A Security Framework for JXTA-Overlay

Joan Arnedo-Moreno
Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya (UOC)
Rambla de Poblenou, 156 08018 Barcelona, Spain
Email: jarnedo@uoc.edu

Keita Matsuo
Graduate School of Engineering
Fukuoka Institute of Technology (FIT)
3-30-1 Wajiro-Higashi, Higashi-Ku, 811-0295 Fukuoka, Japan
Email: bd07002@bene.fit.ac.jp

Leonard Barolli
Department of Information and Communication Engineering
Fukuoka Institute of Technology (FIT)
3-30-1 Wajiro-Higashi, Higashi-Ku, 811-0295 Fukuoka, Japan
Email: barolli@bene.fit.ac.jp

Fatos Xhafa
Department of Languages and Informatics Systems
Technical University of Catalonia (UPC)
Jordi Girona 1-3, 08034 Barcelona, Spain
Email: fatos@lsi.upc.edu

Abstract

At present time, the maturity of P2P research field has pushed through new problems such us those related with security. For that reason, security starts to become one of the key issues when evaluating a P2P system and it is important to provide security mechanisms to P2P systems. The JXTA-Overlay project is an effort to use JXTA technology to provide a generic set of functionalities that can be used by developers to deploy P2P applications. However, since its design focused on issues such as scalability or overall performance, it did not take security into account. This work proposes a security framework specifically suited to JXTA-Overlay's idiosyncrasies.[1]

Keywords: peer-to-peer, security, XMLdsig, JXTA, JXTA-Overlay.

1. Introduction

Peer-to-peer (P2P) have become highly popular in recent times due to its great potential to scale and the lack of a central point of failure. Just as the popularity of P2P systems has risen, so has concerns regarding their security, specially since it is no longer possible to trust a central server which capitalizes all security operations. As P2P applications move from simple data sharing to a broader spectrum, they become more and more sensitive to security threats and it becomes very important for current P2P platforms to include security mechanisms that can fit into a broad set of scenarios. Even at the cost of some impact on performance, a security baseline must be kept in any P2P system in order to ensure some degree of correctness even when some system components will not act properly.

1. This work is partially supported by the Spanish Ministry of Science and Innovation and the FEDER funds under the grants TSI2007-65406-C03-03 E-AEGIS and CONSOLIDER CSD2007-00004 ARES.

JXTA [1] (or "juxtapose") is a set of open protocols that enable the creation and deployment of peer-to-peer networks. JXTA protocols enable peer-to-peer applications to discover and observe peers, enable communication between them or offer and localize resources within the network. Such protocols are generic enough so they are not bound to a narrow application scope, but are adaptable to a large set of application types. For that reason, they also keep implementation independence, so they can be deployed under any programming language or set of transport protocols.

JXTA-Overlay [2] is a JXTA-based framework. Its main goal is to improve the original JXTA protocols, increasing the reliability of JXTA-based distributed applications and supporting group management and file sharing. However, the design focus on JXTA-Overlay was completely concerned with system performance, but not at all with security, a situation which may become a great constraint under today's standards. Even though JXTA provides some basic security mechanisms, they were not taken into account.

The contribution of this paper is a modular security framework specifically suited to the characteristics of JXTA-Overlay. The proposed framework fully realizes the messaging capabilities and functions of both JXTA and JXTA-Overlay and uses them in order to provide a security baseline in a transparent manner. As a result, minimum effort is necessary by application developers and end-users to deploy a secure environment. Furthermore, because of the framework's modular approach, it may be easily ported to different scenarios, according to the final application's needs.

This paper is organized as follows. Section 2 provides a general overview of JXTA and JXTA-Overlay's architecture and functions, which is necessary to fully understand which are its most important parts and which constraints exist when specifying its security framework. This section also provides some insights on the current state of security of JXTA-Overlay and currently security vulnerabilities. Section

3 presents the current related work on securing JXTA-based systems. The proposal of a basic security framework is presented in section 4. Concluding the paper, section 5 summarizes the paper contributions and further work.

2. JXTA-Overlay Overview

JXTA-Overlay is a middleware built on top of the JXTA specification [3], which defines a set of protocols that standardizes how different devices may communicate and collaborate among them. JXTA-Overlay extends JXTA protocols with the goal of overcoming some of its limitations: the need for the developer to manage the presence mechanism, peer group publication and message exchange. To achieve this end, JXTA-Overlay provides a set of basic functionalities, *primitives*, intended to be as complete as possible to satisfy the needs of most JXTA-based applications.

2.1. The JXTA-Overlay network

In a JXTA-Overlay network, the main interacting entities are:

End-users connect to the JXTA-Overlay network by authenticating using a username and password. Once the authentication process is successfully completed, they are organized into different overlapping groups, so only members of the same group may interact. It is also important to take in to account that JXTA-Overlay end-users are mobile, they may connect at different times using different client peers, as well as may be connected through several client peers at the same time.

A *client peer* represents an application, which end-users use to communicate and share resources between themselves, effectively acting as end-user proxies within the JXTA-Overlay network. They forward end-user data to client peers that belong to end-users of the same group and authentication data to a broker. A client peer is assumed to belong to the same groups as its current end-user.

Brokers control access to the network, requesting end-user authentication, and help client peers interact between themselves by propagating their related information. Brokers are very important since they exchange information about all client peers, maintaining a global index of available resources, thus allowing all peers to find network services. Brokers also act as beacons which client peers which have recently gone online use to join the network. For that reason, they usually have well-known identifiers, such as a DNS name or a static IP address.

All the information related to user configuration (username, password and group membership) is stored in a special single entity within the JXTA-Overlay network: a *central database*. Only brokers may access the database data, in order to check end-user authentication attempts and organize them into groups. It is assumed that some *administrator* takes care of properly configuring the database, registering new end-users. Nevertheless, JXTA-Overlay does not impose any constraint on the database architecture.

2.2. General architecture

First of all, it must be remarked that JXTA-Overlay does not provide any full client peer, apart from some demo applications. Only brokers are provided as a fully developed application which may be directly deployed, without the need for any kind of additional development. The architecture of the JXTA-Overlay middleware defines three modules, which let the different entities described in section 2.1 communicate: the Client Module, the Broker Module and the Control Module. Altogether, they form an abstraction layer on top of JXTA, as shown in Figure 1.

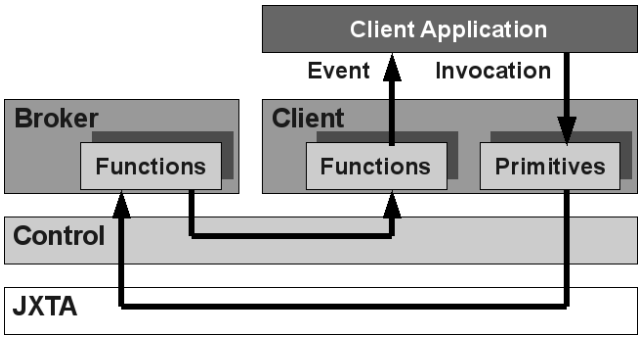

Figure 1. JXTA-Overlay architecture.

- The *Client Module* defines all necessary primitives for peer clients to join a JXTA-Overlay network and interact with other peers and the broker. In fact, applications developed on top of JXTA-Overlay are always based on the invocation of Client Module primitives defined and the processing of events thrown upon primitive execution or a broker response. Primitives are comprised for: (a) peer discovery; (b) peer resources discovery; (c) resource allocation, (d) task submission and execution; (e) file/data sharing, discovery and transmission; (f) instant communication; (g) peer group functionalities and, monitoring of peers, groups, and tasks.
- The *Broker Module* defines all the functions that client peers may call upon a broker in order to be granted access to the the network, create and publish groups or retrieve other client peers' information. Functions always produce a reply from the broker to the calling client peer. Broker functions are always called as a result of Client Module primitives.
- The *Control Module* acts as an intermediate layer between the Broker and Client Modules, providing the generic functionalities on regards to group management and messaging.

The Control Module provides messaging between JXTA-Overlay entities using JXTA *pipes*, a virtual communication channel between peers. Client peers have an input pipe for each group it belongs to, so other group members may send messages using the input pipe associated to that group, as shown in Figure 2. Brokers have a single input pipe which is shared for all incoming messaging.

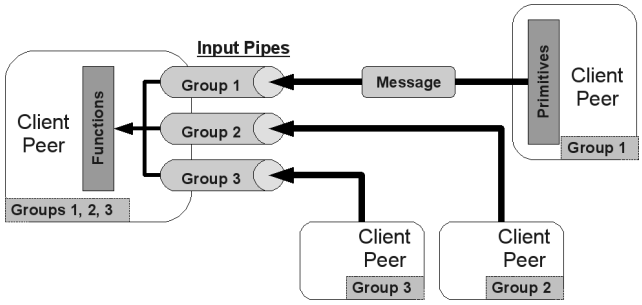

Figure 2. Client peer messaging via input pipes.

2.3. Client peer information propagation

Brokers propagate information between group members, crossing boundaries such as client peers located beyond broadcast range. Such information is formatted as JXTA *advertisements*, metadata documents used to distribute information between peers. All advertisements are codified using XML and passed between client peers using the JXTA core protocols. JXTA-Overlay has a big reliance on both JXTA-defined and custom-defined advertisements to propagate information across client peers via the broker. As a result, their data is critical for the correct operation of the JXTA-Overlay network. Each client peer periodically broadcasts the following advertisement types for each group its end-user belongs to:

- **Peer Advertisement:** Acts a presence beacon for other client peers. While peer advertisements are being received from some peer, it is considered to be online.
- **Peer Group Advertisement:** Announces existing groups. They are necessary in order to properly join any group and interact with its members.
- **Pipe Advertisement:** Transmits the client peer's input pipe location for a particular group. Message exchange between client peers or brokers is not possible without each other's input pipe. Therefore, it is one of the most important advertisements in JXTA-Overlay.
- **Overlay Advertisement:** This is a JXTA-Overlay specific advertisement that provides miscellaneous information to other client peers. A type field defines the specific data transmitted:
  - *Info Advertisement:* General information about a client peer.
  - *Files Advertisement:* List of shared files.
  - *Statistics Advertisement:* Client peer statistics (transmitted data, uptime, etc.)
  - *Criteriums Advertisement:* Client peer capabilities (CPU and connection speed, maximum file size, etc.)

2.4. JXTA-Overlay and security

As previously exposed, JXTA-Overlay's design is not concerned with security, with the only exception of end-user network access control via a username and password. As a result, its is vulnerable to different security threats which may jeopardize the network. A security study must take into consideration the fact that not only entities external to the JXTA-Overlay network may try to subvert it, but also malicious legitimate users.

Some of the greatest security concerns in JXTA-Overlay are the following ones:

- Transmitted data may be easily eavesdropped, since no data privacy is provided. Even though it may be argued that data privacy in message exchanges between end-users is just an optional feature, there are some cases where privacy should not be optional, namely the initial authentication username and password. Currently, both fields are sent as plain text. Therefore, any device at broadcast range may read this information with a network protocol analyzer (such as Wireshark [4]).
- Any legitimate user may forge advertisements with no fear of reprisal. No integrity or source authenticity is maintained. False fields, such as the source client peer identifier or any statistics information, may be added into the advertisement, which will be automatically distributed by the broker and accepted by all group members, unaware of the false data it contains.
- Client peers connect to a self-proclaimed broker, but never check if it is a legitimate one. Even in the case that client peers are connecting to the proper broker address, there are no guarantees that the broker is a legitimate one, since it may be that traffic is being redirected to a fake one via methods such as DNS spoofing [5].

As can be seen, some of the current JXTA-Overlay vulnerabilities are quite obvious ones, such as transmitting sensitive data with no real privacy. Therefore, it can be concluded that JXTA-Overlay does not provide a security baseline and needs serious improvement on this regard.

3. Related Work on Securing JXTA-based Frameworks

Before a security framework for JXTA-Overlay may be proposed, it is useful to review which are the current security

mechanisms available to JXTA-based applications. From this review, it is possible to study which may prove usefull or suitable to JXTA-Overlay's architecture and network setup specifics. In this section, we provide a general overview on security in JXTA applications, but a much more complete survey may be found in [6].

As far as network access control is concerned, one of the original creators of JXTA, Yeager, provides a specific trust model in [7]. Without actually recognizing a specific Certification Authority (CA) for each peer group, he proposes that rendezvous peers become the system's trust anchors, providing credentials to peers, that can be used to prove network membership. To acquire a credential the peer must be authorized via an LDAP (Lightweight Directory Access Protocol) [8] directory with a recognized protected password. Rendezvous peers may use a secure connection to the LDAP service to authorize requesting peers.

Yeager's proposal is extended in [9], with a similar trust model, but adding extra capabilities. This approach is based on a centralized Public Key Infrastructure (PKI) and a basic challenge-response protocol [10] as a means for authentication during the join process. Its main contribution is to provide a method which peers may use in order to authenticate the group itself.

More elaborated proposals are presented in [11], [12], based on joint authorization by multiple peers under voting schemes in order to maximize decentralization. Under these approaches, credentials are also signed certificates issued by a CA, however access is based on an agreement reached between several group members. The main difference between both proposals is that [12] includes a rank system, where peers who join the group ("newbies") have the least privileges, but they may rise to higher positions as they contribute to the group.

On regards to message security, the JXTA reference implementation [13] provides two mechanisms: TLS [14] (Transport Layer Security) and CBJX [15] (Crypto-Based JXTA Transfer). The former provides private, mutually authenticated, reliable streaming communications, whereas the latter provides lightweight secure message source verification (but not privacy).

JXTA provides its own definition of standard TLS as a transport protocol. The JXTA definition of TLS is composed of two subprotocols: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol provides connection security using symmetric cryptography for data encryption. The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by the TLS Handshake Protocol. In addition,the connection is reliable by including message integrity check using a keyed MAC.

On the other hand, CBJX is a JXTA-specific security layer which pre-processes messages to provide an additional secure encapsulation, creating a new message that is then relayed to an underlying transport protocol. The original message's is signed, and an additional information block, is also added to the secured message. This information block contains the source peer credential, both the source and destination addresses, and the source peer ID.

In order to use both mechanisms, TLS and CBJX, a specific group membership service is required: the Personal Security Environment (PSE). The membership service is one of the JXTA core services, taking care of group membership and identity management by providing each group member with a credential. Peers may include credentials in messages exchanged within a group in order to prove membership and provide a means for implementing access control in offered services. However, PSE solely supports on X509 certificates [16] as credentials and Java keystores [17] as a cryptographic module.

It is also possible to provide some degree of advertisement security in the current JXTA reference implementation by signing advertisements. No distinction between different types of advertisements is made, all become a new type of advertisement when signed: the Signed Advertisement. A Signed Advertisement encapsulates the original XML advertisement as plain text encoded via the Base64 algorithm [18]. Signed advertisements are also contrained to the PSE membership service.

An alternative method to secure any advertisement type is proposed in [19]. This method is based on XMLdsig [20] and can be applied to those advertisement types defined by JXTA as well as those custom made by JXTA-based applications. The resulting secure advertisement maintains its original type, instead of becoming a completely different new type of advertisement.

4. A Security Framework for JXTA-Overlay

In this section we present a secure framework for JXTA-Overlay which provides a baseline for protecting end-user applications against the current vulnerabilities exposed in section 2.4. In our proposal, we combine several methods of those previously described in section 3, adapting them to JXTA-Overlay's specific architecture and network setup. Client peers are protected against impersonation by using broker-issued credentials in a similar way to the approach in [7]. However, we further extend this approach to the broker, so a legitimate one may be told apart from malicious ones, and also provide an alternate lightweight method for message source authenticity. Furthermore, advertisement integrity and authenticity, as well as a transparent method for key transport is provided by adapting the method defined in [19]. Finally, data privacy is used to protect message exchanges against eavesdroppers. From this framework, security capabilities may be easily added to JXTA-Overlay.

The main goal of this framework is operating in the most transparent way for a JXTA-Overlay end-user application

developer, providing a complete abstraction layer.

4.1. System setup

In order to deploy a secure framework, JXTA-Overlay entities must be provided some cryptographic data beforehand, necessary to execute cryptographic operations on secure services. Such data is provided at three different stages: at deployment, boot and login. The first case includes data that is generated only once during the whole system life, when entities are deployed into different physical nodes. On the other hand, the second case comprises data which is generated each time an entity goes online, just before it joins the network. The last case regards cryptographic data provided when the end-user authenticates to the JXTA-Overlay network via a broker.

All cryptographic data should be securely stored into to some cryptographic module (e.g. a keystore, smartcard, Hardware Security Module (HSM), etc.) [21]. However, our framework has no constraints on regards to which type of cryptographic module can be used. Client peer developers may freely chose the one that suits its own needs.

4.1.1. Deployment cryptographic data. Whenever a new JXTA-Overlay network is deployed, the administrator generates a public key PK_{Adm} and secret key SK_{Adm}. From both keys, also generates a self-signed credential, $Cred_{Adm}^{Adm}$, thus acting as trusted party by all peers. This is a sensible stance, since, nevertheless, the system administrator is the one who that grants access to the JXTA-Overlay network by creating legitimate usernames and passwords into the database, having absolute control on end-users nevertheless.

Each broker, Br_i, is provided a well-known identifier ID_{BR_i} by the administrator (in fact, JXTA-Overlay already does this: a DNS name or static IP address), which will be the one client peers use to connect to it. Br_i also generates a public and secret key, PK_{Br_i} and SK_{Br_i}. From PK_{Br_i}, the administrator will provide Br_i with a credential $Cred_{Br_i}^{Adm}$, by signing PK_{Br_i} and ID_{BR_i} with SK_{Adm}. Therefore, only legitimate brokers will hold a proper credential and be able to prove its ownership. Such credential is transmitted using an out-of-band method.

Each client peer, Cl_i, who wants to use the JXTA-Overlay network is provided with a copy of $Cred_{Adm}^{Adm}$.

4.1.2. Boot cryptographic data. Only client peers generate additional cryptographic data at boot time, immediately before going online. Namely, they generate they key pair PK_{Cl_i} and SK_{Cl_i} at this precise moment. The main reason for such keys not enduring the whole node's life, in contrast with brokers, is the fact that end-users are mobile, and therefore, different end-users may use the same exact node at different stages during the client peer's lifecycle. In that case, it means end-users would also use the same key pair. Requiring the end-user to transport and manage the key pair between nodes quickly becomes a hassle as well as a constraint for application development.

Once the key pair has been generated, the client peer identifier is set as a Crypto Based IDentifier (CBID), further described in section 4.2.1. At this point, it is sufficient to say that it is a method to bind the identifier to PK_{Cl_i}.

4.1.3. Login cryptographic data. Each time an end-user logins into the network, the broker issues a temporary credential, $Cred_{User}^{Br_i}$, containing the client peer's PK_{Cl_i} and his username. An end-user connecting to the network via several client peer's will be provided several credentials, each one assigned and managed by each specific client peer. Credentials are only valid for a single end-user's session. This accounts for the fact that, since end-users are mobile, PK_{Cl_i} may be a different one each time.

4.2. Key distribution and advertisement security

Once each broker and client peer has established its own public key, it must be distributed to the rest of group members, so it is effectively possible to secure data exchanges. Data privacy may be obtained using encryption and source authenticity and integrity by digitally signing messages, for example, using the RSA algorithm [22] for both cases. The key distribution method must take into account the fact that in a P2P network nodes may go online and offline at any moment, as well as allowing key updates in the case that, for some reason, a new key must be generated.

To achieve this end, we apply the scheme defined in [19], based on XMLdsig, where the public key is included into an advertisement, relying on JXTA-Overlay's standard information propagation mechanism, as shown in section 2.3, for key distribution. The credentials generated at the different stages of system setup, as shown in section 4.1, are used, not the raw public key. However, Pipe Advertisements are used, instead of Peer Advertisements, as proposed in the base scheme. The reasons for this choice are twofold.

First of all, all JXTA-Overlay's messaging capabilities between group members rely on the input pipes, as explained in section 2.2, which can only be accessed by previously retrieving its associated Pipe Advertisement. Therefore, client peers cannot exchange messages unless they have each other's Pipe Advertisement. Consequently, by publishing keys using this advertisement type, it is also always guaranteed that both parties have each other's public key before any message exchange begins. Furthermore, it avoids relying on additional protocols for key distribution.

Additionally, the scheme allows the signature of Pipe Advertisements, providing effective protection against advertisement forgery, as exposed in section 2.4. It must be heavily remarked that the importance of each client peer's JXTA input pipe in JXTA-Overlay. Once a broker

has granted access to a client peer, absolutely all incoming messages are received via this pipe. Therefore, it is very important to secure the distribution of each client peer's Pipe Advertisement to avoid that a rogue peer may publish a forged advertisement, claiming that its own input pipe is assigned to some other Peer ID. In that scenario, all messages outbound to that Peer ID would be automatically redirected to the rogue peer.

A crucial advantage offered by this XMLdsig-based scheme, instead of JXTA's Signed Advertisement, is its capability to become invisible to standard JXTA-Overlay operation, instead of adding a new advertisement type, completely opaque to advertisement indexing and retrieval services (all advertisement fields disappear). As far as Pipe Advertisements are concerned, they are structured into several fields. The *Id* and *Type* are mandatory fields, the former to defining the advertisement unique identifier, and the latter specifying the message transport pipe type, which in the case of JXTA-Overlay is always unicast. The *Name* and *Desc* are optional fields. However, JXTA-Overlay always makes use of them in order to define which client peer is the input pipe owner, by including the client peer identifier in the *Name* field, and which end-user is connected through that client peer, by including the username into the *Desc* field. In this manner, JXTA-Overlay may easily search for some peer of user's Pipe Advertisement and send messages to the associated pipe. By signing Pipe Advertisements, its fields cannot be tampered. Therefore, it is very important that both fields are kept visible to JXTA-Overlay.

4.2.1. Key authenticity. Key distribution must always guarantee public key authenticity, a method so that anyone may check whether an endpoint is the legitimate owner of some claimed public key. However, first of all, it must be taken into account that in the particular case of JXTA-Overlay, messaging may occur between two different endpoint types: client peers, for control messaging such as advertisement propagation, and end-clients, for direct communication, such as chatting or file exchange. Hence, key ownership must consider both types of endpoint data exchanges. It must also take into account the fact that an end-user may be connected to the JXTA-Overlay network using several client peers at the same time.

We accommodate to these constraints by using a joint scheme were a different mechanism is used depending on the endpoint type. On one hand, since exchanges between client peer endpoints are very frequent, a lightweight method is desirable. Thus, CBIDs are used. On the other hand, exchanges between end-users require some method that provides additional information that may be presented to the user, such as the source username and an easily recognizable identifier of a trusted entity, so he may decide to accept or not the exchange. Thus, the signed credentials are used in this case. The schematics of this key authenticity method are shown in figure 3.

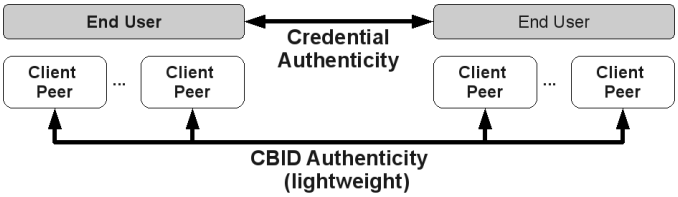

Figure 3. JXTA-Overlay key authenticity model.

The concept of CBIDs, or statistically unique and cryptographically verifiable IDs (SUCV IDs), was initially conceived for IPv6 addressing in order to solve the issue of address ownership in a lightweight manner [23]. We adapt this secheme to our security framework by applying it to the client peer identifier. The client peer public key PK_{Cl_i} is bound to the client peer identifier by applying a pseudo-random function on the public key. The result is henceforth used as the JXTA peer identifier. Secure messages between client peer endpoints are signed using SK_{CL_i}. In order to validate CBID ownership, the message's signature is validated. If validation is correct, its is proved that the source peer holds the associated private key. Then, the validating public key is used to generate the source peer identifier, *i.e.* the CBID. If the obtained identifier is the same as the claimed one, the message is authentic. This method cannot be used for end-user endpoint data exchanges, since it is not possible generate a CBID that conforms to his username, which cannot be changed.

Key authenticity via credentials is checked by verifying the credential's signature against the issuer's public key, validating the full certificate path: end-user, broker, administrator (being the latter the trusted party by all entities in the system). The brokers' credentials are available to client peers, since they are also distributed with the Pipe Advertisements. The administrator's single credential is provided at deployment, as explained in section 4.1.

4.3. Secure primitives and functions

Once cryptographic data is provided to peers and propagated across the network, it is possible to provide a set of secure primitives and functions. In this proposal, only some of the most basic, but not least important, primitives have been secured. The extension of the JXTA-Overlay framework to secure every single primitive is beyond the space limitations of this work (about 122 primitives and 84 events). However, once the building blocks for a secure system have been established, an integral key distribution and authenticity scheme, it is feasible to extend security to every single primitive.

Two very important primitives are the ones related to discovery which search for a broker and allow authenti-

cation, sending the username and password, in order to join the network: the *connect* and *login* primitives. The functionalities of these primitives have been expanded by creating two new secure versions which allow to properly setup client peers as described in section 4.1 as well as offering protection against those threats discussed in section 2.4:

- *authBroker*: Locates a Broker Br_i and waits for a connection to open. Then, authenticates the broker requesting $Cred_{Br_i}^{Adm}$ and checking whether it is a proper one verifying its signature against $Cred_{Adm}^{Adm}$, provided to all client peers at deployment, and challenge-response protocol [10]. If authentication succeeds, Br_i is a legitimate broker and login may proceed.
- *secureLogin*: The end-user's username and password are sent encrypted to the broker, using the public key enclosed in $Cred_{Br_i}^{Adm}$. Furthermore, a credential request, containing PK_{Cl_i} is sent to Br_i. If username/password authentication succeeds, Br_i responds with $Cred_{User}^{Br_i}$ and proceeds with the JXTA-Overlay standard group initialization procedures.

Basic messaging is included into the set of instant communication primitives. This set has been expanded to include two new secure versions of former primitives, so some degree of privacy and data integrity is provided:

- *sendSecureMsgPeer*: Send a simple text message to some other end-user. However, data is previously signed using PK_{Cl_i} and encrypted using the destination client peer's public key. Since message exchange endpoints are end-users, key authenticity at destination is processed via credential validation as explained in section 4.2.1.
- *sendsecureMsgPeerGroup*: Sends a simple message to all members of a group. It is actually resolved by iteratively calling *sendSecureMsgPeer*.

4.4. Secure architecture

An overview of security framework proposal JXTA-Overlay is presented in Figure 4. All modules are located at different layers within JXTA-Overlay and JXTA's own architecture.

The following modules are not JXTA-Overlay specific, but an extension of JXTA protocols. As such, they may be used in any JXTA-based application:

- The *Crypto Manager* provides an abstraction layer for any cryptographic module and key management. It enables the integration of security services with any kind of module such as hardware cryptographic tokens. This approach takes into account the fact that not all such modules are accessed via plain text passwords (for example, some use may biometrics). Furthermore, even in cases were passwords are used, sometimes, each module has its methods for accessing private keys (such as the special GUI integrated into the operating system in the the Windows CryptoAPI [24]). The Crypto Manager is completely modular and accepts different implementations, according to the needs of the specific cryptographic module being used for any particular deployment of JXTA-based application. This is in contrast with PSE, as exposed in section 3, which is constrained to a single type of cryptographic modules, java keystores.
- The *Secure Adv* defines the secure advertisement format, providing a method for key transport and additional fields related to its signature, as defined in [19].
- The *Adv Signer* manages JXTA advertisement signature and validation by interacting with the Crypto Manager. Secure advertisement management is provided in a modular way to both JXTA and JXTA-Overlay without the need to modify the standard JXTA libraries.

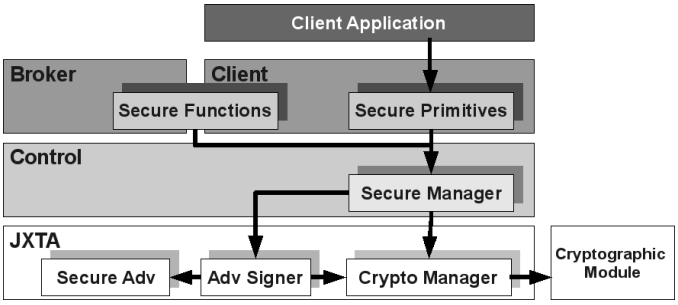

Figure 4. Security architecture for JXTA-Overlay.

The JXTA-Overlay specific modules follow:

- The *Secure Manager* is the common interface to all additional security capabilities within the JXTA-Overlay Control Module, operating as a single entry point for all secure services. This module provides and initializes the Crypto Manager implementation for each instance of JXTA-Overlay.
- Finally, the *Secure Functions* and *Primitives* Modules just extend the base Broker and Client Modules, discussed in subsection 2.2, providing a set of additional primitives and functions which take into account security considerations. As far as secure framework management is concerned, the end-user application developer just has to choose which primitives to use, just as in standard JXTA-Overlay. The secure version of JXTA-Overlay's primitives do not replace JXTA-Overlay's original non-secure versions, but just complement them, leaving the final choice on which primitives to use to the end-user application developer.

5. Conclusions

A security framework proposal for JXTA-Overlay been presented. Apart from providing a baseline for the deployment of security mechanisms in JXTA-Overlay, which up to now had not been considered into its design, the main contributions of the chosen approach are threefold.

First of all, an effective framework for secure key distribution is provided, by securing pipe advertisements and using standard JXTA-Overlay procedures for key publication and update, guaranteeing that keys are always available whenever messages must be exchanged between peers. As a result key distribution becomes invisible to the Control Module and both client peers which use the secure framework and those who don't may coexist in the same JXTA-Overlay network, in contrast with JXTA's current approaches.

Second, key authenticity is provided by a combination of CBIDs and signed credentials. The net results of this approach is that, on one hand, those cases where no end-user intervention is necessary can be resolved in a lightweight manner using CBID's. However, on the other hand, it is also possible to provide meaningful information to the end-user in those circumstances where its intervention may be necessary or helpful, by providing a user credential.

Finally, the proposed framework is completely modular and can be adapted to different scenarios (different types of credentials or cryptographic modules) suitable to the application developers' needs. This is also an improvement over the security mechanisms provided by JXTA, which tie end-user applications to a very specific credential and cryptographic module type. In our implementation, RSA public/private keys are used, and credentials are issued in the form of X509 certificates, but the system accepts any keystore and credential type in a modular way via different CryptoManager implementations.

Further work includes using the proposed security framework to define secure primitives for those interactions which are deemed sensitive to attacks, in a manner that they complement existing ones, but not forcibly replace them. Of special note are those of the executable set of primitives, related to remote code execution.

References

[1] SUN Microsystems, "Project JXTA", 2001, http://www.jxta.org.

[2] F. Xhafa, R. Fernandez, T. Daradoumis, L. Barolli, and S. Caballe, "Improvement of JXTA protocols for supporting reliable distributed applications in P2P systems", in *International Conference on Network-Based Information Systems (NBiS)*, 2007, pp. 345–354.

[3] SUN Microsystems, "JXTA v2.0 protocols specification", 2007, https://jxta-spec.dev.java.net/nonav/JXTAProtocols.html.

[4] G. Combs, "Wireshark", 1998, http://www.wireshark.org/.

[5] D. Sax, "DNS spoofing (malicious cache poisoning)", 2003, http://www.sans.org/rr/firewall/DNS_spoof.php.

[6] J. Arnedo-Moreno and J. Herrera-Joancomartí, "A survey on security in JXTA applications", *Journal of Systems and Software*, *To be published (accepted, in press). 2009*.

[7] B. Yeager, "Enterprise strength security on a JXTA P2P network", *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P'03)*, 2003.

[8] M. Wahl, T. Howes, and S. Kille, "Lightweight directory access protocol (v3)", 1997, http://www.ietf.org/rfc/rfc2251.txt.

[9] L. Kawulok, K. Zielinski, and M. Jaeschke, "Trusted group membership service for jxta", in *Computational Science (ICCS'04)*, 2004, Lecture Notes in Computer Science Volume 3038.

[10] W. Simpson, "PPP challenge handshake authentication protocol (chap)", 1996, http://tools.ietf.org/html/rfc1994.

[11] L. Yunhao and H. Jinpeng, "Access control in peer-to-peer collaborative systems", *First International Workshop on Mobility in Peer-to-Peer Systems (MPPS)*, pp. 835–840, 2005.

[12] M. Amoretti, M. Bisi, F. Zanichelli, and G. Conte, "Introducing secure peer groups in SP2A", 2005, pp. 62–69.

[13] "JXTA 2.5 RC1", June 2007, http://download.java.net/jxta/build.

[14] T. Dierks and C. Allen, "IETF RFC 2246: The TLS Protocol Version 1.0", 1999, http://www.ietf.org/rfc/rfc2246.txt.

[15] D. Bailly, "CBJX: Crypto-based jxta (an internship report)", July 2002.

[16] CCITT, "The directory authentication framework. recommendation", 1988.

[17] SUN Microsystems, "Java cryptography architecture (JCA)", 2008, http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.

[18] Ed. S. Josefsson, "IETF RFC 3548: the base16, base32, and base64 data encodings", 2003, http://www.ietf.org/rfc/rfc3548.txt.

[19] J. Arnedo-Moreno and J. Herrera-Joancomartí, "Persistent interoperable security for jxta", in *Proceedings of the Second International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC) 2008*. 2008, pp. 354–359, IEEEPress.

[20] W3C, "XML-signature syntax and processing", 2002.

[21] NIST, "Validated FIPS 140-1 and FIPS 140-2 cryptographic modules", 2009.

[22] B. Kaliski and J. Staddon, "PKCS#1: RSA cryptography specifications. version 2.0", 1998.

[23] T. Aura, "Cryptographically generated adresses (CGA)", http://www.ietf.org/rfc/rfc3972.txt.

[24] Microsoft, "MSDN library. cryptography", 2007.