Secure Inter-Process Communication

Valentin RADULESCU
IT&C Security Master
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ROMANIA
valentin.radulescu@enea.com

Abstract: This article reveals the necessity in modern distributed systems for authentication of a process running in a distributed system and to provide a secure channel for inter-process communication in which both the client authenticates to the server and the server authenticates to the client. The distributed system is a client-server system based on ENEA LINX inter-process communication framework. Enea LINX is a Linux open source project which allows processes to exchange information between several media channels: shared memory (local process communication), Ethernet (local network inter process communication), TCP/IP (inter process communication through Internet) in which nodes are communicating regardless of the underlying media. Because ENEA LINX offers no security mechanism it appears the need for securing the communicating over LINX protocol. Process authentication disables the need for personal authentication of the user and also prevents an attacker from starting a process which will harm the entire system. Besides authentication, using public key combined with symmetric key technologies the secure inter-process communication system must provide integrity and confidentiality.

Key-Words: inter-process communication, Enea LINX, distributed system, process authentication, confidentiality

1. Introduction

The definition of a distributed system doesn't have a firm meaning and is debatable. A distributed system is a collection of independent computers that appear to the users of the system as a single computer. [1] In this article a distributed system is assumed to be a set of several processors supporting processes which collaborate to accomplish a common mission. As an example take into consideration a banking system. A distributed banking system implies that there are a lot of computers, geographically distributed, which are nodes of the system and which need to exchange information. A local office of such bank system has a network in which these nodes are connected. On each node, there are processes that are running and each process must be authenticated before getting access to the resources of the system. Processes on local nodes must also cooperate in order for the entire system to work properly. Also, the local offices need to communicate with each other over the Internet or, over the banks private network.

Considering this very large distributed system there are a lot of risk factors in such distributed system. Existing distributed systems offer compelling favorable circumstances for the addition of insecure or malicious processes. They also permit hacking and browsing. A specially designed program could propagate itself across the distributed system and could try to access or damage important information. Another security problem consists of systems that are not protected and someone can take advantage and could be used as an entry point to access secure systems. Unprotected systems could be used as an entry point to analyze security faults. Also, this might result in revealing of classified information.

However, distributed systems can improve the overall system security. One advantage of distributed systems is that private, top-secret data can be divided among the entire system and the intruder must know all the locations in order to access the data. Distribution enables the availability of processing resources in case of high system loading or in case of deliberate actions to put off the system.

ISSN 2067 – 4074

From the distributed system description it is obvious the need of a mechanism for communication between the processes that are running in bank's system. In order to allow a group of nodes to communicate over a network, they must all agree on the protocols to be used. [1] From a distributed system point of view an inter-process communication (IPC) implies a set of well-defined rules for exchange of data between one or more processes.

A clear distinction between what inter-process communication in a distributed system means and how inter-process communication at an Operating System level is defined. For example, the Linux operating system provides a set of IPC methods for processes running on top of Linux to communicate with each other [2]. These methods are: half-duplex UNIX pipes, Named Pipes (FIFO), System V IPC (message queues, semaphores and shared memory) and Unix Sockets. A Socket is a bidirectional communication device that can be used to communicate with another process on the same machine or with a process running on other machines [3]. In other words, a Socket allows two processes on different machines to exchange data. Operating systems offer a framework for communication between processes on local system and, also, a foundation to communicate outside the Operating System.

Another IPC mechanism is remote procedure call (RPC) which allows a process to cause a function or procedure to execute in another address space or by another process on the same network without explicitly coding the remote interaction. RPC is a powerful technique for building distributed applications based on client-server model. The RPC model extends the notion of local procedure call, the difference being that the called procedure is not in the same address space as the calling procedure. The two processes involved may be on the same machine or on two machines on the same network. Using RPC, developers can build distributed applications bypassing development of application's networking interface. RPC implies that client process is executing functions on a server and is getting the results back. RPC seem like an easy solution for a big problem but they are high coupled because require a fairly explicit detailing of the messages sent back and forth and the speed is very low because of marshalling, transforming the data in local representation [7].

ENEA LINX [8] is an open source IPC protocol, designed to be platform and interconnect independent. It enables applications to communicate transparently regardless whether they are running on the same machine or are located on same network. The most important advantage of LINX is that it supports any type of cluster configuration, from a single multi-core board to large systems with many nodes interconnected by any network topology [4]. LINX offers a general interface which allows applications to be independent of the hardware and network topology. LINX is available for Linux (2.6 kernels) and is supported by ENEA OSE and OSEck Operating Systems as IPC mechanism.

LINX nodes communicate using signals that are sent to the peer using an identifier of the remote peer. The LINX nodes are unaware of whether the peer is a local process, a process in local network, or a process in the Internet. LINX doesn't provide security and the LINX traffic can be observed by simply accessing the network.

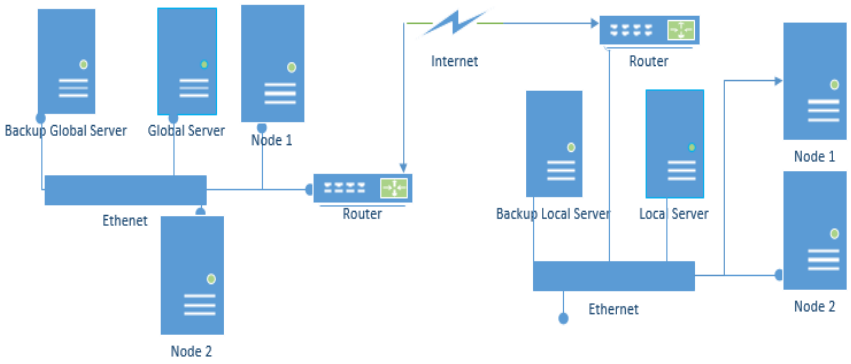

Figure 1. Distributed bank system

2. Problem Formulation

Distributed systems are now everywhere, starting from the telecommunication systems, which now offer access to high speed Internet, to banking systems which are geographically spread and need to assure that global availability. Also, distributed applications are involved in aircraft control systems and in industrial control systems. Interactions with distributed system are everywhere starting from using personal smart phone, or when using a connected car, a car with Internet access and, also, most of the money transactions that we do. The connected cars are using the smart phone model and many applications are available to interact with the car's system. A distributed banking system implies that there are a lot of computers, geographically distributed, which are nodes of the system and which need to exchange information. A local office of such bank system has a network in which these nodes are connected. On each node, there are processes that are running and each process must be authenticated before getting access to the resources of the system.

2.1 Problem definition

Given a bank distributed system, the scope of this article is to design a secure distributed system in which each local network has one Local Server and one Backup Local Server. Each node of the system is connected to the local servers. There is Global Server and a Global Backup Server to which all the Local Servers and Backup Local Servers are connected. Figure 1 describes the structure of the distributed system.

The goal is to create a secure environment in which processes authenticate and to provide confidentiality of the communication channel. Authentication is defined by message content authentication, message origin authentication and general identity authentication. Message content authentication means checking that the composition of the message hasn't been changed. Message origin authentication is defined by allowing the receiver to verify the identity of the sender of a received message. General identity authentication provides both parties to authenticate each other.

The model used in the system is client-server model. The system is controlled by a Global Server to which the nodes connect in order to access classified information or to report events that occurred on that node.

2.2 Process authentication

Processes running on nodes need to authenticate to the server while the server needs to authenticate to the processes doing a mutual authentication. The user login method requires the users to enter their passwords in response to a system prompt. Unfortunately, there isn't any process that allows users to authenticate the system. This is a bigger problem in a distributed system in which users could be fooled into giving the passwords to malware software. It appears the need that both client and server process are mutually authenticated and based on the authentication the client

ISSN 2067 – 4074

processes request services from server processes.
Contrary to the ordinary user authentication using a secret password, in client-server architecture, process authentication demands new and unique security challenges. The user's secret, a password or a passphrase, can be learned by a user. The process' private information must be kept on the host and also needs to be kept hidden from other processes.
The process could be authenticated when the process is created and the authentication state of the process needs to be controlled by the system. Also, when the process demands access to system the status of authentication can be inquired and used for choosing to grant access to resources.

2.3 Credentials

Credentials are used to identify the identity of the process. Unfortunately, when it comes to applications, the executable could be modified or replaced when the host is compromised. To address this issue, the applications need to have fixed file path and a hash of the executable of the application together with the file path will be used as credential. [6]

2.4 Confidentiality

Confidentiality could be defined as prevention of unauthorized disclosure of information. Message confidentiality means hiding the contents of the information. Shared key technologies are the tools which guarantee information security. The same key is used for both encrypting and decrypting the message. The secret key known by both parties must be changed frequently. The best way to accomplish this is to encrypt the shared secret key using public key encryption before exchange.

3. Problem Solution

The processes in the distributed system need to change data. For transferring data from a client process to the server process the Enea LINX inter-process communication protocol is a good solution. It is an open source solution that uses a message based protocol. One advantage of this protocol is that LINX is platform and interconnect independent. Also, LINX scales well to large systems regardless of the topology being used and it also provides the performance that is required for high traffic bearing applications.

3.1 ENEA LINX

LINX is the main message exchange protocol (IPC) used in Enea OSE/OSEck family of real-time operating systems. The implementation of the protocol is different on the Linux machines. LINX for Linux consists of a set of kernel modules and a LINX library is available in user-space to be linked with applications. There is one LINX kernel module that implements IPC mechanism and the Rapid Link Handler (RLNH) protocol which enable LINX to span multiple nodes transparently over logical links. Besides the logical link handler, in order for inter-process communication to work, a Connection Manager (CM) Kernel module that supports the underlying interconnect must be loaded as well. The CMs encapsulate the transport layer. The LINX functionality can be extended to support underlying transport by adding new CMs.
The LINX RLNH implements a logical layer which allows the applications to change messages without needing information about the network topology. From the application's point of view the remote process could be on the same machine, connected through a shared memory CM, or it could be connected through Internet using the TCP CM. The application only needs to find out the binary identifier that is assigned to the LINX endpoint. In order to find out the LINX binary identifier of a remote endpoint the application needs to hunt for the remote process using the name of the LINX link that is connecting the two machines and the remote process LINX name associated.

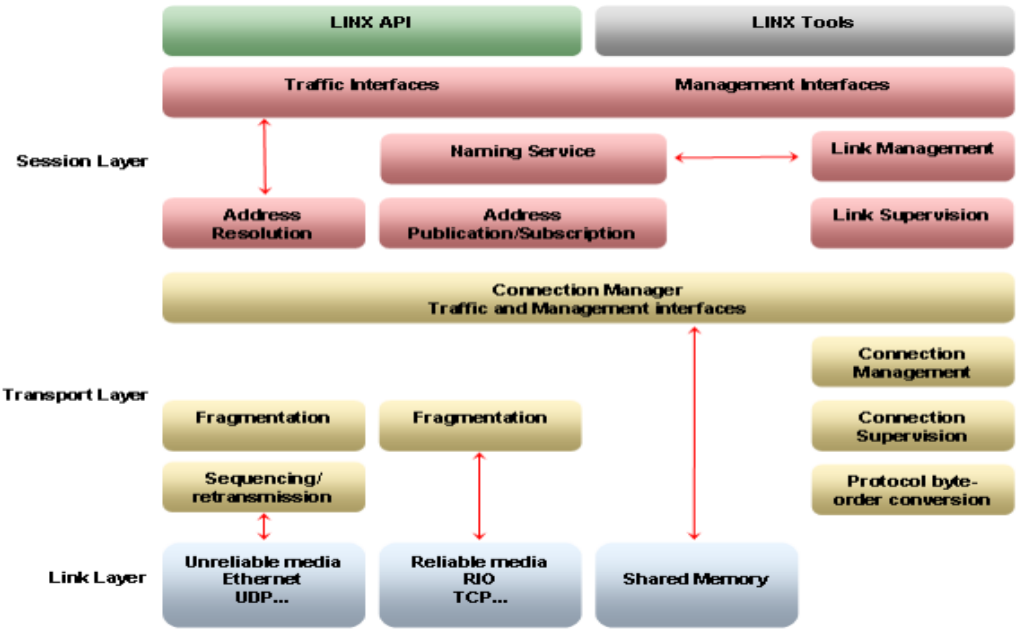

Figure 2. LINX Architecture

Using this combination of names the LINX client can find its way to the server process. Using the binary identifier, the LINX client is able to send and receive message over the LINX link.

3.2 Security considerations

The trusted parts of the system are kernel code and kernel's memory. The system administrator has a lot of responsibility to make sure that when applications are installed the correct steps are fallowed. However, the users of the system are not trusty. It could be that the users are not necessary bad intended but they could access malware software that finds its way in our system.

The attack on model on the system is based on damaging code that could try to access classified information from server. This malware could be retrieved from web page or could be installed by an inside attacker. Using the names of running processes in the system a malware could try to pretend to be one of trusted applications.

After the application is installed, when it is run the first time by the system administrator it must create its identity information which is compose of: Application Name (AppName), Application Path (AppPath), hash of the binary of the application, host name (HostName). Based on this identification information the server creates a record of the application and generates the private secret information that is sent to the application for safe keeping.

If the application terminates or the host is rebooted than the application will need to provide the server the necessary information to get access to system resources.

For the mutual process authentication the Needham-Schroeder public-key protocol will be used. This protocol is described in [5] as follows:

The Needham-Schroeder public-key protocol provides mutual entity authentication and mutual key transport (A and B each transfer a symmetric key to the other). The transported keys may serve both as nonce for entity authentication and secret keys for further use. Combination of the resulting shared keys allows computation of a joint key to which both parties contribute.

Protocol Needham-Schroeder public-key:
SUMMARY: A and B exchange 3 messages.
RESULT: entity authentication, key authentication, and key transport (all mutual).
1. Notation. PX(Y) denotes public-key encryption (e.g., RSA) of data Y using party X's public key; PX(Y1;Y2)denotes the encryption of the concatenation of Y1 andY2. k1, k2 are secret symmetric session keys chosen by A, B, respectively.

ISSN 2067 – 4074

2. One-time setup. Assume A, B possesses each other's authentic public-key. (If this is not the case, but each party has a certificate carrying its own public key, then one additional message is equired for certificate transport.)
3. Protocol messages.
A-->B: PB (k1; A)(1)
A<--B: PA (k1; k2)(2)
A-->B: PB (k2) (3)

Because SecureLinxServer doesn't know in advance all the applications that will be installed, the LINX server will receive the client's certificate CliAuthPubKey after the application is installed. The CliAuthPubKey is delivered to the server encrypted with ServerPubKey. To increase the security of this approach a hash over the entire message will be calculated and encrypted with the public key.

3.3 Application's Credentials

An application's credential is private and unique information that is generated by the server to a trusted application so that application can demonstrate its identity when authentication to the server. This credential must uniquely identify the application and must be kept hidden from other processes running in the system.
The server must generate an application's credential based on a request done by the system administrator when installing the application on a node in a system. The server application must keep a list with the applications that are installed in the entire system and this list must be updated whenever a new trusted application is installed and, also, when uninstalling an application.

3.4 System components

Next are the components of the application:

- *LibSecureLinx* - is a security module over LINX protocol which offers basic LINX API with the addition of message signing, encryption and decryption of the messages that are exchanged over the LINX link. The application will be a library that will be used by all other applications. For the security protocols implementation the OpenSSL library will be used. OpenSSL is an open-source implementation of SSL and TLS and is widely used by highly secure applications;
- *SecureLinxServer* is the main server application. The purpose of this application is to register client processes. The system administrator registers the password when installing this application and he is responsible for installing client processes;
- *LinxChat* is the application which wants to authenticate its processes to the server. After authenticating to the server the client application will exchange data with the server through a secure LINX channel;
- *LinxAuthenticationTool* (LAT) is the tool which will be used when first installing a client application in the system. The LAT requires System Administrator password and the path to the client application. Running the LAT will cause the client application to generate the client's private key and public key. The certificate composed of public key will be sent to the server and the server maintains the application process information. The LAT will be used only once, after the client application was installed.

3.5 Authentication procedure

Let's take the example of a chat application called LINXChat which must be authenticated to the SecureLinxServer running on a Local Server machine in the local network. The LINXChat application is installed by the system administrator on Node1. To ensure that the installation worked fine and that the LINXChat application is a trusted application the system administrator needs another application which is called LINXAuthenticationTool.
When the SecureLinxServer is installed a certificate is generated by the SecureLinxServer containing (ServerPrivKey, ServerPubKeyr). The private key is kept secret, while the public key is made available to the LINXAuthenticationTool.

Authentication on installation

System administrator installs LINXChat application on Node1. Then it runs the LINXAuthenticationTool with

ServerPubKey and the path where LINXChat is installed. The LinxAuthenticationTool(LAT) requires system administrator's password. The authentication tool lets the client generates the authentication certificate CliAuthPubKey, CliAuthPrivKey. The CliAuthPrivKey is the secret information that will allow the client to authenticate to the server.

- LAT-->LINXServer: The message begins with a hash of system administrator's password. The authentication application generates the LINXChat applications identity name: LINXChat, application path: /opt/install/LINXChat/chat, it computes the hash over the binary of the application, host name: Node1 and it also generates a random number, rand1 and it appends the CliAuthPubKey to the message. A hash over the entire message information is generated and appended to the message and this entire message is encrypted with the ServerPubKey.
- SecureLinxServer-->LAT: Using the PrivKeyServer, SecureLinxServer decrypts the message. First, it compares the hash of the password with the stored password and if there is a match the procedure continues, otherwise it fails. After extracting the necessary information it computes the hash over the message and if the received hash is different the server detects the error and the authentication fails. Server stores the hash of the binary and it will be used later on when the process will try to authenticate to the server. Server prepares the process data. The public key, CliAuthPubKey, is associated with the received information and binary hash. SecureLinxServer creates a new message which starts with received rand1. It generates a new random, rand2 and adds it to the message. Next a hash over the entire message is computed. The message is encrypted with CliAuthPubKey.
- LAT-->SecureLinxServer: Using CliAuthPrivKey the received message is decrypted by the client. First the client compares received rand1 with the one generated at startup and continues on success. Client computes the hash over received message and continues on success. Client generates a new message which contains rand2, a hash over rand2 encrypted with CliAuthPrivCli.

When LINX server received the last message it compares rand2 with generated rand2 and the new client process data is added as trusted application.

Authentication on reboot or restart

- LINXChat-->SecureLinxServer: The message starts with new random rand1. The application identity information is added to the message. A hash over the message is computed and added to the message. The entire message is encrypted with ServerPubKey.
- SecureLinxServer-->LINXChat: The server decrypts the message and verifies the hash. It computes a new random rand2 and it creates a message containing rand1, rand2 and a hash over them. This message is encrypted using clients stored public key CliAuthPubKey.
- LINXChat-->SecureLinxServer: Client decrypts the message and verifies hash and rand1. Client computes a new message with rand2, a hash over rand2 encrypted with server's public key.
- When the SecureLinxServer verifies hash and rand 2 the two parties are authenticated.

3.6 Confidentiality

The confidentiality of the communication is done using a session key which is computed using a function *f =func(rand1,rand2)* which both parties compute after authentication. Using a session key helps protect against attack

ISSN 2067 – 4074

which use encrypted material for finding the key. Using rand1 and rand2 for computing session keys it will be very hard for an attacker to predict them.

4. Conclusion

The banking system that was chosen as a model shows the importance of IPC mechanism in a distributed system. The security risks that are generated by such type of system were analyzed and the distributed system offers gates for malware software to access confidential information. Also, the risk of a bad intentioned user which introduces malicious software in the system was considered.

The ENEA LINX is a good choice because it is independent of the media on which the traffic is exchanged and the processes communicate using the LINX links without managing the underlying connection. The solution presented here is a middleware set of applications which permit the system administrator to install only trusted processes in the system. The model chosen is a client-server model but the current design allows building extending to a peer to peer model. Also, the solution offer confidentiality of the information that is exchanged by the trusted processes. The applications rely on public key technologies to assure mutual authentication between the client and server. Also, symmetric key is used for offering confidentiality while hashing provides the authenticity of the data.

The presented solution has same drawbacks also. The random that is used to ensure protection against man-in-the-middle attacks and replay attacks is software generated. Using a Hardware Security Module increases the security of the solution. HSM provide both logical and physical protection of keys used. Also, existing solutions need to be updated to support the LINX protocol. Also, the way the process keeps the private key may also present a problem to deny other processes access to the key.

Acknowledgment

Parts of this research have been published in the Proceedings of the 6th International Conference on Security for Information Technology and Communications, SECITC 2013.

References

[1] A. S. Tanenbaum, M. van Steen, *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2006

[2] Sven Goldt, *The Linux Programmer's Guide*, 1995, Availabe: http://www.tldp.org/LDP/lpg/node1.html

[3] M. Mitchell, J. Oldham, A. Samuel, *Advanced Linux Programming*, New Riders Publishing, 2001

[4] Enea Software AB, *LINX for Linux User's Guide*, online, Available: http://linx.sourceforge.net/linxdoc/doc/usersguide/UsersGuide_LINX_for_Linux.html

[5] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1965

[6] Hussain M.J. Almohri, Danfeng (Daphne) Yao, Dennis Kafura, Process Authentication for High System Assurance,

[7] E. Hopper, *Why CORA and other forms of RPC are bad*, online, http://www.omnifarious.org/~hopper/corbabad.html

[8] LINX project, online, Available: http://sourceforge.net/projects/linx/