2007

SCAAS: A Secure Authentication and Access Control System for Web Application Development

Drew Hwang  
*California State Polytechnical University Pomona*

Wendy Wang  
*San Jose State University*

Blake Politte  
*Southern California Edison*

Follow this and additional works at: [https://scholarworks.lib.csusb.edu/ciima](https://scholarworks.lib.csusb.edu/ciima)

Recommended Citation  
Available at: [https://scholarworks.lib.csusb.edu/ciima/vol7/iss1/6](https://scholarworks.lib.csusb.edu/ciima/vol7/iss1/6)

This Article is brought to you for free and open access by CSUSB ScholarWorks. It has been accepted for inclusion in Communications of the IIMA by an authorized editor of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
ABSTRACT

User authentication and data access are becoming two of the most common areas for web attacks. Most security vulnerabilities occur in areas of coding where Web security has lapsed. This paper describes the design and development of a Secure Authentication and Access Control System (SCAAS) implemented as a reusable library that provides data driven and encryption based authentication and access control for the use with ASP.NET applications.

INTRODUCTION

Web sites today face many threats to the confidentiality and integrity of the data used and the functionality provided by the application. This problem is compounded by the fact that Web developers are simply lack of either adequate knowledge and skills in writing secure Web application codes (Huang et al., 2005) or sufficient testing methodologies for the audit and control of Web development (Mansour and Houri, 2006). Works in the design and implementation of security measures for Web applications are greatly in need.

User authentication and data access are becoming two of the most common areas for web attacks when procedures such as single sign-on and authentication delegation have become practically indispensable for e-business environment (Paulus, 2001). These two types of on-line vulnerability can be counterattacked by securing user account database that opens the gate of the application and by encrypting SQL connection that leads to the data store.

This paper describes the design and development of a Secure Authentication and Access Control System, herein referred to as SCAAS, implemented as a reusable library that provides data-driven and encryption-based authentication and access control for the use with ASP.NET applications. SCAAS employs Microsoft SQL Server to persist the security definitions that the SCAAS run-time system utilizes. The SCAAS database will be herein referred to as the SCAAS User Registry. The system also provides an ASP.NET based administration application that is used to maintain the data in the SCAAS User Registry.

SCAAS COMPONENTS

SCAAS consists of four major components. Their definition and functionalities are described as follows:
SCAAS Framework

This is the core of the SCAAS run-time application and a .NET library written in C#. Included in the namespace are four classes that make up the SCAAS Framework: SCAASManager, SCAASManagerHelper, SCAASDataProtector, and SCAASException. These classes will be further discussed in Section 3.

SCAAS User Registry

This is a Microsoft SQL database named UserAccounts that provides the basis for the SCAAS User Registry. The SCAAS Framework works closely with the UserAccounts database. Any connectivity between the SCAAS framework and the UserAccounts database is done securely with .NET enabled encryption and decryption procedures. The SCAAS User Registry can be updated through the SCAASAdmin ASP.net application included in the system.

SCAAS Admin ASP.NET Application

This is the ASP.NET application developed to update the SCAAS User Registry. This application utilizes the FormsAuthentication mode of the SCAAS Framework. Because of this, the application also serves as a good example of an implementation of the FormsAuthentication mode of the SCAAS framework.

DPAPIClientWeb ASP.NET Application

This is the utility application that is vital to get the SCAAS run-time to operate correctly. This ASP.NET application is used to generate encrypted connection strings used by both the SCAAS run-time as well as client applications that wish to use the SCAAS secure database connection management SCAAS API.

THE SCAAS FRAMEWORK

The SCAAS framework is the core component of the SCAAS system and is based on Microsoft’s Forms Authentication model for authentication and authorization of ASP.NET applications. Microsoft’s Forms Authentication model is not a complete security solution but rather the bits and pieces required to be built upon. A key component of the model is the System.Web.Security namespace included in the .NET framework. This namespace includes several classes, enumerations, and delegates that can be used to develop secure ASP.NET applications (Curphey, 2003). However, implementing this model requires a lot of customization in ASP.NET applications. For example, when using the authentication aspects of the model, a Web developer must create a user defined User Registry. Furthermore, if more than one application were to be using the same custom security implementation, it would be prudent to abstract that functionality to an independent library to gain the leverage of reusability. This is exactly what the SCAAS Framework was built to achieve: a security model implemented in a reusable library.

There are two operational modes that SCASS offers: the FormsAuthentication mode and the PassiveAuthentication mode. The FormsAuthentication mode accommodates the traditional need to secure all pages of an ASP.NET application. It is the mode to be used in an “all or none” fashion, where only one page can be offered for everybody to view without being authenticated. Typically, this would be the “Logon.aspx” page. While this is certainly a secure scenario, it can sometimes be inefficient in building recurring Web applications. In other words, some applications require that a particular page can be viewed in a “generalized” manner while running in a “specialized” manner once a user is authenticated. Think of the “specialized” state as being one of an elevated privilege level depending on who is running the application and whether or not they have been authenticated. For example, consider an application that displays product prices to a retail user. If the user has logged on as a member of a certain role such as the “WholesaleUser” role, the page will display prices with a 20% discount. The SCAAS Framework can accommodate this sort of requirement by using the PassiveAuthentication mode. This mode is to be used when the need arises to selectively elevate the privilege level for a web application user to offer a “specialized” view or functionality of the application.
The SCAAS Framework is housed in the SCAAS .NET namespace which contains four classes. Figure 1 shows a conceptual model of the four classes with the SCAAS components and databases. The functionalities of these four classes are discussed as follows:

**SCAAS.SCAASManager**

This is the core class of the SCAAS Framework functionalities. Most of the authentication, authorization, role setting, credential management, exception generation, database connection management, and other core functions are found in this class. The two core methods in the *SCAASManager* for secure database connection management are *GetSqlConnection* for Microsoft SQL Server and *GetOleDbConnection* for other generic databases such as Oracle, Microsoft Access, and so on. A Web developer can use this class for secure database connection by hiding plain text connection strings behind encryption. The two methods work in concert with the encrypted strings generated by the *DPAPIClientWeb* ASP.NET application.

**SCAAS.SCAASHelper**

This is the developer’s helper class that groups functionalities of the *SCAASManager* class and simplifies the integration of the SCAAS Framework into an ASP.NET application. The *SCAASManagerHelper* class has some key methods such as Logon, Logout, Authentication Processing, Error Handling, and SCAAS Application Initiation. Also, the class constructor is where the determination of whether the *FormsAuthentication* mode or *PassiveAuthentication* mode is implemented.
SCAAS.SCAASException

Many different exceptions can be generated in the SCAASManager class and subsequently handled by the SCAASManagerHelper class. SCAAS attempts to reframe as many exceptions as possible by initiating some variant of a SCAASException class. The key to the SCAASException class is the internalized SCAASExceptionType enumeration and the declaration of private variable of this type. When an exception is generated in the SCAASManager class, this enumeration is always set in the SCAASException class for error processing to occur at a higher level from the thrown exception. Again, much of this is automatically handled by the SCAASManagerHelper class, but a developer working directly with the SCAASManager class will need to be aware of this custom exception generation mechanism.

SCAAS.SCAASDataProtector

This class is generally used for the secure database connection mechanism for SCAAS. In particular, the DPAPIClientWeb ASP.NET application uses this class to generate encrypted connect strings for the web.config file, and the SCAASManager database connection methods also use this class to provide secure database connections by reading the same encrypted strings created with the DPAPIClientWeb ASP.NET application. This class leverages
the Windows DPAPI (Crypto32.dll) security framework and contains two primary methods: the CryptProtectData method and the CryptUnprotectData method for the data encryption and decryption procedures.

IMPLEMENTING SCAAS FRAMEWORK

Implementing the SCAAS framework requires the SCAAS User Registry, the SCAASAdmin ASP.NET application, and the DPAPIClientWeb ASP.NET application to be properly installed and configured.

The SCAAS User Registry is implemented as a Microsoft SQL Server based database named UserAccounts. Note that the SCAASAdmin ASP.NET application and the SCAAS User Registry don’t have to be installed on the same machine running the SCAAS compliant ASP.NET applications. However, it is a must that the DPAPIClientWeb ASP.NET application should be on the same machine as the SCAAS compliant ASP.NET applications, because this is a system requirement of the underlying Microsoft DPAPI technology as the encryption/decryption algorithm using the DPAPI is machine specific.

Installing the SCAAS User Registry first requires the UserAccounts.mdf and UserAccounts_log.mdf of the SCAAS system to be attached to the local machine running a Microsoft SQL Server. Using the SQL Server Manager, a Web developer will create an instance level user named “SCAASAllPrivs,” add this user to the UserAccounts database, and create a Role to the UserAccounts database named “SCAASAllPrivs_Role”. Once the role is available, the Web developer will grant the SCAASAllPrivs user to the SCAASAllPrivs_Role role and all privileges for the UserAccounts database to the SCAASAllPrivs_Role.

The DPAPIClientWeb ASP.NET Application also needs to be installed on every machine running SCAAS compliant ASP.NET applications because of the requirements for DPAPI encryption/decryption procedures. Folder that houses the application needs to be registered with the Internet Information Services server.
Finally, the SCAASAdmin ASP.NET Application needs to be installed to maintain the SCAAS User Registry. There is one unique configuration step to this application and all SCAAS compliant applications to be developed using the SCASS system. Because the SCAASAdmin ASP.NET application is itself a SCAAS compliant application (using the FormsAuthentication mode), one encrypted string must be created for secure database connections by running the string encoder page of the DPAPIClientWeb ASP.NET application. As shown in figure 2, after correct parameters for the SQL database name, SQL database server name, database user name (SCAASAllPrivs), and password are entered and an appropriate database type is chosen, the system produces an encrypted connection script that is unique for every machine. The string then needs to be saved for the web.config file of the SCAASAdmin application. Here is an example of the <appSettings> section of the Web.config file for the SCAASAdmin application:

```xml
<appSettings>
    <add key="SHOW_ALL_ERRORS" value="true"/>
    <add key="SCAAS_SESSION_TIMEOUT" value="30"/>
    <add key="SCAASAllPrivs_UserAccounts" value="AQAAANCMnd8BFdERjHoAwE/CIsBAAAAAnCMyA6xa0ysQB7Dl8ePaAQAAAAAC"/>
</appSettings>
```

Take notice of the key located in the <appSettings> section in the entry of <add key="SCAASAllPrivs_UserAccounts value="">. What it dictates is that the value in the <appSettings> section of the entry <add key="SCAASAllPrivs_UserAccounts value=""> is the encrypted connection string that the SCAAS
Framework uses to create a SQL Server connection for the SCAASAllPrivs user for the UserAccounts SqlServer database. This is a key concept not only for the SCAASAdmin ASP.NET application but also for all applications that are using encrypted connection strings.

Once the above three system components are installed and configured for the SCAAS framework, the Web developer needs to determine which mode the application is to be implemented between the FormsAuthentication mode and the PassiveAuthentication mode. The key to using a particular mode lies in the SCAASManagerHelper constructor which is implemented in the Global.asax file as part of any ASP.NET application. In fact, outside of the actual programming logic involved to secure the application, making the application SCAAS compliant is merely a matter of implementing the correct structure of the Global.asax file and the Web.config file and adding the application’s users/roles using the SCAASAdmin ASP.NET application.

For FormsAuthentication mode, the Global.asax with C# code file will look like this:

```csharp
Protected void Application_Start(Object sender, EventArgs e)
{
    // Initialize all SCAAS specific
    HttpApplicationState level items
    SCAASManagerHelper.ApplicationStartWrapper
        (this.Context.Application);
}

Protected void Session_Start(Object sender, EventArgs e)
{
    // Initialize all SCAAS specific
    HttpApplicationState level items
    SCAASManagerHelper.SessionStartWrapper
        (this.Session);
}

Protected void AuthenticateRequest(Object sender, EventArgs e)
{
    // Implement the FormAuthentication SCAAS
    security mode
    SCAASManagerHelper scassManagerHelper = New
    SCAASManagerHelper(SCAASManagerHelper.
        FormsOrPassive. FormsAuthentications);
}
```

The web.config section concerned with authentication and authorization will look like this:

```xml
<authentication mode="Forms">
    <forms loginURL="Logon.aspx" name="SCAASAdminAuthCooki" Timeout="30" path="" />
</authentication>

<authorization>
    <deny user="?" />
    <allow user="*" />
</authorization>
```

Note that the setting “Logon.aspx,” under FormsAuthentication mode, is the only page that will be able to be viewed by the users until a successful authentication is transacted. In addition to handling logon sequences, this page will also host all the error handling, logout messaging, and other SCAASException driven events.

Lastly, the web.config section concerned with application specific settings will look something like this:

```xml
```
<!—Custom settings for FormAuthentication-->  
<appSettings>
 <add key="SHOW_ALL_ERRORS" Value="true" />
 <add key="SCAAS_SESSION_TIMEOUT" value="30" />
 <add key="SCAASAllPrivs_UserAccounts"
   value="AQAAANCMnd8fDerOW..." />
 <add key="MyAppAllPrivs_MyDatabase" value="AQAAANCMnd8fDerOW..." />
</appSettings>

Note that because the application is a SCAAS compliant application, it will always need the SCAASAllPrivs_User Accounts encrypted string. The SHOW_ALL_ERRORS settings will enable more detailed error reporting on exceptions. The Web developer would want its value to be "true" for development but likely set it to "false" for a production implementation.

For PassiveAuthentication mode, the Global.asax file with Visual Basic code will look like this:

Sub Application_Start(ByVal sender As Object, e As EventArgs)
  // Initialize all SCAAS specific
  HttpApplicationState level items
  SCAASManagerHelper.ApplicationStartWrapper(Application)
End Sub

Sub Session_Start(ByVal sender As Object, e As EventArgs)
  // Initialize all SCAAS specific
  HttpApplicationState level items
  SCAASManagerHelper.SessionStartWrapper(Session)
End Sub

Sub Application_AuthenticateRequest(ByVal sender As Object, e As EventArgs)
  // Implement the FormAuthentication SCAAS
  security mode
  Dim scassManagerHelper As SCAASManagerHelper =
  New SCAASManagerHelper(SCAASManagerHelper.FormsOrPassive.FormsAuthentications)
  scassManagerHelper.ApplicationAuthenticateWrapper(Context)
End Sub

The web.config section concerned with authentication and authorization will look like this:

<!— Setting for PassiveAuthentication-->  
<authentication mode="None" />
<!— Setting for PassiveAuthentication-->  
<authorization>
  <allow user="*" />
  <allow user="" />
</authorization>

Obviously, the Global.asax and the section in the Web.config are significantly different from those for a FormsAuthentication application. In short, any page can be viewed regardless of whether a user is authenticated or not in the PassiveAuthentication model.

Finally, the web.config section concerned with application specific settings will look like this:

<!—Custom settings for PassiveAuthentication-->  
<appSettings>
 <add key="SHOW_ALL_ERRORS" value="true" />
 <add key="SCAAS_SESSION_TIMEOUT" value="30" />
</appSettings>
Like the *FormsAuthentication* example, the SHOW_ALL_ERRORS settings will enable more detailed error reporting on exceptions. The value needs to be set to "true" for development but would likely be set to "false" for a production implementation. However, unlike the *FormsAuthentication* mode, the *PassiveAuthentication* mode can name any page needed for the error handling page. This is because under *PassiveAuthentication* mode, the Web developer will not be bounded by only one page being free to view despite authentication.

The final step in implementing the SCAAS framework is to add users/roles with the SCAASAdmin ASP.net Application. Internal to the SCAASAdmin ASP.net application there are four roles: *SCAASAdmin*, *RoleAdmin*, *UserAdmin*, and *PasswordAdmin*. Once logged on to the SCAASAdmin ASP.net application, the Web developer can add roles, delete roles, add users, delete users, grant roles to users, and revoke roles from users.

For instance, assume the application to have two roles such as *WholesaleUser* and *WholesaleAdmin*. Before the application is coded, the Web developer needs to thoroughly understand what these two roles mean in the context of the application usability and design. The *WholesaleUser* role may mean that the user will get a certain type of discount on selected products or quantities. It may also mean that the user will access pages other than those non-authenticated users aren’t allowed to see. The idea that non-authenticated users can see any part of the application implies that the application will be using the *PassiveAuthentication* mode of the SCAAS system. In addition, the Web developer may want one page available for only users that have the *WholesaleAdmin* role. This may be used for special users that are designated to alter prices on certain products and quantities for other logged on users that have the aforementioned *WholesaleUser* role.

Because the SCAAS Framework is tightly integrated with Microsoft’s Forms Authentication model (not to be confused with the SCAAS FormsAuthentication mode), many of the inherited ASP.NET features are needed to work with the SCAAS Framework. The key to making SCAAS Framework a security implementation is found with the ASP.NET’s intrinsic object called *User*. The intrinsic *User* object is actually a member of the HttpContext class, and the ASP.NET intrinsic object *Context* is an instance of an HttpContext class that the ASP.NET Framework also automatically provides (hence the word “intrinsic”). Examples of the *User* object methods Include *User.Identity.IsAuthenticated()*, *User.Identity.Name()*, and *User.IsInRole()*. For example, if a user logged on and was granted the *WholesaleUser* role by the SCAASAdmin ASP.net application, then the *User.IsInRole("WholesaleUser")* would return true or false. It can be easily seen how this can be used in the SCAAS compliant applications. Another example is the utilization of the *User.Identity.IsAuthenticated()* method. The application can completely control access to a particular page by just calling this method in the PageLoad() ASP.NET event handler. If true, allow access. If not, redirect to the error page with the proper SCAASException.

As the users/roles have been added and the roles have been assigned to the users with the SCAASAdmin ASP.net application, Web developers can now use SCAAS to enhance the security of their Web applications through secured authentication and access control.

**CONCLUSIONS**

Recent computer programming languages have enabled more effective development of secure web applications. For instance, Java technology allows one to construct applications by using a large set of APIs, tools, and implementations of commonly used security algorithms, mechanisms, and protocols. Microsoft’s recent .NET technology also provides a variety of security features commensurate with the breadth of the framework itself.
SCAAS presents a security implementation that leverages the .NET intrinsic libraries and makes the implementation a reusable component.

REFERENCES


