10.1 Authenticator

This pattern addresses the problem of how to verify that a subject is who it says it is. Use a SINGLE ACCESS POINT (279) to receive the interactions of a subject with the system and apply a protocol to verify the identity of the subject.

Example

Our system has legitimate users that use it to host their files. However, there is no way to make sure that a user who is logged in is a legitimate user. Users can impersonate others and gain illegal access to their files.

Context

Operating systems authenticate users when they first log in, and maybe again when they access specific resources. The operating system controls the creation of a session in response to the request by a subject, typically a user. The authenticated user, represented by processes running on its behalf, is then allowed to access resources according to their rights. Sensitive resource access may require additional process authentication. Processes in distributed operating systems also need to be authenticated when they attempt to access resources on external nodes.

Problem

A malicious attacker could try to impersonate a legitimate user to gain access to her resources. This could be particularly serious if the impersonated user has a high level of privilege. How can we prevent impostors from accessing our system?

The solution to this problem must resolve the following forces:

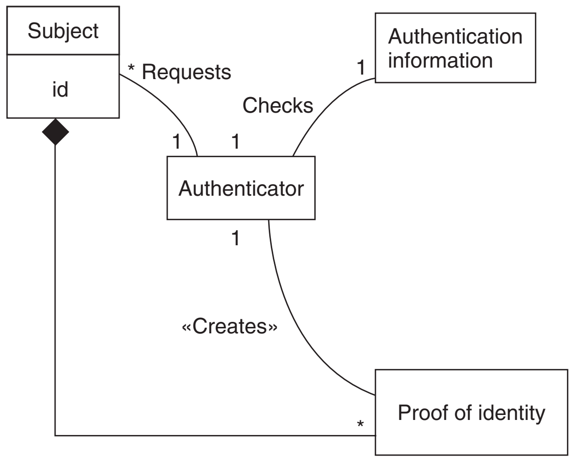
- There is a variety of users that may require different ways to authenticate them. We need to be able to handle all this variety, or we risk security exposures.
- We need to authenticate users in a reliable way. This means a robust protocol and a way to protect the results of authentication. Otherwise, users may skip authentication or illegally modify its results, exposing the system to security violations.
- There are trade-offs between security and cost—more secure systems are usually more expensive.
- If authentication needs to be performed frequently, performance may become an issue.

Solution

Use a SINGLE ACCESS POINT (279) to receive the interactions of a subject with the system, and apply a protocol to verify the identity of the subject. The protocol used may imply that the user inputs some known values, or may be more elaborate.

Structure

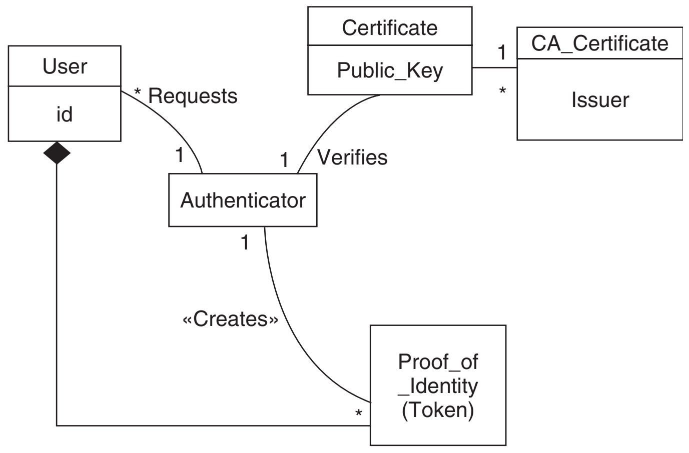
The figure below shows the class diagram for this pattern. A `Subject`, typically a user, requests access to system resources. The `Authenticator` receives this request and applies a protocol using some `Authentication Information`. If the authentication is successful, the `Authenticator` creates a `Proof of Identity`, which can be explicit, for example a token, or implicit.



AUTHENTICATOR

Dynamics

The figure on page 326 shows the dynamics of the authentication process. A user requests access to the AUTHENTICATOR (323). The AUTHENTICATOR (323) applies some authentication protocol, verifies the information presented by the user, and as a result a proof of identity is created. The user is returned a handle for the proof of identity.



Variant of AUTHENTICATOR: PKI authentication

Implementation

Assuming a centralized system, we need to carry out the following tasks to implement the pattern:

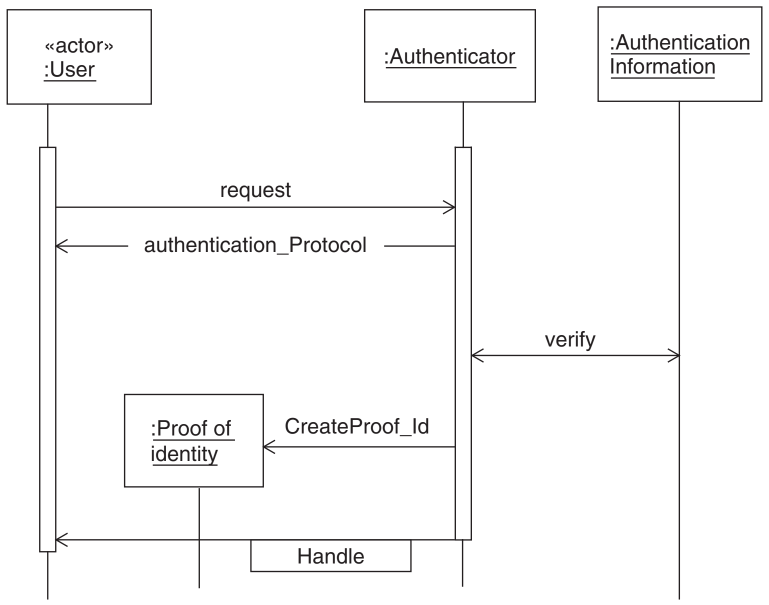
- Define authentication requirements, considering the number of users, degree of security required, and so on—see I&A REQUIREMENTS (192)
- Select an authentication approach
- Build the list of registered users—the authentication information.

Example Resolved

We adopted the use of passwords for authenticating our users. While not a perfect solution, we can keep out most of the impostors.

Variants

Single Sign-On (SSO) is a process whereby a subject verifies its identity, after which the results of this verification can be used across several domains and for a given amount of time. The result of the authentication is an authentication token used to qualify all future accesses by the user.



Authentication dynamics

PKI Authenticator. Public key cryptography is a common way to verify identity. This authentication can be described with a slight modification of the pattern in the first figure, as shown in the second figure above. An `Authenticator` class performs the authentication using a certificate that contains a public key from a certificating authority that is used to sign the certificate. The result of the authentication could be an authentication token used to qualify all future accesses by this user. In this case this is also a variant of SSO.

Known Uses

Most commercial operating systems use passwords to authenticate their users. RADIUS provides a centralized authentication service for network and distributed systems [Has02]. The SSL authentication protocol uses PKI for authentication. SAML, a Web Services standard for security, provides a way to implement an SSO architecture as one of its main uses [SAML].

Consequences

The following benefits may be expected from applying this pattern:

- Depending on the protocol and the authentication information used, we can handle any types of users and we can authenticate them in diverse ways.

- Since the authentication information is separated, we can store it in a protected area to which all subjects may have read-only access at most.
- We can use a variety of algorithms and protocols of different strength for authentication. The selection depends on the security and cost trade-offs.

  Three varieties include: something the user knows (such as passwords), something the user has (such as an identity card), something the user is (their biometrics), or where the user is (the terminal or node).
- Authentication can be performed in centralized or distributed environments.
- We can produce a proof of identity to be used in lieu of further authentication. This improves performance.

The following potential liabilities may arise from applying this pattern:

- The authentication process takes some time.
- The general complexity and cost of the system increases with the level of security.

See Also

DISTRIBUTED AUTHENTICATOR [Bro99] discusses an approach to authentication in distributed systems. SINGLE ACCESS POINT (279) (see Chapter 8) is an abstract pattern applied here: AUTHENTICATOR (323) is a concrete application of it. SINGLE SIGN ON is a variant implemented in many systems. REMOTE AUTHENTICATOR (and AUTHORIZER) [Fer03b] is intended for remote access to shared resources in a distributed system. Passwords are a specific authentication protocol—see PASSWORD DESIGN AND USE (217).