

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

PRINTED: TUESDAY, JULY 12, 2005

PREPARED BY HILBERT COMPUTING, INC.,
2005

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

TABLE OF CONTENTS

OVERVIEW.....	3
KEY SECURITY CONCEPTS.....	4
Authentication is Mostly Decoupled from Authorization.....	4
Authorization Should Be External to the Application.....	4
Security Implementation Should be Decoupled from the Application.....	5
IMPLEMENTATION APPROACH.....	7
Defining Key Abstractions.....	8
Declarative Assembly of Parts.....	9
SECURITY IN A BROADER CONTEXT.....	10
MIGRATION STRATEGY.....	11
SERVER-TO-SERVER SECURITY.....	12
Obtaining OS Credentials.....	12
Adding Public/Private Keys.....	13
Avoiding Replay Attacks.....	13
Network Validation.....	13
CHALLENGES.....	15
View Filtering.....	15
External Applications with Closed Security.....	16

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

Federated Identity Management.....16

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

OVERVIEW

I have been working on the implementation of authentication and authorization intermittently with two governmental organizations and five corporations over the past three years. From those efforts has come an object model that appears to be able to accommodate the complexity that is inherent in application security. Many of the concepts are based on the Java 2 security model. These concepts were pushed to a higher level of abstraction in order to provide independence from Java 2 security and to enable a migration path from the implementations in current applications.

This document outlines the key concepts and drivers for this architecture and software implementation for application security. It should be noted that all of the concepts discussed in this document (except for the section on challenges) have been implemented in Java. Most of the code is in production in at least one enterprise.

The work is not complete. There are areas that need additional development. The emergence of interoperability between enterprises creates additional challenges that are not fully investigated. Although the body of work is not finished¹, there is a sufficiently robust code base that can be leveraged to solve the majority of the issues facing those involved in deploying secure applications for their enterprise.

¹ As if anything in IT is ever really finished

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

KEY SECURITY CONCEPTS

There are a few concepts about security that are not entirely intuitive. It is important that some key concepts become well understood, since this provides the basis for the application security architecture and the framework that implements the architecture.

AUTHENTICATION IS MOSTLY DECOUPLED FROM AUTHORIZATION

Authentication is the process that validates the identity of a person or service. Authorization uses the identity of the person or service requesting access to a resource to determine if access should be granted. This description makes the two seem coupled together, but for the most part, they are not. The process by which the authentication takes place has no bearing on the resources that can be accessed by that authenticated entity. The only area of overlap is that the artifact from authentication must be recognized by the code handling the authorization.

If the authorization implementation uses LDAP, Kerberos, Active Directory or a relational database, it should have no bearing on the resources that are accessible by the authenticated entity. That realization provides the opportunity to decouple the authentication implementation from the application and the authentication implementation from the authorization implementation. This “artifact” of authentication in the Java 2 security implementation is the `javax.security.auth.Subject` object. That same artifact is used in the implementation in the framework described here, but the use of that object does not couple the implementation to Java 2 security. The implementation details are discussed later in this document and in several other documents.

AUTHORIZATION SHOULD BE EXTERNAL TO THE APPLICATION

This seems like an odd statement. Security is very much an internal part of the way an application works. However, the *mechanism* that is used to facilitate the relationships between the security elements such as roles, groups, permissions, etc. should be external to the application.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

There is a natural tendency for application developers to think in terms of if-then logic. “If the current user has a role of 'staff', then they are allowed to perform this task.” This approach significantly limits the flexibility and extensibility of security within the application. The application creates a semantic meaning for the word 'staff' within the application and fixes the granularity of control available in how that role is used throughout the application. This is the approach that was taken with J2EE servlet security and the limitations are well-known and often discussed in the industry.

Instead of thinking of authorization in “if-then” terms, authorization should be viewed as a gatekeeper function. Before code can proceed into a protected section of code, it has to pass an authorization block of code. The authorization block uses the artifact of authentication to call an external service to see if that particular user is allowed to access the resource. If so, the authorization block of code returns. Otherwise, it throws a security exception. Ideally, the implementation uses an external data model that contains the relationships between the authenticated entity and the permissions that are available to that entity so that those relationships can be changed and the granularity of access can be changed without affecting the application logic.

SECURITY IMPLEMENTATION SHOULD BE DECOUPLED FROM THE APPLICATION

There are multiple implementations of authentication services. In the list of commonly used authentication services are LDAP, ActiveDirectory, and Kerberos. Of course, many applications have an in-house developed method for authenticating users. If the applications are developed using the application programming interfaces (APIs) for those authentication technologies, that application is tightly coupled to that authentication technology. The better approach is to delegate authentication to a factory-constructed authenticator object that implements the specific authentication protocol. This approach will decouple authentication from the application, but there is still a loose coupling between the credentials and the authentication technique. For example, most current authentication techniques are implemented with userid and password authentication. If the implementation changes to PKI², then userid and password credentials are no longer the appropriate credentials to pass to the authenticator. For this reason, the collection of credentials should also be delegated to a factory-constructed assembly object to further decouple security from the application.

2 PKI stands for “Public Key Infrastructure” and involves the use of certificates and certificate authorities for third-party validation of certificates.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

Authorization should also be accomplished by delegating to a distinct object that implements the appropriate algorithms for authorizing access according to the security policies of your organization. That authorization object should also be factory constructed. As in the case of the assembly of credentials and the authentication, this will limit the coupling between the functional parts of security architecture to logically-named things as opposed to concrete implementations of code.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

IMPLEMENTATION APPROACH

The primary goal for the security implementation is, obviously, to support the key security concepts. There are additional considerations based on experiences in the field that motivated the implementation approach.

Some of my customers deliver technology solutions to customers as opposed to implementing solutions for internal corporate use. This drove the need to be able to adapt to whatever security implementations were chosen by the end-customer's organization. Even for those organizations that are developing an internal security implementation, this adaptability is still very relevant. Different authentication technologies are becoming available and may be more appropriate than the current implementation choice. The internal requirements may change due to internal policy or legislated requirements. There needs to be a way to move existing applications to those newer authentication technologies with a minimum of effort.

Another customer has spent over three years trying to determine the appropriate implementation technology for identity management. Since they are a large organization with a complex infrastructure, they consider their choice of implementation technology to be a key strategic decision. My assertion is that the effective use of software abstractions makes those kinds of decisions tactical instead of strategic. The bulk of the cost of moving to a different authentication technology is not the acquisition of the authentication software. It is the cost of converting the applications to use that technology. If the cost of changing applications to use different authentication technologies becomes trivial, then the choice of implementation technology can be viewed as tactical.

Another motivation for this design is to remove the applications programmer from the responsibility of implementing security. Applications development has a different focus than the development of infrastructure code such as security. Typically, most organizations will have considerably more developers working on delivering business functionality than there are developers working on infrastructure and operationally-oriented code. There is a desire to be able to keep applications developers focused on delivering the code to meet business requirements. One customer took that notion one step further. They had experienced a problem in the past where applications developers working on financial software had defrauded the company. They wanted a security system that would remove the semantics of security from the developer's domain and place it in an operational group that didn't have programming skills. Externalizing the implementation of security is a step toward defending against internal fraud. However, it must be noted that it is not a complete solution against internal fraud.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

Every enterprise has a mixture of application architectures for deployment. For Java developers, the focus for enterprise applications is on servlet development. However, there is an expectation that there will be some batch processing and some Java applications that run outside of a servlet container. The architecture and implementation approach should not be coupled to a servlet container, but neither should it be hostile to applications developed as servlets. This realization makes J2EE security unsuitable for the complex enterprise.

DEFINING KEY ABSTRACTIONS

The most difficult part of any design is properly identifying the key abstractions. The object model must be abstract enough to accommodate all of the known requirements and the requirements not yet realized. Conversely, the object model needs to be concrete enough to provide a structural basis for applications to exploit and must be easy for applications developers to understand and use.

The following table lists the primary abstractions for applications security

<i>Interface</i>	<i>Purpose</i>
Authenticator	This interface models the algorithms that can be used to authenticate credentials.
Authorization	Before code is executed that protects a resource, one or more <code>Authorization</code> objects are instantiated with a reference to the artifacts from authentication and executed. If the authorized subject has sufficient authorization to access the protected resource, then the <code>Authorization</code> code returns. Otherwise, the code throws a security exception.
CredentialsAssembler	Most application security implementations use a userid and password for credentials and those credentials are obtained through a user-interface challenge to the user. When other credentials are used, for example PKI, split password credentials or certificates, it is useful to create an abstraction between the application code and the implementation of how credentials are gathered. That allows for a complimentary change in the implementation of credentials gathering when there is a change in the authorization implementation.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

Concrete implementations of these abstractions are called out by the applications programmer through the use of the factory design pattern. Instead of the application instantiating a concrete implementation – and creating an undesirable coupling – the factory will instantiate the appropriate implementation of the security interfaces. All application code is programmed to the interface.

DECLARATIVE ASSEMBLY OF PARTS

Now that we have the key abstractions defined, we need a mechanism to put the concrete implementations in place. Specifically, we need a way to populate the factories that will be called by the applications code in a way that is external to the application code.

In order to manage the instantiation and population of the factory classes and provide the programming interface for the applications developers to use, we create a service class that provides for the lifecycle management of these security objects. The `Security-Service` class will provide a single programming interface for the applications developers to leverage security-related services within their application.

The security services class is configured using a DOM (most likely persisted as an XML document) to contain the information about the concrete classes and their parametric configuration information. This can be viewed as the venerable configuration file. When we place this information in the context of a larger framework, this XML information can be thought of as a declarative language that defines the runtime behavior of the application. This concept of declarative assembly exists in other areas such as Jakarta Struts and EJB assembly descriptors. It is the next logical step past external configuration files and is a powerful concept that can be used in application security and frameworks in general.

This implementation of declaring our security implementation means that security technologies can be changed with no impact to the application. This satisfies our goals of externalizing the implementation and providing a technique that allows us to ship a security implementation that is customized for different customers without having an impact on the development process.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

SECURITY IN A BROADER CONTEXT

Given the topics discussed so far within this document, we have a technique that will allow the applications developer to instantiate and call credentials gathering, authentication and authorization methods. This is accomplished through factories so that the concrete implementation is decoupled from the applications code. One of the stated implementation goals is to remove the need for the applications developers to be deeply involved in the development of the security code for the application. The business requirements should state the resources and actions that require authorized access, but there is a strong desire to not have that code pervade the application. How do we accomplish this?

Organizations that practice architecture-centric development most likely have an applications framework that they have obtained from one or more of the open-source communities or they developed an in-house framework. These are typically built on an MVC design pattern³. The controller within the MVC implementation is responsible for managing the flow within the application based on external events from the user interface, timings, network events, etc. Since this code acts as a dispatcher, it is an ideal place to put authorization logic.

Before the controller dispatches an action to the view or model components, it can invoke an authorization check. This removes the needs for explicit programming of authorization calls within the application as long as the granularity of trips through the controller is at least as fine-grained as the granularity of control required for authorization. In most cases that I have been exposed to, the granularity issue works out and there is no need for direct authorization calls within application logic. If there is a need for very fine-grained security within a part of the application, the developer can still make an explicit call to the security services for an additional authorization check whose implementation is still an abstraction.

If the controller framework provides for the declarative assembly of model and view parts, then there is an opportunity to declare the authorization requirements for those model and view parts. Pluggable security is powerful in its own right, but when coupled with a suitably powerful controller framework, the applications programmer can be all-but-completely removed from involvement in the application security. That liberates those developers to focus on the business functionality of the application.

3 MVC stands for “model-view-controller”. It is a design pattern that separates the user interface of an application (the “view”) from the underlying business model (the “model”). Requests are dispatched between the view and the model by use of a controller that serves as a binding layer.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

MIGRATION STRATEGY

Most enterprises have applications that have implemented a variety of security mechanisms. This approach is suitable for new applications development, but it also must be analyzed for how well it can be implemented within existing applications.

Without an analysis of the application, it is difficult to gauge the scope of effort for a retrofit. That said, this approach was developed with existing applications in mind. The expectation is that applications will have explicit calls to an implementation-specific authorization and authentication programming interface. That code that makes the implementation-specific calls can be factored out and reimplemented using the `Credential-Assembly`, `Authentication` and `Authorization` interfaces. The place in the applications code that used to make the explicit call is replaced with a call to the factory to instantiate the concrete implementation.

In this first implementation step, nothing has functionally changed. All we have done is to open the application for future changes in the implementation with no impact to the application by interjecting the abstraction. The next implementation step is to replace the explicit implementation with one that is required of the enterprise or the customer with a differing security implementation.

In most cases, authorization is fairly easy to replace in this manner. If the existing application implemented authorization with the “if the user has this role, then they are allowed” logic instead of the “gatekeeper” approach outlined in the key concepts, then the retrofit for authorization can be a little more difficult. It still may be possible to move that “if logic” into an `Authorization` interface to interject the abstraction into the application.

The other challenge that may present itself is that the application may not have an explicit lifecycle management approach. Prior to the execution of the application the factories have to be instantiated and populated in order for the security objects to be constructed. For applications that are implemented as servlets, this does not present a problem because the servlet specification (version 2.3 and above) support lifecycle listeners for web applications to declared in the `web.xml` file.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

SERVER-TO-SERVER SECURITY

Unattended authentication of a server process presents an interesting challenge. We typically think of credentials for authentication as “something you know.” Two-factor authentication takes this one step further such that authentication requires “something you know” and “something you have.” Two-factor authentication is usually implemented with a password that is memorized and a software or hardware device that can generate the remainder of the password.

Since we don't want to prompt for “something we know”, all we are left with is to authentication with “something we have.” This opens the possibility for the security token to be stolen from the machine on which the software server is running and a hacker to impersonate the service. The challenge is to determine a way increase the security of a resident credential against an independent security store using some of the concepts from two-factor authentication, but without requiring human intervention.

OBTAINING OS CREDENTIALS

In both Windows and Unix implementations, processes run with an authenticated user. This basic information – the SID⁴s in Windows and the userid/group information in Unix – is to create part of the credentials. These credentials can then be authenticated against an external security service.

In Java, information about the currently authenticated user is available in the Sun Microsystems JDK through the use of JAAS login modules that return this information as `Principal` objects placed within the `Subject`. That same information is available to C applications via operating system APIs. The information from those calls as well as environmental information, such as the local TCP/IP address, is placed into a message digest to create a hash value. That hash value is part of the password that is used to ultimately authenticate the service.

This is a good start to server authentication, but these artifacts from operating system authentication are fairly public, especially in the case of Unix, so authentication would be fairly easy to spoof.

4 Short for “security IDs”

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

ADDING PUBLIC/PRIVATE KEYS

In order to make the credentials for authentication, our hash password, harder to obtain, we need to add cryptography to the equation. The operating system artifacts that we used to create the hash value is analogous to “what we know” in two-factor authentication. A private key can serve as “what we have.”

A private key is generated and stored in the local file system. That key is used to encrypt the hashed password before it is sent to the server for authentication. This augments the what we know about the machine with the key, which we must have in order to authenticate. The server has access to the companion public key, which is stored in a central data repository, such as an LDAP server.

AVOIDING REPLAY ATTACKS

This step brings us closer to secure unattended authorization, but if the authentication process occurs over a clear-text transport such as TCP/IP without SSL/TLS, then there is a possibility of obtaining the encrypted credentials from the network and authenticating via a replay attack.

We can dramatically reduce the possibility of a replay attack by adding a time element to the credentials that are sent to the server for authentication. If we change our implementation to add a time stamp value to the hashed password before encrypting it with our private key, then the server can check the time stamp after decrypting the credentials and verify that the credentials were assembled recently, say within the last few seconds. This reduces the possibility of a replay attack to the few seconds in which the credentials are validated. This implementation gives us a fairly simplistic one-time password mechanism.

NETWORK VALIDATION

We included the TCP/IP address as part of the hashed password for authentication against the server. If the private key was compromised and the operating system credentials and the TCP/IP address of a machine was known, it would be possible for another machine to assume the identity of another.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

We can add a network element to the authentication process by moving the hashing of the TCP/IP address to the server side. That is, the client-side process computes a hash password that contains the operating system artifacts, but not the local TCP/IP address. The time stamp is added and the hash is encrypted as noted above.

When the server receives the credentials from the network, the server will decrypt and check the time stamp. However, before authenticating the hashed password in the security store, it hashes in the TCP/IP address of the connecting client and then validates the information.

Instead of a hacker needing to know the operating system artifacts and the TCP/IP address of the machine that they want to impersonate and obtaining the private key, they also have to be able to spoof the TCP/IP address of the machine.

This increases the security by splitting the credentials and requiring a greater degree of sophistication by a would-be hacker. It also has some additional complexities for some environments. If the client-side process goes through NAT translation before reaching the server, then the hashed password must be constructed with the NAT address instead of the real TCP/IP address of the client process.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

CHALLENGES

Application security is a very complex subject. Significant progress has been made in the last three years in implementing pluggable application security. The concepts discussed so far are instantiated in code and are in production in some organizations.

As the IT industry moves toward greater interoperability with both internal applications and in partnerships with other enterprises, the challenges increase significantly. It is safe to say that the work is not complete. The following are areas that need additional study and additional code.

VIEW FILTERING

The user interface should reflect the capabilities that a given authenticated user is allowed to perform. That is, if the user is not allowed to delete items from a database, the menu option or button to delete items should not appear on the user interface. There is clearly a relationship between security and the view layer, but *they are not the same thing*.

One approach would involve making calls to the security services while the user interface is being constructed. The code that builds the view would recover from security exceptions by not including those user interface elements. This seems like a brute-force technique, but it should work and would suffice in keeping the security implementation distinct from the view rendering within the application.

I think a better implementation would be to implement a similar kind of declarative structure around the code that is rendering the view. The security context would be passed to a framework that inspects that security context to make a boolean determination (as opposed to throwing exceptions) as to whether the user interface element should be rendered.

Although the implementation of authorization is abstract enough to not require it, I recommend an approach that authorizes based at a permission level. This is the approach taken in the Java 2 security model⁵. The implementation of view filtering could possibly be accomplished with a level of granularity at role or group membership. Ultimately, it is quite likely that the implementations for view filtering and authorization will access the same persistent store.

5 Sometimes called “JAAS Security” for “Java Authentication and Authorization Services”

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

As a side note, I view this much like input validation. Input is often checked for syntactic correctness at the user-interface level. The implementation of the business model should also validate the input, this time for both syntactic and semantic correctness. While this creates a redundant effort in that the input is syntactically validated twice, to assume the input is valid would create an undesirable coupling between the user interface and the business model. View filtering is similar. If the user doesn't have the "delete button", it may be difficult or (theoretically) impossible to delete an item from the data store. However, it would be inappropriate to not check authorizations before invoking the business function to delete objects from the data store.

EXTERNAL APPLICATIONS WITH CLOSED SECURITY

Every enterprise has off-the-shelf applications that have implemented security that is not replaceable. For example, there may be a desire to use a web-based e-mail application that resides on a remote service. The implementation of security is inaccessible beyond the login page that challenges for a userid and password.

One possible solution is to have the local authentication process provide embedded credentials. The authentication to the remote system would be to extract the credentials for the remote system from the private credentials within the security context that was authenticated locally. Those credentials could be presented in response to the challenge from the remote system.

The main problem here is that the remote credentials are shadowed in the local security store. This creates an issue with keeping that information synchronized. There are inelegant solutions that would respond to a remote authentication failure with a facade challenge that repopulates the local store and re-presents the new credentials to the remote system.

FEDERATED IDENTITY MANAGEMENT

If it wasn't challenging enough to get the internal systems all working with a pluggable security implementation that shares an implementation, modern IT systems are starting to collaborate across enterprises. This has been an issue for some time, but the urgency is increasing with the emergence of web services.

SECURITY IMPLEMENTATION STRATEGY

CAPABILITIES AND CHALLENGES

There are existing standards that try to accomplish the secure interoperation of services across the global Internet. I have confidence that those standards, such as SAML⁶, can be modeled within the interfaces discussed in this document in such a way that keeps the details abstracted from the applications developers implementing web services.

Author Information

*Gary Murphy
Hilbert Computing, Inc.
13632 S. Sycamore Dr.
Olathe, KS 66062*

*e-mail: glm@hilbertinc.com
voice: (913) 780-5051*

⁶ SAML stands for “Security Assertion Markup Language”. It is an open XML grammar that allows for authenticating users a in federated environment and to exchange authorization assertions and attributes between remote systems.