AQTR 2004 (THETA 14)
2004 IEEE-TTTC - International Conference on Automation, Quality and Testing, Robotics
May 13 – 15, 2004, Cluj-Napoca, Romania

ENHANCING SECURITY IN JINI NETWORKING TECHNOLOGY

Nikolaos Papamichail, Luminita Vasiu

School of Computer Science, Middlesex University, White Hart Lane, N17 8HR, London, UK
Email: n.papamichail@mdx.ac.uk, l.vasiu@mdx.ac.uk

ABSTRACT
A number of solutions have been proposed to overcome security problems related with Jini networking technology. The Davis Project is the latest security effort that has been integrated with the existing Jini model. The purpose of this paper is to recognise some disadvantages related with the algorithm responsible for verifying trust in Jini proxy objects. We present some initial ideas of an alternative algorithm for performing trust verification and the advantages that this approach would possess.

KEYWORDS: Jini security, Proxy trust, Service discovery.

1. INTRODUCTION

Jini [1] is a service discovery system based in Java that enables the creation of communities of networked components. Hardware and software components employ Jini in order to advertise their services on the network. Similarly, clients use Jini's discovery capabilities to locate and utilize services in a spontaneous manner. There is no fixed protocol that needs to exist for the communication between the components of the system to take place. Services are able to dynamically join and leave the network and strong self-healing capabilities are employed to overcome faults of traditional networks.

The spontaneity that characterises a Jini system offers enormous advantages but is also responsible for some immense security problems. Issues related with Jini security have been identified and numerous solutions have been proposed [2, 3, 4, and 5]. The proposed ways of counteracting Jini security problems involved well known security mechanisms like certificates [2], capability managers [4] and digital signatures [6]. All of them, however, involved some kind of centralised entity that somehow restricted the spontaneity of the system.

The Davis Project [8] is the latest security model that has been tightly coupled with the existing Jini model. The security model involves an algorithm responsible for verifying trustworthiness in Jini proxy objects. The purpose of this paper is to underline some deficiencies of the existing algorithm and point out the advantages that an alternative approach would possess.

In Section 2 we present some background information about Jini. The current algorithm used for verifying proxy objects is presented in Section 3, together with some potential problems related with the existing algorithm. In Section 4 we propose an alternative way of verifying proxy trust and summarise the advantages gained. Finally, conclusions are drawn in Section 5.

2. BACKGROUND

This Section describes the main components and protocols employed by Jini technology. The following has to be present in order for a Jini interaction to take place

2.1 Services

Every entity that participates in a Jini system and provides some functionality is perceived as a service. No separation is made regarding the type or the characteristics of the service. A service could be either a hardware device, a piece of software or a human user. Jini provides the means for services to form interconnected systems, and each one separately to offer its resources to interested parties or clients.

2.2 Proxy Objects

In order to expose their functionality, services create an object that provides the code by which they can be exploited by potential clients, the proxy object. The proxy object contains the knowledge of the service's location and the protocol that the service implements. It also exposes an interface that defines the functions that can be invoked. A client is able to make use of a service only after the correspondent service's proxy object is downloaded to the client's local space. By invoking functions defined in the proxy interface, clients are able to contact and control services. Clients need only to be aware of the interface that the proxy implements and not of any details of the proxy implementation.

2.3 Lookup Service

The Lookup Service (LUS) is a special kind of service that is part of the Jini infrastructure. It provides a mechanism for services to participate in a Jini system and for clients to find and employ these services. The Lookup Service may be perceived as a directory that lists all the available services at any given time inside a Jini community. Rather than listing String based entries that point back to the location of a service, the Lookup Service stores proxy objects registered by Jini services.

2.4 Discovery Join and Lookup

Relevant to the use of Lookup services are three protocols called discovery, join and lookup [1]. Discovery is the process where an entity, whether it would be a service or a client, is trying to obtain references to a lookup service. After a reference has been successfully obtained, the entity might register a proxy object with the Lookup service (join), or search the Lookup Service for a specific type of service (lookup). The discovery protocol provides the way for clients and services to find available Lookup Services in the network, and for Lookup Services to announce their presence.

3. THE DAVIS PROJECT

Numerous solutions have been proposed to enhance security in a Jini system. These solutions are relevant to a specific portion of security requirements, like trust establishment between services [2, 4] and lease related issues [7]. Their methodology was based on applying well known security mechanisms to deal with specific Jini problems. While the application of proven security models might be adequate to deal with some of Jini's security weaknesses, other security requirements might be hard to satisfy without restricting usability to application dependent scenarios. Jini has introduced some innovative features like dynamic downloaded proxies. Therefore the adaptation of any single existing security model as a way to achieve end-to-end security without restricting applicability would be hard to prove adequate, unless being tightly coupled with the Jini model.

The problem of tightly coupling security with the existing Jini model and infrastructure has been resolved by the Davis Project [8]. The main focus of the Davis Project is to satisfy the primary requirements for Jini security [9], specifically authentication, authorisation, integrity and confidentiality. The integration of the programming model provided by the Davis Project with the existing Jini libraries has resulted in the release of the Jini starter kit version 2 [1].

The Davis project involves numerous security mechanisms. Constraints are employed by service providers and clients to specify the type of security required. Integrity mechanisms are also utilised to ensure that the code of proxy object remains unchanged and encryption techniques are used to guarantee the confidentiality of the communication. In this paper we focus in the algorithm employed to establish trust between the client of a service and proxy objects.

3.1 The Proxy Trust Verification Algorithm

In order for clients of a Jini system to verify the trustworthiness of downloaded proxy objects, the following algorithm is employed by the current security model:

1. The client examines the classes that the proxy object is composed of. If the classes are local, relevant to the client, the proxy object is considered trustworthy and is used to authenticate the service provider, otherwise the client performs step 2.
2. The client places a call through the proxy object asking for a second object, the bootstrap proxy. The bootstrap proxy is constructed by the service provider and should only be consisted of local classes relevant to the client.
3. The client examines the locality of the bootstrap proxy classes. If the classes are local, in relation with the client, the client performs a method call through the bootstrap proxy, requesting the service provider to authenticate.
4. After the service has been successfully authenticated to the client, a third object is passed to the client called the verifier. The verifier is responsible for performing a number of checks to the first proxy object, in order to verify whether the service provider trusts the initial proxy object or not.
5. If the verifier determines that the initial proxy object is trustworthy, the client makes use of the initial proxy object to access the service functionality. The bootstrap proxy and the verifier are no longer required and are deleted. In any other case the initial proxy object is dropped.

3.2 Problems with the Current Algorithm

The algorithm described above handles the problem of verifying trust in a Jini proxy object. However a number of potential problems also arise. The first is that clients have to rely on an object downloaded from an unknown source (the proxy object) to obtain the bootstrap proxy. In order for the latter to occur, clients have to place a remote call through an untrusted object. Since the functionality that proxy objects implement is unknown, clients may unintentionally execute an operation that presents a security risk in case the proxy is a malicious object.

The second problem that potentially arises is that the service provider has to have some knowledge of the type of classes that are local to the user. If the bootstrap proxy is not consisted entirely by local classes, relevant to the client, the client would not utilize it to obtain the verifier.

A third type of problem is related to the way and type of checks that the verifier performs to the proxy object. There is no standardised set of tests that could be performed, since these are left for the service providers to implement. The method suggested is that the verifier carries the code of the proxy object. By checking the equality of the code that the verifier carries with the code of the proxy object, it is possible for a service to identify the correctness of the proxy object. However, there is no way to ensure whether the checks performed are adequate or if any checks are performed at all. Therefore a ‘lazy’ verifier that just confirms the correctness of proxies without performing any checks might incorrectly identify a malicious object as a legitimate one.

Finally faults might occur if a service provider updates the implementation of the proxy object without updating the implementation of the verifier too. In that case legitimate proxy objects would not be able to be identified correctly, since the equality check would fail. Therefore the service provider might unintentionally cause a denial of service attack not initiated by a malicious client, but by himself.

4. PROXY VERIFICATION BASED ON A LOCAL GENERATED VERIFIER

Instead of relying on the untrusted proxy object downloaded from an unknown source to obtain a proxy verifier, clients might be able to protect themselves from malicious proxy objects by using their own local verifier. The verifier is generated locally by clients before any participation in a Jini federation takes place. In order to create the verifier, clients specify their security requirements such as authentication, confidentiality and integrity. These requirements are injected to the verifier and might vary for different scenarios. Specification of the security requirements is similar to the concept of constraints specified by the current Jini security model [8]. This permits the specification of application independent security requirements and allows better interoperability with the current security model. The difference is that the client requirements are not injected into a downloaded proxy, but into the locally generated verifier.

In order to verify that the downloaded proxy object can be trusted, the following process is performed:

1. Before any discovery process takes place, the client generates a local verifier
2. Client’s security requirements are injected to the verifier by the client
3. The client performs discovery of the Lookup Service and downloads a service proxy object
4. Before any interaction with the proxy takes place, the proxy object is passed to the verifier
5. The verifier performs a series of security checks according to the client requirements and makes a decision on behalf of the client about the trustworthiness of the proxy object

6. In case the verifier has decided that the security requirements are satisfied, the client interacts with the service through the proxy object as defined by Jini programming model. The described process is illustrated in Figure 1.

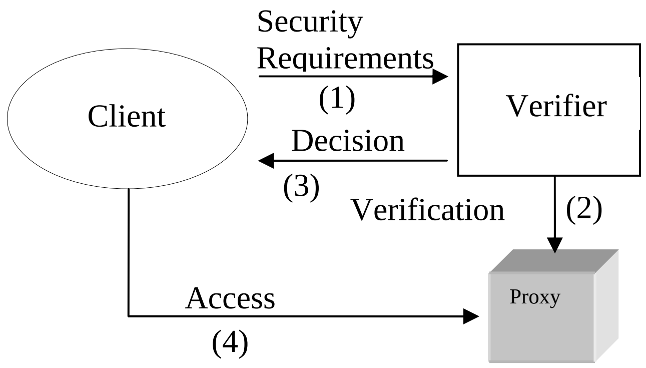

Figure 1 Proxy trust verification by a local verifier

Initially the client generates the verifier and specifies the security requirements (1). The verifier performs a series of tests to verify trust in the proxy object (2). The result of the verification procedure is expressed as a decision and the client gets notified (3). If the proxy has been considered to be trustworthy, the client is allowed to contact the proxy object (4) and access the related service.

Comparing this solution with the default proxy verification algorithm, in both algorithms the client is responsible for specifying the type of security required. However, the entity that is responsible for enforcing these requirements is not an untrusted proxy object anymore, but a locally generated verifier. The type of checks performed and the way these are carried out is much more transparent from the client's point of view. Moreover, clients do not have to rely on a verifier object downloaded from a service since the process of such object verifying the initial proxy object is not clear to the client. Therefore the problem of a service generated verifier that performs no actual check to the proxy object, resulting in the verification of a faulty proxy, is eliminated.

Service providers also do not need to worry about having to provide a bootstrap proxy and a verifier. The only entity that services need to expose is the default proxy object. Absence of a bootstrap proxy eliminates the need for services to implement an object based on the assumption that it would consist of classes that the client already has. Moreover, the current algorithm dictates that every time the implementation of a proxy object changes, the verifier object has to change as well, since proxy verification is based on equality checking. Finally by eliminating the need for services to produce two additional objects (the bootstrap proxy and the verifier), administration burden is removed from the service provider.

5. CONCLUSION

In this paper we review the current algorithm responsible for verifying trustworthiness in proxy objects and some problems related to it. We sketched out the advantages that an alternative implementation would possess based on a local generated verifier. By generating a local verifier the client is allowed better control over the required security. Assumptions upon the locality of the classes that compose the verifier are abandoned, and the administrative burden of updating the verifier according to the implementation of the proxy object is also removed from the service provider. Finally the proposed solution is fully interoperable with the existing security model.

REFERENCES

[1] Sun Microsystems Inc. Jini architecture specification. http://wwws.sun.com/software/jini/specs/jini2_0.pdf , [Accessed 12 Feb. 2004]

[2] Eronen, P., Lehtinen, J., Zitting, J., and Nikander, P. [2000]. Extending Jini with Decentralized Trust Management. *In Short Paper Proceedings of the 3rd IEEE Conference on Open Architectures and Network Programming (OPENARCH 2000),* pages 25-29.

[3] Eurescom, [2001]. Jini & Friends @ Work : Towards Secured Service Access. http://www.eurescom.de/~pub-deliverables/P1000-series/P1005/D5/p1005ti4.pdf [Accessed 09 Dec. 2003]

[4] Hasselmeyer, P., Kehr, R., and Voß M. [2000].Trade-offs in a Secure Jini Service Architecture. In *3rd IFIP/GI International Conference on Trends towards a Universal Service Market (USM 2000), Munich, Germany*. Springer Verlag, ISBN 3-540-41024-4, pp. 190-201

[5] Kagal, L., Finin T. and Peng, Y. [2001]. A Delegation Based Model for Distributed Trust. *In Proceedings of the IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, pp 73-80

[6] Schoch, T., Krone, O., and Federrath, H. [2001]. Making Jini Secure. *In Proc. 4th International Conference on Electronic Commerce Research*, pp. 276-286.

[14] P. Hasselmeyer, M. Schumacher, M. Voß. [2000]. Pay As You Go - Associating Costs With Jini Leases.*4th International Enterprise Distributed Object Computing Conference (EDOC 2000),* IEEE Publishing, ISBN 0-7695-0865-0, pp. 48-57.

[15] The Davis project http://davis.jini.org/ [Accessed 21 Feb. 2004]

[16] Bob Scheifler [2002], Comprehensive Network Security for Jini Network Technology Java One Conference Presentation, San Francisco. http://servlet.java.sun.com/javaone/sf2002/conf/sessions/display-1171.en.jsp [Accessed 10 Dec. 2003]