU and memory. They

suggest the implementation of a special applet that would allow other applets to be controlled, and note at the end of their work that the HotJava browser included this, but was found to be insufficient. One must wonder, however, if an average user would have the knowledge necessary to identify that a Java applet is creating the degradation of service, and how to stop it.

Finally, an outstanding issue is that of auditing. A major component of security systems is the ability to audit. Hartel and Moreau (2001) state that there is no known work presently being done to implement auditing capabilities in Java.

CORBA

CORBA (Common Object Request Broker Architecture), created by the Object Management Group (OMG), is based on the fact that developers are not be able to agree on common language, such as Java, or on a common operating system. Therefore, there is a need to have a security layer between applications and the operating system. The Object Management Group (n/d, ¶3) states that a CORBA application "from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network."

General Architecture

CORBA-based applications interact with objects. Object-oriented languages may have several instances of an object, or a legacy application may have a wrapper making it one object. In order for objects to interact with each other, and for users to interact with the objects, an Interface Definition Language (IDL) is created for each type of object. Information is passed to the IDL, which in turn brokers it to the correct object, which interprets the information, and sends any requested information back to the caller through the IDL. Standardized mappings for C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript have been created (Object Management Group, n/d). Each object instance has a unique entry in the Object Request Broker (ORB) which handles requests between objects and between user and objects.

Objects with similar security requirements are grouped into domains, and a security policy is applied to the domain, which is enforced by the ORB. Communication between ORBs is handled by " bridges, gateways, and inter-ORB protocols like the General Inter-ORB protocol (GIOP) and the Internet Inter-ORB Protocol (IIOP)" (Gollmann, 2001, p. 182).

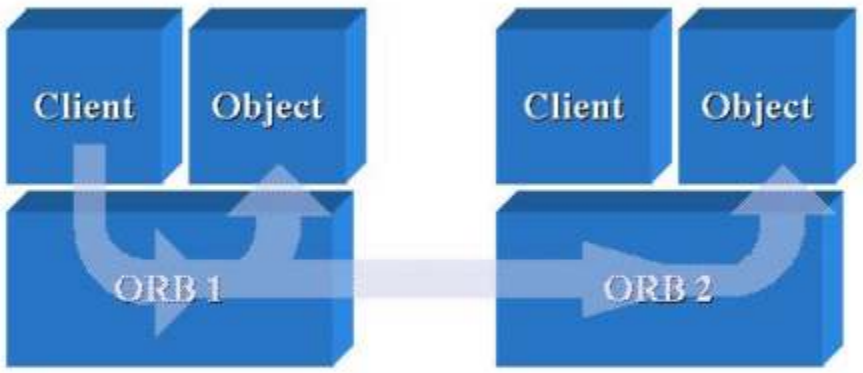

Figure 2 - CORBA model

There are several different security aspects that are covered in the CORBA specification. The Object Management Group (2002) specify key security features of CORBA including: Identification and authentication, authorization and access control, security auditing, non-repudiation, and administration.

Identification and Authentication

Identification and authentication deal with verifying that the user really is who they claim to be. A user (or a system) will authenticate, for example using a password – the user or system is referred to as the principal. This is used as a means for accountability, for access to objects with different permissions, for identification of the principal sending information, for controlling access to different objects, message signing, and usage charging for object implementation (Chimadia, 1998, Object Management Group, 2002).

Authorization and Access Control

As users are authenticated, the applications will use those credentials to access other objects through the ORB, where the CORBA security service operates. A security policy defines what objects the principal has been given access to and through the policy's implementation at the ORB level, access is either granted or denied.

Security Auditing

A key component to security is auditing. Auditing enables the administrator to detect intrusions, attempted intrusions, or other security anomalies. It also allows the administrators to verify that the security policies are working as expected. There are two types of auditing in CORBA: system and application. Which events are to be logged is determined by the respective audit policies. System policies could include the logging of events such as the authentication of principals, when privileges change, whether the invocation of an object was successful or not, and events relating to administration. This audit is automatically enforced for all applications, which is especially helpful in the case of applications that are not security-aware. The application-level auditing could be specific to the application, such as the auditing of specific transactions.

Non-Repudiation

Non-repudiation services in CORBA ensure that the principal is held accountable for their actions. Evidence is maintained that will either prove or disprove a particular action. CORBA provides for non-repudiation of creation and non-repudiation of receipt, the former proving whether or not a principal created a message, and the latter whether or not a principal received the message. Unlike auditing, however, this service is only available to applications that are aware of and write to this service.

Administration

Security is administered in CORBA through the use of domains, which refer to the scope or boundaries of the items being examined, grouped by some commonality. There are three security domains: security policy domains, security environment domains, and security technology domains. Within security policy domains, certain items must be administered: the domains themselves, the members of the domain, and the policies associated with the domains. The environment domains refer to the "characteristics of the environment and which objects are members of the domain" (Object Management Group, 2002, p. 2-27). Since this is specific to the environment, they do not provide management interfaces. Finally technology domain administration may refer to establishing and maintaining the security services, the trusts between the different domain, and any other entities such as principals and keys, that would be within the scope.

Outstanding Issues

Xingshe and Xiadong (2000) note that is there is inconsistency among the different security models for the applications being integrated through CORBA, there may be an opportunity for an intruder to compromise the architecture's security.

Koutsogiannakis and Chang (2002) state that "a Corba implementation is labor- and resource-intensive" (p.41). The more a developer has to learn about an architecture to create a secure system, the more security problems may be introduced due to that lack of understanding. The architecture's complexity in itself can make security more difficult to ensure.

Gollmann (2001) shows that security is based on all requests being brokered through the ORB. It "does not guarantee that the ORB cannot be bypassed and that the data used by CORBA's security services are properly protected" (2001, P. 184). Further, the fact that non-repudiation is not at the ORB level, and has to be implemented at the application level, can weaken the security. In other words, there is no proof that the ORB actually carried out the request on an object – non-repudiation is only at the application level, which must be taken into consideration.

COM+

The third architecture that is often mentioned in discussions of distributed systems application security is Microsoft's COM+ architecture. COM+ represents the next generation in Microsoft's history of distributed architectures.

General Architecture

Previous distributed systems architecture called for two-tier programming. The client would run software that would allow it to connect to other back-end systems, such as SQL databases. This was found to be problematic, so the architecture progressed to multi-tier, or n-tier application development. In this architecture, the clients would run an application that would connect to a server with COM+ services running. This server would in turn connect to the back-end servers. This offered many benefits, such as sharing of resources on the COM+ server, and limited updating of clients. As with Java, interface-based programming was implemented. Various languages can use the COM+ architecture, such as C++, Visual Basic, Java, Delphi, and COBOL (Pattison, 2000).

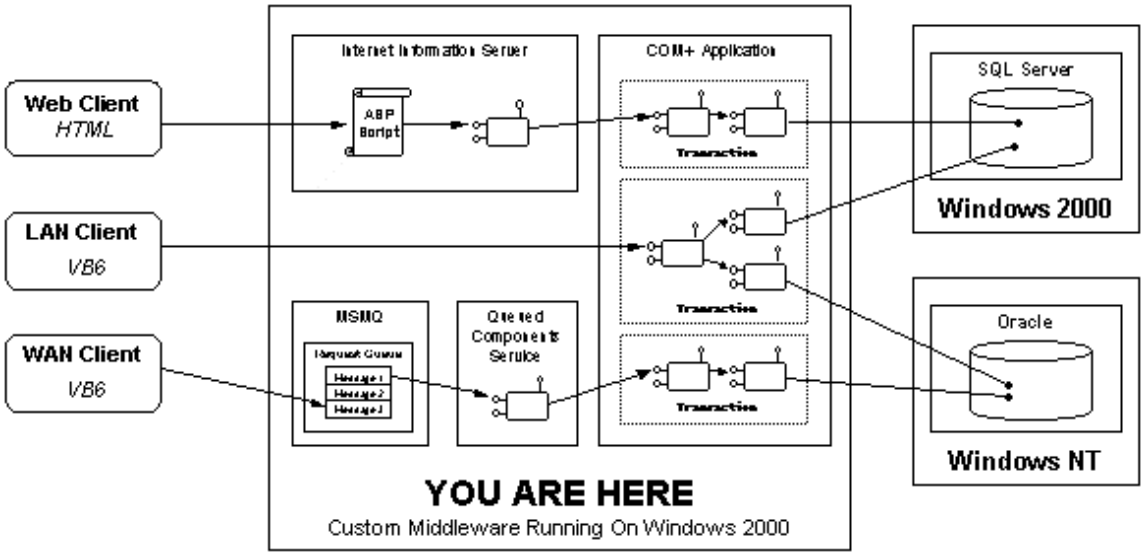

Figure 3 - COM+ Model (Pattison, 2000, p.24)

COM+ includes automatic security. With COM+ automatic security, the developer can leave security out of the components they create. This makes the code easier to write and maintain, it is easier to design security at a higher level and throughout an entire application, and it facilitates the configuration of a security policy. The developer can also build upon these automatic security features. Some of the options they have at their disposal include: role-based security, impersonation and delegation, and software restriction policies.

Role-Based Security

Role-based security, an automatic service of COM+, can be extended in order to construct and enforce access policies. The security is not placed in the component itself, but implemented rather on a method-by-method basis. Role-based security can be implemented either as declarative or programmatically ("COM+ Security Programming, Part 1", 2001). In declarative security, permissions can be set on components much the same way that permissions are set on files within the Windows NT operating system. This has the advantage of allowing security to be administered and configured without having to recompile code. This also frees the component developer from concerning themselves with writing security into the components themselves.

If the permissions need to be more granular than at the component level, then role-based security must be implemented programmatically. In order for COM+ to authorize a client to access some resource, it must determine who the client is, through the authentication service. There are several authentication options, but the higher the level of security, the bigger the performance hits, which needs to be taken into consideration.

Impersonation and Delegation

When a client accesses a resource, often the server that the client is connecting to needs to retrieve information with the client's credentials, ensuring that the client can only access information that it has been granted rights to. This is done in COM+ through impersonation. In the case of distributed systems, delegation would be more frequently used. Delegation refers to the impersonation of a client over the network. For example, if a client is running an application that makes a call to the COM+ middleware server, and it in turn needed to access a SQL database, delegation would be used.

Software Restriction Policies

Introduced with the release of Windows XP is the proactive framework of software restriction policies. A similar architecture as discussed in the section on Java security is used. Trusted code is given unrestricted access to the local system, where unknown (untrusted) code is limited to a sandbox where the access is restricted. The access is determined by setting one of two trust levels: unrestricted and disallowed. Unrestricted will allow the code to execute up to the limits given to the user executing the code, whereas disallowed is restricted to the sandbox. As with role-based security, software restriction policies can either be set through a graphical user interface (GUI) or programmatically (Microsoft, 2002).

Outstanding Issues

The first outstanding issue that is evident as current research is analyzed is the marked absence of evaluation of the COM+ architecture and its current security problems. Microsoft is often criticized for security problems, but it is difficult to find solid research detailing what problems may exist in the COM+ architecture.

Further, it does not appear that Microsoft's sandbox allows granular enough permissions. Only two levels of trust can be configured, which would appear to be inadequate for most implementations.

Future Research

There are several areas that warrant additional research in order to further distributed systems application security. The first area that should be explored is that of Java auditing. Hartel and Moreau (2001) write that there is currently no work being done to implement auditing in Java. This is a major part of securing systems that should be investigated further.

Hartel and Moreau (2001) stat that there have been many investigations into the specifications of a subset of the language, but a "unifying frameworks that help understand interactions between components" is still needed (p. 530). They state that a complete understanding of the language and specifications are needed to implement better security. They also suggest that tools are needed to

work on and analyze the byte code of a program, as the original source code may not be available.

An area of further research for the CORBA architecture is simply watching the industry, how it implements this framework, and analyzing the implementations for further security enhancements. CORBA is a new framework and there has not been a very high level of implementation. With any new system, there is a learning curve for the developer community, and additional security problems will be found along the way.

Although many compare Java and CORBA to COM+ in their research, and regard COM+ as a viable solution for distributed systems security, (Chizmadia, 1998, Emmerich & Kaveh, 2002, and Pattison, 2000), not a lot of research has been conducted on it. Additional research should be completed to further understand the security implications of COM+, comparing it thoroughly to Java and CORBA in order to assist the industry in deciding which architecture to implement.

Finally, as with any architecture, there is the continuous battle between ease of implementation, backwards compatibility, and security. It is extremely difficult, if not impossible, to find the ideal balance, but additional research should be conducted in an attempt to at least arm the developer community with the knowledge necessary to make an educated decision.

Conclusions

The three most common distributed systems application architectures are Java, CORBA, and COM+. Each of these architectures has its strengths and weaknesses. When deciding which to employ, different aspects that need to be considered. First, the security of the architecture must be considered. Java has several published security vulnerabilities, but knowing what they are is half the battle towards finding a remedy. CORBA does not appear to have many, but it also has not been as widely implemented as Java, so the vulnerabilities may yet be discovered. Security for COM+ may be inconclusive as there is not a wealth of information on it. The difficulty of implementation must also be considered. If the system is overly complex, security problems may exist due to implementation problems. If the architecture is too simple however, there may not be enough flexibility to create the necessary security configurations. Finally, the environment needs to be taken into consideration. Koutsogiannakis & Chang (2002), among others, recommend Java when the systems that need to communicate are primarily Java-based, COM if the environment is mostly Microsoft based in order to take advantage of close integration with the other Microsoft products, and CORBA as a general implementation.

References

Chizmadia, D. (1998). *A quick tour of the CORBA security service*. Retrieved August 27, 2002, from http://www.itsecurity.com/papers/corbasec.htm

COM+ security programming, part 1: declarative role-based security. (2001, June 11). Retrieved August 28, 2002, from http://www.itworld.com/nl/windows_sec/06112001/

COM+ security programming, part 2: programmatic role-based security. (2001, June 18). Retrieved August 28, 2002, from http://www.itworld.com/nl/windows_sec/06182001/

Emmerich, W., & Kaveh, N. (2002). Component technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA component model. *Software Engineering, 691-692.*

Gollmann, D. (2001). *Computer security*. West Sussex, England: John Wiley & Sons Ltd.

Gong, Li. (1997). Java security: present and future. *IEEE Micro, 17, 14-19.*

Hartel, P., & Moreau, L. (2001). Formalizing the safety of Java, the Java virtual machine, and Java card. *ACM Computing Surveys, 33, 517-558.*

Hassler, V., & Then, O. (1998). Controlling applets' behavior in a browser. *Computer Security Applications Conference, 120-125.*

Koutsogiannakis, G., & G. Chang, J. M. (2002). Java distributed object models: an alternative to Corba? *IT Professional, 4, 41-47.*

MageLang Institute. (1998). *Fundamentals of Java security*. Retrieved September 4, 2002, from http://developer.java.sun.com/developer/onlineTraining/Security/Fundamentals/Security.html

Microsoft. (2002, August). *COM+ security concepts.* Retrieved August 28, 2002, from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cossdk/htm/pgservices_security_4fw3.asp

Object Management Group. (n/d). *CORBA basics*. Retrieved August 21, 2002, from http://www.omg.org/gettingstarted/corbafaq.htm

Object Management Group. (2002, March). *Security service specification, version 1.8.* Retrieved September 6, 2002, from http://www.omg.org/cgi-bin/doc?formal/2002-03-11

Pattison, T. (2000, February). COM+ overview for Microsoft Visual Basic programmers. *DevelopMentor,* Retrieved September 6, 2002, from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomser/html/complus4vb.asp

Sun Microsystems (n/d). *Trail: security in Java 2 SDK 1.2*. Retrieved September 4, 2002, from http://java.sun.com/docs/books/tutorial/security1.2/index.html

Wheeler, D. M., Conyers, A., Luo, J., & Xiong, A. (2001). Java security extensions for a java server in a hostile environment. *Computer Security Conference, 64-73.*

Xingshe, Z., & Xiaodong, Li. (2000). Design and implementation of CORBA security service. *Technology of Object-Oriented Languages and Systems, 140-145.*

Last Updated: August 20th, 2017

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Chicago 2017	Chicago, ILUS	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VAUS	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS San Francisco Fall 2017	San Francisco, CAUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FLUS	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NVUS	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, IE	Sep 11, 2017 - Sep 16, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MDUS	Sep 25, 2017 - Sep 30, 2017	Live Event
Data Breach Summit & Training	Chicago, ILUS	Sep 25, 2017 - Oct 02, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, DK	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, GB	Sep 25, 2017 - Sep 30, 2017	Live Event
Rocky Mountain Fall 2017	Denver, COUS	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, NL	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS DFIR Prague 2017	Prague, CZ	Oct 02, 2017 - Oct 08, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, NO	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS October Singapore 2017	Singapore, SG	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS AUD507 (GSNA) @ Canberra 2017	Canberra, AU	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZUS	Oct 09, 2017 - Oct 14, 2017	Live Event
Secure DevOps Summit & Training	Denver, COUS	Oct 10, 2017 - Oct 17, 2017	Live Event
SANS Tysons Corner Fall 2017	McLean, VAUS	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS Brussels Autumn 2017	Brussels, BE	Oct 16, 2017 - Oct 21, 2017	Live Event
SANS Tokyo Autumn 2017	Tokyo, JP	Oct 16, 2017 - Oct 28, 2017	Live Event
SANS Berlin 2017	Berlin, DE	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS Seattle 2017	Seattle, WAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS San Diego 2017	San Diego, CAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, AE	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Miami 2017	Miami, FLUS	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Amsterdam 2017	Amsterdam, NL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, IT	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Sydney 2017	Sydney, AU	Nov 13, 2017 - Nov 25, 2017	Live Event
Pen Test Hackfest Summit & Training 2017	Bethesda, MDUS	Nov 13, 2017 - Nov 20, 2017	Live Event
SANS Paris November 2017	Paris, FR	Nov 13, 2017 - Nov 18, 2017	Live Event
SANS Adelaide 2017	OnlineAU	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced