Credential Delegation: Towards Grid Security Patterns

Michael Weiss
School of Computer Science, Carleton University, Canada
weiss@scs.carleton.ca

Introduction

Security is a central concern in grid computing. A grid is a platform for sharing resources (such as computers and storage) across organizational boundaries. Thus, using a grid raises fundamental security challenges such as ensuring that only trusted organizations access our resources, data integrity and confidentiality, and delegation of credentials to grid applications. The Grid Security Infrastructure (GSI) implemented by Globus [5] and other grid toolkits [1] addresses many of these security challenges. A practical introduction to the GSI is given in the Globus Toolkit 4 tutorial [8]. Our goal is to document the patterns underlying such grid security solutions.

In this paper we describe the Credential Delegation pattern. This is one pattern from a pattern language for grid security we are writing. One of the other patterns that we will describe is Mutual Authentication, a precondition for applying the Credential Delegation pattern. Its intent is for two parties (client and server) to verify each other's identities. In our description of Credential Delegation, we follow the format for security patterns defined in [7]. Our focus is on security aspects. For a general pattern-oriented introduction to grid computing see the grid architectural pattern in [3].

Credential Delegation

In a grid parties may need to act on behalf of other parties. They should be able to authenticate themselves as acting on behalf of those parties. Therefore, issue a special type of certificate (proxy certificate) signed by the original party (grantor) that confirms that the holder of this certificate (grantee) is allowed to act on its behalf.

Example

A user at site A requests to run a simulation on a powerful server at site B. But this simulation may – on the user's behalf – need to access input data or invoke services on another server at site C. Typically, these parties (user, simulation, and providers of input data or services) are all in different security domains, and may use different local security mechanisms (Windows, Unix). Figure 1 summarizes this scenario.

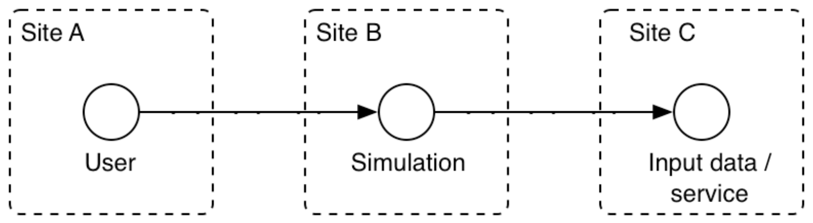

Figure 1: A user at site A requests to run a simulation at site B. This simulation needs to access input data or invoke services at site C on the user's behalf

Context

A grid in which parties need to act on behalf of other parties. You are using Mutual Authentication to establish secure communication between the parties.

Problem

When a party requests a service on behalf of another party, it should be able to authenticate itself as acting on behalf of that party. Refer to those parties as George, Fred, and Ted. Fred requests a service from George, who in turn requests services – on Fred's behalf – from Ted. One solution would be to ignore Fred's identity, but this has the obvious drawback that George may lack the credentials to request the service from Ted. Another solution would be to specify that the request is made on Fred's behalf, and for Ted to contact Fred about the validity of the request. But it is impractical to ask Fred about each service requested on its behalf. Fred may also already have signed off by the time Ted tries to contact it. Finally, George could demonstrate that it is acting on Fred's behalf, if Fred had provided its private key to George. However, this solution results in a severe security breach, as private keys must always remain secret.

The solution to this problem must balance the following forces:

- George must operate independently from Fred: Fred cannot be expected to be available, or to provide any information beyond what it initially provided.
- George, as it is acting on Fred's behalf, must be provided with a means of demonstrating to other services such as Ted that it represents Fred.
- This means should not be valid beyond the scope of the request. Otherwise, George[1] could pretend to act on Fred's behalf in the context of unrelated requests.

Solution

Issue a special type of certificate signed by the original party (grantor) that confirms that the holder of this certificate (grantee) is allowed to act on its behalf. A private/public key pair is generated specifically for such a proxy certificate, and the

[1] Or somebody else who manages to compromise the means.

authority to act on the grantor's behalf extends only to the holder of this private/public key pair, that is, the grantee. One issue to be addressed by this solution is that such a proxy certificate would allow the grantee to act unconditionally on the grantor's behalf. Above we referred to the grantor as Fred and the grantee as George.

While the grantor might trust the grantee to act on its behalf during the context of the specific request, it is unlikely to trust the grantee indefinitely. Thus, the lifetime of the proxy certificate is limited (usually to a few hours). Should some other party compromise the proxy certificate, it will now be of limited use. Some implementations of this solution (e.g. X.509) also support the placement of restrictions on the grantee by means of proxy certificate policies. For example, the proxy certificate might only allow the grantee to read certain files. This is another way of mitigating the unintended use of a proxy certificate. The use of proxy certificates results in a delegation of credentials (i.e. the grantor's identity) to the holder of the certificate (grantee).

Structure

Figure 2 shows the structure of this pattern. A `Grantor` asks a `Grantee` to perform a request. As executing the request may involve the invocation of another `Service`, the `Grantor` gives the `Grantee` permission to invoke such a `Service` on its behalf. This permission is embodied in a `Proxy Certificate`, which the `Grantee` creates and asks the `Grantor` to sign. The `Service` can verify the identity of the `Grantor` by checking the `Proxy Certificate`, that is, it does not need to contact the `Grantor`.

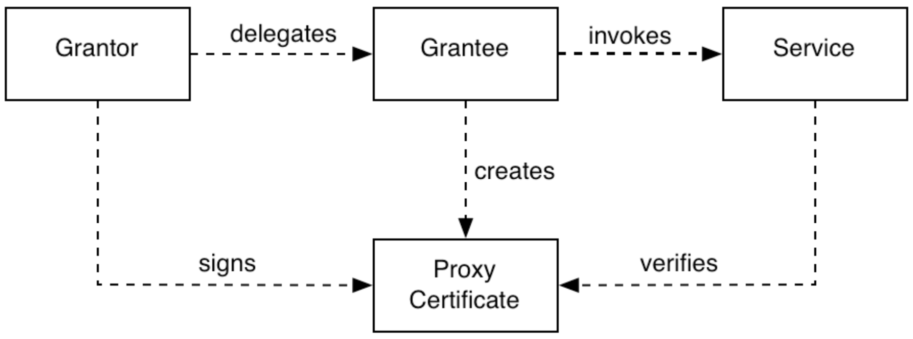

Figure 2: Structure of the Credential Delegation pattern

Dynamics

The sequence diagram in Figure 3 illustrates the process of generating and using a proxy certificate. The steps are as follows:

1. The `Grantor` and `Grantee` establish a secure channel (using SSL) to assure the integrity and confidentiality of any subsequently exchanged information.
2. The `Grantee` generates a temporary public/private key pair to be used for the `Proxy Certificate` and signs the new public key with its own private key.
3. The `Grantee` sends the signed `Proxy Certificate` request to the `Grantor`.

4. The `Grantor` signs the `Proxy Certificate` request with its private key. This results in the creation of a new certificate, the `Proxy Certificate`.
5. The `Grantor` then sends the `Proxy Certificate` to the `Grantee`.
6. The `Grantee` uses the `Proxy Certificate` to establish a secure channel with the `Service` that it needs to invoke on the `Grantor`'s behalf.

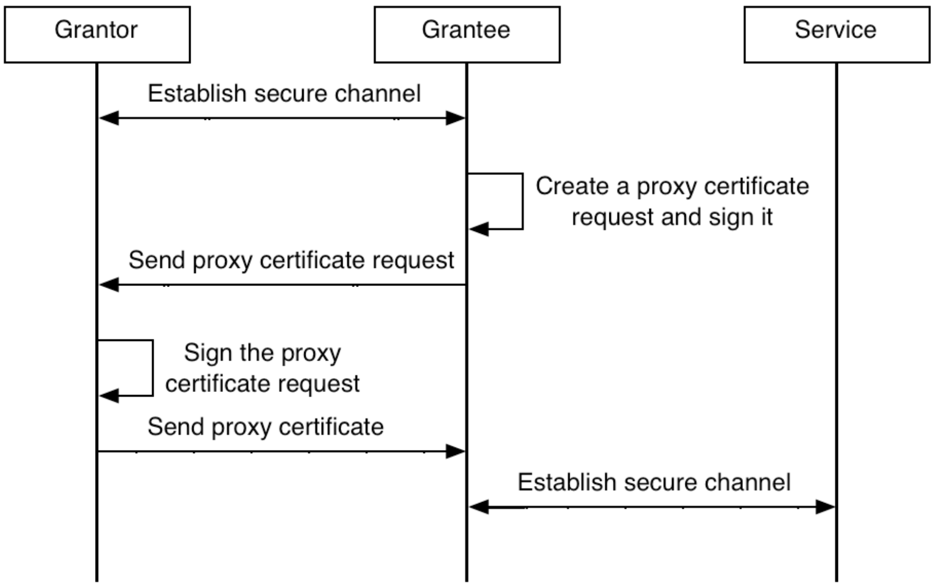
Grantor
Grantee
Service
Establish secure channel
Create a proxy certificate request and sign it
Send proxy certificate request
Sign the proxy certificate request
Send proxy certificate
Establish secure channel

Figure 3: Generation and use of a proxy certificate

A proxy certificate is like a digital certificate, except that it is not signed by a certificate authority. Thus, for a service to establish the validity of the proxy certificate, it first verifies that the certificate was signed by the grantor, applying the grantor's public key to obtain the certificate's public key as signed by the grantee. Then, it applies the grantee's public key to extract the certificate's public key. The service obtains the grantee's public key from the grantee's certificate, which is typically sent together with the proxy certificate by the grantor, and is signed by a certificate authority.

Example Resolved

Before the simulation (grantor) at site B can access input data or invoke services at site C on behalf of the user (grantee) at site A, the simulation and user application mutually create a proxy certificate. The service request is then accompanied by the proxy certificate and the user's certificate, which allows the service at site C to verify the proxy certificate, and to establish a secure channel between simulation and service.

Figure 4 shows how the example is resolved. Using stereotypes the role of each component (grantor, grantee, service) is indicated in correspondence with the pattern.

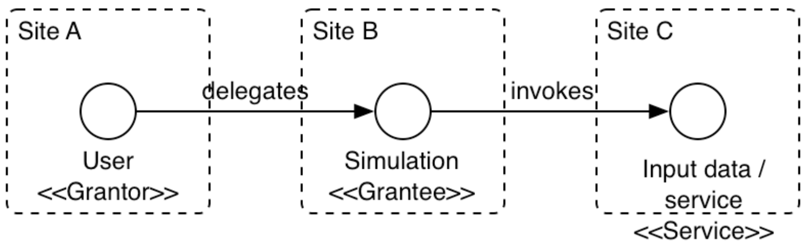

Figure 4: Example resolved (stereotypes indicate roles in the pattern)

Known Uses

The principle of credential delegation using proxy certificates was independently introduced in the Digital Distributed System Security Architecture (DDSSA) [4], and in version 5 of the Kerberos authentication system [6]. Later it became the basis for the design of the Grid Security Infrastructure (GSI) [2]. GSI is a library for public key based authentication and authorization designed for inter-domain resource sharing. It is the most widely adopted security solution for grid middleware, and prominently in the Globus Toolkit [5]. Other uses of credential delegation, which predate the development of Globus, are documented for the I-WAY system [1, chapter 4], and the Legion Grid [1, chapter 10], among others. In I-WAY, proxy user ids were used to authenticate the user to remote resources on the user's behalf. In the Legion Grid, user credentials are communicated through authentication objects, and subsequently used to access data on the user's behalf. RFC 3820 [9] provides a detailed discussion of the proxy certificate technology underlying the GSI. Finally, it should be noted that although a critical component of GSI, credential delegation is not limited to grids.

Consequences

The following benefits may be expected from applying this pattern:

- Proxy certificates allow users to delegate authority to parties that then act on their behalf, even if the user is no longer available.
- The user's security is protected due to the short lifespan of proxy certificates.[2] Compromising a proxy certificate has far less impact, in terms of damage that can be done and recovery, than if the user's own certificate had been compromised.
- Use of proxy certificates also gives us single sign-on. When interacting with multiple resources, the user only needs to be authenticated once in order to create a proxy certificate, and can use the proxy certificate for subsequent authentications.

The following liabilities may arise from applying this pattern:

[2] A participant of the writer's workshop suggested refactoring this pattern into smaller patterns. For example, this and the next benefit of using the pattern could be documented in separate patterns with potential names of Short Lifespan and Single Sign-On. As we add to our pattern language we plan to rework this pattern accordingly.

- Performance is slightly degraded, because of handshake required to create a proxy certificate, but also improved slightly on subsequent authentications.
- The infrastructure becomes more complex in order to support proxy certificates.

See Also

The pattern uses Mutual Authentication (also known as Known Partners [7]) to establish the initial trust between a grantor and the grantee to which it delegates its credentials. There are many issues not directly addressed by the pattern, such as dealing with timeouts during transactions longer than the timeout period, and delegation of user credentials through a chain of systems. They will be addressed by future patterns.

Acknowledgements

My thanks go to my tiresome shepherd Jorge Ortega Arjona. His feedback has helped me significantly in bringing the workshop version of this paper into shape. I also want to thank the participants of my VikingPLoP workshop for their thoughtful comments: Jim Coplien, Aino Vonge Corry, Yngve Espelid, Jayashree Kar, Lars-Helge Netland, Rebecca Rikner, Andreas Rüping, Kristian Elof Sørensen, and Birgit Zimmermann.

References

1. Berman, F., Fox, G. and Hey, T., Grid Computing: Making the Global Infrastructure a Reality, Wiley, 2003.
2. Butler, R., Engert, D., Foster, I., Kesselman, C., and Tuecke, S., A National-Scale Authentication Infrastructure, IEEE Computer, 33, 60-66, 2000.
3. Camargo, R., Goldschleger, A., Carneriro, M. and Kon, F., Grid: An Architectural Pattern, Conference on Pattern Languages of Programs (PLoP), 2004.
4. Gasser, M. and McDermott, E., An Architecture for Practical Delegation in a Distributed System, Symposium on Research in Security and Privacy, 20-30, IEEE, 1990.
5. Globus Alliance, GT4 (Globus Toolkit 4) Security: Key Concepts, http://www.globus.org/toolkit/docs/4.0/security, 2006.
6. Neuman, B., Proxy-Based Authorization and Accounting for Distributed Systems, International Conference on Distributed Computing Systems, 283-291, 1993.
7. Schumacher, M., Fernandez-Bugliono, E., Hybertson, D., Buschmann, F. and Som- merlad, P., Security Patterns, Wiley, 2006.
8. Sotomayor, B. and Childers, L., Globus Toolkit 4: Programming Java Service, Morgan Kaufmann, 2006.
9. Tuecke, S., Welch, V. et al, RFC 3820 – Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, 2004, http://www.faqs.org/rfcs/rfc3820.html.