1. **Audit Implementation Goal: To maintain a persistent security log across all user sessions including administrative.**
2. **QADashboard Security Requirements Summary Posted Thu 11/2/2006**
   1. We want to have a "hidden" page on the QA dashboard that can only be accessed by typing in the URL, and that will have restricted access only to site administrators. This page will contain auditing data, such as who has accessed the site and when.
   2. It ought to contain a table showing which accounts have accessed the site and on what date they did so.
   3. From there, we might expand to show which pages on the site a given account accessed.
   4. You can presume the site will use domain authentication to control access both to the main dashboard and to the admin page.
3. **QADashboard Security Requirements Summary Additional Consideration, posted Fri 11/3/2006 after studying** [**Forms Authentication with Active Directory in ASP.NET 2.0**](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/PAGExplained0001.asp)
   1. Specify hidden QADashboard administrator URL as http://<servername>/QADashboard/DashBoardAdmin.aspx.
   2. Have the DashBoardAdmin.aspx page locked out by removing anonymous access to the page in IIS or Implement the ActiveDirectoryMembershipProvider API to identify who is administrator and who is just a user. The admin credentials will be known only to whomever will bear the responsibility as QA Dashboard Administrator.
   3. QADashboard currently loads the Default.aspx page automatically when a user navigates to the QADashboard web application root. This will be augmented with a forms Login that will authenticate against the ActiveDirectoryMembershipProvider API as a datasource and store as strongly encrypted stateless, session-less, non-persistent HTTP Cookie if an authenticated user return back to QADashboard within a 24 or 48 period.
   4. Another possibility I am researching is to have the security logic of Default.aspx automatically detect the everyone’s Domain credentials through the ActiveDirectoryMembershipProvider API much the same way that <http://autosecure> does.
   5. **Maintaining a persistent security log across all user sessions including administrative activity is the solitary goal of this audit implementation.**
   6. Through the API credential tracking I do not see any issue with writing the implementation of what user’s call which page’s Page\_Load() event handler now instead of later.
4. **Requirements Summary Additional Consideration, Posted 11/4/2006 after studying** [**Windows Authentication in ASP.NET 2.0**](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/PAGExplained0001.asp)**,** [**ADAM for Roles in ASP.NET 2.0**](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/PAGExplained0001.asp)**, and** [**Instrument ASP.NET 2.0 Applications for Security**](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/PAGExplained0001.asp)
   1. The hidden DashBoardSecurity.aspx security monitoring page should not be a child page of the original MasterPage.master. It should provide its own set of administrative site activity monitoring/auditing.
   2. Actual user application role management should be handled only through the Authorization Manager mmc snap-in. This should be so to prevent anyone from “accidently” stumbling across the admin monitoring page and then finding a way to access a control that can block the access of another user.
   3. The security implementation is more securely and efficient done by detecting the domain member user through the ActiveDirectoryMembershipProvider to see how each user authenticates based on the Redmond Active Directory. This eliminates the need for ever having to use a login control since all users are on corpnet – everyone just authenticates as themselves.
   4. Have a local or remote data store of Groups Roles and User Roles at the application level that are custom defined and configurable through Authorization Manager MMC snap-in and AzMan Policy store in Active Directory Application Mode for the potential to authorize how will get to access which page resource.
   5. This sort of approach is clearly expandable to include the added security feature of detecting who accessed what page in the dashboard and when they did it.
   6. After understanding the potential security implications of using a security implementation that used HTTP Cookies or even Cookieless URI embedded authentication through Forms Authentication, it no longer made any sense to use having found several easy-to-Implement code samples for the ASP.NET 2.0 Role Manager authorizing through ADAM and authenticating through the Membership Provider. This done as a rebuttal against the earlier proposed requirement of 2.3.
   7. Instead of implementing the security in the code behind logic of Default.aspx, as proposed in section 2.4, the implementation should be completed behind both the MasterPage.master and also the DashBoardSecurity.aspx pages.
   8. A later version of the QADashboard may use implement the <http://autosecure> as a more streamlined approach for a Role Provider and Role Management mechanism. In this version, it should not be implemented.
   9. The goal statement in section 3.5 was placed in bold and moved to the top indicating the overall intention of what must be done for this project piece.
   10. The UI portion of the security should be carefully implemented considering this approach: It makes more sense to not use any login controls accept for an <asp:LoginStatus /> control to display what the user has authenticated as in the upper right portion of the website’s main MasterPage.master.
   11. The Admin page itself it not a child of the MasterPage.master and should have its own <asp:LoginStatus /> control so the administrator may see what account they have authenticated as.
   12. User Activity Data should be stored as flat file XML so that SQL Server is not required if the feature of tracking user activity to what resource is determined to be required as part of this version.
   13. If the user tracking feature is required, then clicking on an authenticated user should reveal their details and complete access history.
5. The total time to completion for the feature is guestimated to be between 2 business days and 3 business days based on what this feature developer has learned and barring any unforeseen problems not already encountered.