Future Generation Computer Systems 27 (2011) 476–485

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Verifying a delegation protocol for grid systems

Benjamin Aziz [a,*], Geoff Hamilton [b]

[a] School of Computing, University of Portsmouth, Portsmouth, UK
[b] School of Computing, Dublin City University, Dublin, Ireland

ARTICLE INFO

Article history:
Received 9 August 2010
Received in revised form
7 December 2010
Accepted 9 December 2010
Available online 17 December 2010

Keywords:
Delegation
Security
Protocol verification
Static analysis
Grid

ABSTRACT

In this paper, we design a non-uniform static analysis for formally verifying a protocol used in large-scale Grid systems for achieving delegations from users to critical system services. The analysis reveals a few shortcomings in the protocol, such as the lack of token integrity and the possibility of repudiating a delegation session. It also reveals the vulnerability of non-deterministic delegation chains that was detected as a result of adopting a more precise analysis, which allows for more participants in the protocol than the original protocol designers envisaged.

1. Introduction

DToken is a lightweight delegation protocol [1] that has recently been proposed as one solution for the problem of delegation from users to software services and resources in large-scale Grid systems. Grid middleware systems, such as Globus[1] or GLite,[2] allow users to access the computational and storage resources of the Grid. A typical model of Grids is often called Virtual Organisations, which permits users from one organisation to access and use resources belonging to another organisation under certain resource sharing policies. However, such cross-organisational provisioning of resources requires that critical issues of trust and security be managed. One issue is related to delegations performed by the user to the gateway, and within the system on behalf of the user's original delegation.

Applying analysis techniques is one means by which critical services, such as a delegation service, can be verified and validated against vulnerabilities and incorrect usage therefore increasing the levels of confidence in the overall system functionality. In this paper, we apply formal analysis techniques that we previously developed in [2–5] to verify the DToken protocol against certain core properties expected to hold in any robust delegation protocol. We show, through our analysis, that properties like basic token integrity validation, verifiable non-repudiation and deterministic delegation chains actually do not hold in the protocol. Our non-uniform analysis allows for different vulnerabilities to be discovered at different levels of abstraction. Indeed, the lowest level of abstraction (i.e. the most precise analysis) demonstrates a vulnerability, the lack of deterministic delegation chains, which was overlooked in the original design of the protocol as a result of adopting approaches which were too abstract.

In the rest of the paper, we give an overview of the DToken protocol in the next Section 2 and discuss three essential properties that we expect to hold of this protocol. In Section 3, we define the simple applied pi calculus language and give its operational semantics. In Section 4, we define a non-standard semantics that captures name substitutions as a result of communications, and in Section 5, we introduce a computable approximation of this non-standard semantics. In Section 6, we revisit the DToken protocol applying our static analysis to it. Finally, in Section 7, we discuss related work and in Section 8, we conclude the paper.

2. The delegation protocol

We give an overview here of the DToken delegation protocol as was defined in [1]. The protocol comprises secure communications between a *Delegator*, *Dor*, and a *Delegatee*, *Dee*. The following sequence of messages describes the interactions in the protocol:

1. $Dor \rightarrow Dee$: $C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee0}, Sig_{Dor \rightarrow Dee}$
2. $Dee \rightarrow Dor$: $C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \rightarrow Dee}, DS_{Dor \rightarrow Dee}, Sig_{Dor \rightarrow Dee}, Sig_{Dee \rightarrow Dor}, C_{Dor_{CAs}}$

* Corresponding author.
E-mail address: benjamin.aziz@port.ac.uk (B. Aziz).

[1] www.globus.org.
[2] glite.web.cern.ch.

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.12.003

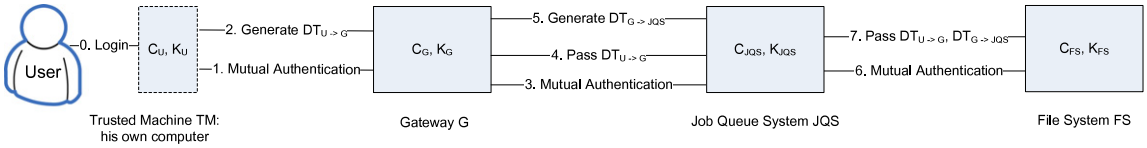

Fig. 1. DToken chained delegation through a gateway (cited from [1, Figure 5]).

where,

C_{Dor}: Long-term public key identity certificate of *Dor*,
C_{Dee}: Long-term public key identity certificate of *Dee*,
V_{fr}: The starting validity date of the delegation,
V_{to}: The expiry date of the delegation,
TS: A timestamp representing the time the message is generated,
$P_{Dor \to Dee}$: The delegated permissions from *Dor* to *Dee*, which include the delegation policies,
$DS_{Dor \to Dee}$: A number representing the delegation session identifier,
$DS_{Dor \to Dee0}$: Initial empty value of $DS_{Dor \to Dee}$, which for simplicity is assumed to be *Null*,
$Sig_{Dor \to Dee}$: The signature of the delegation information in the first message signed by the private key of *Dor*, K_{Dor}, where

$$Sig_{Dor \to Dee} \stackrel{\text{def}}{=} |\{C_{Dor}, C_{Dee}, V_{fr}, V_{to}, TS, P_{Dor \to Dee}, DS_{Dor \to Dee0}\}|_{K_{Dor}},$$

$Sig_{Dee \to Dor}$: The signature of *Dor*'s signature in the first message signed by the private key of *Dee*, K_{Dee}, where $Sig_{Dee \to Dor} \stackrel{\text{def}}{=} |\{Sig_{Dor \to Dee}\}|_{K_{Dee}}$, and
$C_{Dor_{CAs}}$: The list of subordinate CAs linking C_{Dor} to the trusted root authority.

There are a few points to note about the protocol as described in [1]. In the first message, $DS_{Dor \to Dee}$ has an empty value, which we assume to be some default value like *Null*. The choice of the delegatee's decision to assign the delegation session identifier rather than the delegator was not explained by the designers of the protocol. Timestamps in both messages are neglected in our analysis, as these are often non-reliable means of sequencing events in distributed systems due to the problem of clock synchronisation.

The second message is referred to as the DToken (Delegation Token) from *Dor* to *Dee*, written as $DT_{Dor \to Dee}$, which represents the mutual delegation agreement between *Dor* and *Dee*. In this message, *Dee* will update the value for $DS_{Dor \to Dee}$ assigning it the current delegation session identifier. Furthermore, in between the two messages, *Dee* performs some verification tests to ensure that *Dor* is authorised to delegate permissions to *Dee* and to ensure that the security information *Dor* has sent in the first message is indeed valid. For example, *Dee* will ensure that the certificates are valid and can be traced up to the root of trust and that the token has not expired.

Another main assumption in the protocol is that all the communications between *Dor* and *Dee* are carried over Secure Sockets Layer (SSL)-based channels [6]. This means that *Dor* and *Dee* are sure of each others identities and the privacy of messages is guaranteed against external intruders. However, communication security does not imply that such external intruders cannot participate in the protocol like any other agents.

The protocol is claimed to form chains of delegation. Once the last delegatee in the delegation chain decides to stop delegating, it is assumed that it will execute the delegated permissions, $P_{Dor \to Dee}$, by applying them to a *Delegation Enforcement Point* (DEP), typically a service or a resource. The DEP will perform a couple of validation steps to check the integrity of the DToken containing the permissions and other DTokens forming the full delegation chain.

In [1], the authors give an example of a delegation chain in Grid systems as shown in Fig. 1.

This chain consists of the user as the delegation root, who then delegates some permissions to run a job to a gateway (a computer on which the user can login). Then the gateway delegates the job to a job queueing system, which itself is the end of the delegation chain. The job queueing system will then execute the job on a file system (the DEP). One aspect of the communication between the job queueing system and the file system is that the DTokens generated in the previous communications are passed as a set. We demonstrate later how this aspect introduces a vulnerability in the protocol.

2.1. Protocol properties

The DToken protocol was designed to achieve lightweight delegation focusing on the traceability of the participants rather than their privacy (in contrast to protocols such as [7]), therefore, it should sustain a few properties related to its purposes and functionality.

2.1.1. DToken integrity

This property refers to the success of a DEP in validating the integrity of a DToken. This implies that the two hash comparisons mentioned in Section IV in [1] must always succeed.

Property (*DToken Integrity Validation*)**.** *A DToken is said to be valid if the following equations are true:*

$$hash(C_U, C_G, V_{fr}, V_{to}, TS, P_{U \to G}, DS_{U \to G}) = decrypt(Sig_{U \to G}, C_U)$$
$$hash(Sig_{U \to G}) = decrypt(Sig_{G \to U}, C_G). \quad \square$$

The first of these compares the hash of the delegation information to the decryption of the signature of the delegator. The success of this validation implies the consent of the delegator to the delegation. The second compares the hash of the delegator's signature with the decryption of the delegatee's signature. This second validation implies the consent of the delegatee to the delegation. In general, the success of both comparisons ensures that the DToken's integrity is preserved.

2.1.2. Traceability and accountability

Traceability is defined in [1] as the ability of the delegatee to uniquely identify the identity of any of the previous delegators. Accountability, on the other hand, is verifiable traceability where such identity is cryptographically identifiable. Accountability is also called non-repudiation. More specifically, we define non-repudiation as the property that neither the delegator nor the delegatee can deny their acceptance of the delegation at the point of permission execution. This implies further that the delegatee must not be able to use the delegated permissions before at least having signed the DToken containing those permissions initiated by the delegator.

Formally, we define the property of verifiable non-repudiation as follows, assuming *Perms* is the set of all permissions.