International Journal of Information Technology and Knowledge Management
July-December 2011, Volume 4, No. 2, pp. 679-694

E-MCES: SECURITY ISSUES AND PREVENTING HACKERS' ATTACKS

Ashok B. Kulkarni[1] & Valeri Pougatchev[2]

This paper briefly describes security issues and some preventing hackers' attacks techniques of the e-Management Control and Evaluation System developed by us at the University of Technology (UTech), Jamaica. As an Integrated Management System, e-MCES offers to the customers a wide spectra of services – Strategic/ Operational planning for the units and their staff, Evaluation of the Performance of units and Academic/non-Academic Staff as well as an Administrative/Ancillary personnel, and finance management of the institution. These services must be supported by solid technical and organizational solutions due a great confidentiality of data handled by the system. We describe main security principles implemented in the e-MES including authentication of users and navigation solutions. The paper describes our vision and proposed solutions to make the system invulnerable of any hackers' attacks of different malicious user's attempts to get and/or destroy data. We have made analysis of the possible injection attacks and have suggestions how to prevent the data in the system as well as preventing any attempts to destroy a structure of the system. We share these techniques with the Academic Society.

Key words: e-MCES, Security Issues, Hackers Attacks, Data Protection.

1. INTRODUCTION

The security is an essential part of any Integrated Management System (IMS) for educational institutions. The e-MCES, that we built for the UTech is an example of such Integrated Management Systems. It has been developed for all categories of staff of the University, including academic/ nonacademic, administrative, technical, ancillary staff with different areas of responsibility and the students. Some of the users of this system are managers and supervisors while others are employees or students – and all of them are members of the one university's community.

The e-MCES accumulates confidential information of staff evaluations, students' evaluations and feedback, suggestions, etc. Most of this information are anonymous in nature. The system serves all members of the university community, with a single Internet interface and common data storage. The system must protect the confidentiality, integrity, and availability of the customers' information, and the integrity and availability of processing resources, under control of the system's owner or administrator. A level of the system invulnerability must make it infeasible – even when using the system properly – to prevent an attacker from usurping privileges on the user's system, regulating its operation, compromising data on it, or assuming ungranted trust [1].

For educational institutions the last issue is quite important. We can recognize at least two categories of users which can carry out a potential security threat to the system - persons who are not satisfied with results of their evaluation and advanced students with high level of in discipline who are trying to 'implement' their knowledge in compromising or bypassing restrictions of the current university's information systems. The first group of users could try to destroy the consistency of the data in the system in revenge of their poor evaluation results. Another group (students) despite of their interests for obtaining the knowledge and skills in technologies are still kids. After obtaining of the beginning knowledge in the Database Management Methods and Programming techniques, they consider their illegal activity as a game, not as malicious efforts. These two types of threat motivations are a potential risk for the system. We make an analysis of possible threats and give recommendations for system protections of these attacks.

In this paper we trying to show how to build secure data access code and avoid common vulnerabilities and pitfalls. We present a series of countermeasures and defensive techniques that must be done in the system data access code to mitigate the top threats related to data access.

The security issues for the integrated educational management system are critical. Because the system serves all categories of the university staff using a single Internet interface and common data storage, it must establish very strong restricted access to the resources. Our solution is based on the concept of the specific Role of the user [2]. During process of Authentication and Authorization, the system recognizes the role of the logged-on user and grants him/her the permission. We associate this process with process of building a run-time menu - a unique menu for each logged-on user.

Actually, our system itself has no menu-system. Each time it automatically generates a Navigation menu for each

[1]Academic Affairs Division, University of Technology, Jamaica
[2]School of Computing & Information Technology, University of Technology, Jamaica
E-mail: [1]akulkarni@utech.edu.jm, [2]vpougatchev@utech.edu.jm

particular logged-on user and it is discarded, when the user logs-off, or the session for this user has expired or is terminated. Using this solution, the System Administrator does not need to modify the application or configuration files, if, for instance, it needs to add a new user with a new role, which is associated with some new permission or if it is necessary to institute a new position in the institution. It is important to note that this solution does not depend of size of the organization and number of different positions. We have developed an original menu builder, which can also be implemented in other information systems.

2. THREATS AND COUNTERMEASURES

The database is a prime target for application level attacks. Application level attacks are used to exploit vulnerabilities in your data access code to gain access to the database [3].

The top threats (Fig. 1) to data access code are:

- Structured Query Language[1] (SQL) injection
- Disclosure of configuration data
- Disclosure of sensitive application data
- Disclosure of database schema and connection details

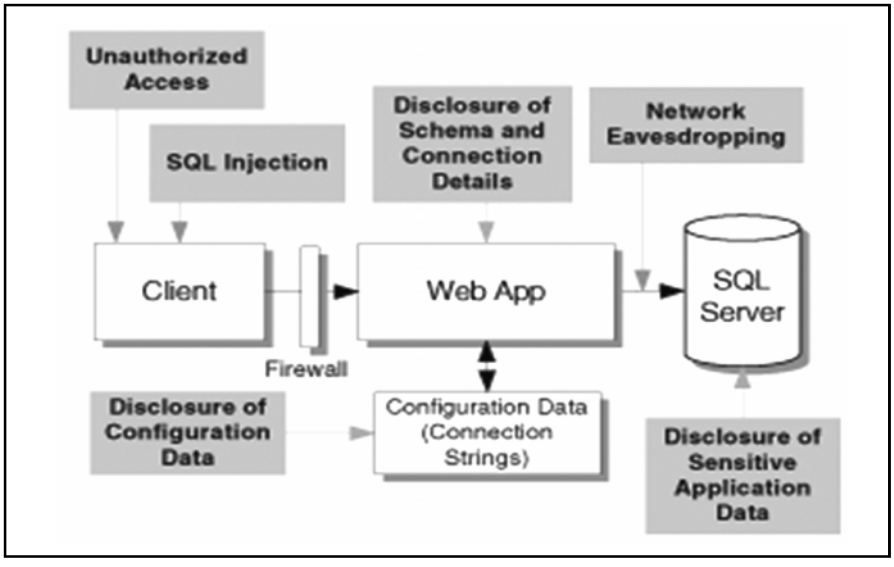


Fig. 1: Top Threats of the Internet/Intranet Application [3].

- Unauthorized access
- Network eavesdropping

We consider some of these threats and will present practical recommendations of solutions.

2.1 SQL Injection

SQL injection attacks exploit vulnerable data access code and allow an attacker to execute arbitrary commands in the database.

Common vulnerabilities that make our data access code susceptible to SQL injection attacks include:

- Weak input validation
- Dynamic construction of SQL statements without the use of type-safe parameters
- Use of over-privileged database logins

Researchers and practitioner developers recommend the following Countermeasures to prevent application of this type of attack:

- Constrain and sanitize input data.
- Use type safe SQL parameters for data access.
- Use an account that has restricted permissions in the database.

The easiest way for SQL Injection attacks is an Input data. Howard and LeBlanc [1] said: "All Input is Evil! If someone you didn't know came to your door and offered you something to eat, would you eat it? No, of course you wouldn't." So why do so many applications accept data from strangers without evaluating it? These questions define one of the most important rules in building robust, reliable application, which warrants high level of security. We must never blindly trust user input. Textboxes and Textareas controls are typical elements of the user interface of the application to obtain information from the user. These are "doors" for any data, typed by user. In simple terms, SQL Injection is the process of passing SQL code into an application, in such a way that it was not intended or anticipated by the application developer. This may be possible because of the poor design of the application, and it affects only applications that use SQL string building

[1]SQL is described in many resources. We recommend [4].

techniques to create a command with user-supplied values. Let us consider some examples.

Example 1: Getting illegal access to the application.

Let us consider fragment of C# programming code, which uses SQL statement (Fig. 2). This SQL query gathers data (UserID), entered by user to textbox txtID and check whether UserID matches to data, presented in the database. If "Yes", it will accept user and redirects him/her to the Login.aspx page. If "Not", it will generate warning message on the browser's form.

```
SqlConnection cn = new SqlConnection (Web
ConfigurationManager.ConnectionStrings
    ["eMCESConnectionString"].ConnectionString);
string sql =
    "SELECT COUNT(*) FROM UserInfo " +
    "WHERE UserName ='" + txtID.Text + "' ";
SqlCommand cmd = new SqLCommand(sql, cn);
cn.Open();
SqlDataReader reader = cmd.ExecuteReader();
if (reader.HasRows)
{
    lblMessage.Text = " ";
    s = redirect + "Login.aspx?U=" + sID;
    reader.Close();
}
else
 lblMessage.Text = "Warning: Sorry, but you are not a
                   member of the university
                   community or your status is
                   temporarily suspended";
```

Fig. 2: A Fragment of C# code, Which Matches Value of Textbox txtID in the Database

If user enters in txtID data – ANYDATA' OR '1 = 1', the SQL statement will be like this:

SELECT COUNT(*) FROM UserInfo

WHERE UserName = 'ANYDATA' OR '1 = 1'

Which, obviously, returns a value of function COUNT(*) > 1, because WHERE condition will generate a logical expression (False or True), which is True.

Another similar example is as follows:

Suppose the user has entered a valid user name of "Paul" and a password of "password". SQL statement becomes:

SELECT Count(*) FROM Users

WHERE UserName= 'Paul' AND Password='password'

But when the hacker enters

Or 1=1 —

the query now becomes:

SELECT Count(*) FROM Users

WHERE UserName= "Or 1=1 —' AND Password=''

Because a pair of hyphens designates the beginning of a comment in SQL, the query becomes simply:

SELECT Count(*) FROM Users WHERE UserName='' Or 1=1

The expression 1=1 is always true for every row in the table. It means a search condition will always return true. So, assuming there's at least one row in the Users table, this SQL will always return a nonzero count of records.

Example 2: Worse scenario - deleting database.

Using textbox user can enter data like this:

ANYDATA' OR '1 = 1'; DROP DATABASE eMCES

In this case SQL statement will be split on two statements:

SELECT COUNT(*) FROM UserInfo

WHERE UserName = 'ANYDATA' OR '1 = 1';

DROP DATABASE eMCES

These two statements are separated by legal, in terms of SQL syntax, symbol ";", which means just end of SQL statement. SQL engine will be executing second SQL query and will delete entire database eMCES!

Example 3: Malicious Connection String ("Connection string injection").

This example is presented in [5]. We might decide to prompt the user for credential – supplying text boxes that allow the user to supply a user name and a password – and then construct a connection string based on that input in a connection string with code such as the following:

```
string strConn;
    strConn = @"Data Source=.SQLExpress;"Initial
              Catalog=Northwind;' +
              "User ID =" + txtUserID.Text;" +
              "Password=" + txtPassword.Text + ";"
    Console.WriteLine("Resulting connection string: {0}",
strConn);
```

At first glance, this might seem like a safe and logical thing to do. Now consider a malevolent user who might like to change the connection string. Suppose the user's input information is as follows:

- To the txtUserID textbox:

  MyUserID;Data Source=EvilSourceName

- To the txtPassword textbox: - MyPassword

In this case user could change a server of the application.

Using previous code snippet and input information, presented above, the resulting connection string is as follows:

Data Source=SQLExpress;Initial Catalog = Northwind ;User ID=MyUserID;

Data Source= EvilSourceName;Password=MyPassword;

As we can see, the Data Source keyword is specified twice. Which value wins if we use the resulting connection string – the first one (local machine) or the second one (the "evil" server)? Is there are way to manually inspect the user input to identify possible connection string injections such as these? Are the answers consistent across other connection string keywords or across other .NET Data Provider? What can a poor programmer to do?

It's important to realize that the SQL injection attacks are not limited to SQL Server. Other databases, including Oracle, MySQL, DB2, Sybase, and others are susceptible to this type of attacks. SQL injection attacks are possible because the SQL language contains a number of features that makes it quite powerful and flexible, namely [6]:

- The ability to embed comments in a SQL statement using a pair of hyphens.
- The ability to string multiple SQL statements together and to execute them in a batch.
- The ability to use SQL to query metadata from a standard set of system tables.

SQL injection attacks are not limited to ASP.NET applications. Classic ASP, Java, JSP, and PHP applications are equally at similar risks.

3. A STRATEGY FOR DEFENDING AGAINST MAIN THREATS

In view of possible hackers' attacks, we have described in previous sections, we have reduced a number of Input dialog boxes in e-MCES. In the system, we have only two types of such textboxes – for Login Form and several dialog boxes, where appraiser/appraisee leaves his/her comments and recommendations. From one point of view, this is more secure, from other methods – it simplifies a user interface of the system. Instead of input textboxes, we offer comboboxes, listboxes, checkboxes and other types of ASP.NET built-in controls with predefined contents of data, and without any opportunity to user to modify. All the user has to do is – just to select one or several items, offered by these controls, without typing any data.

In e-MCES (version 1.1.1) we have filter routines that add escape characters to characters that have special meaning to SQL, such as the single apostrophe character and remove other special characters, using code, as shown here:

```
string s = s.Replace("'", "''");
s = s.Replace("\\", "");
s = s.Replace("/", "");
```

The problem with routines such as this and the reason why developer should not rely on them completely is that an attacker could use ASCII hexadecimal characters to bypass our checks. We should, however, filter input as part of our in depth defense strategy.

Litwin recommended following actions to prevent SQL injection attacks [6]:

Table 1
Preventing SQL Injection Attacks

Principle	Implementation
Never trust user input	Validate all textbox entries using validation controls, regular expressions, code, and so on.
Never use dynamic SQL	Use parameterized SQL or stored procedures.
Never connect to a database using an admin-level account	Use a limited access account to connect to the database.
Don't store secrets in plain text	Encrypt or hash passwords and other sensitive data; you should also encrypt connection strings.
Exceptions should divulge minimal information	Don't reveal too much information in error messages; use customErrors to display minimal information in the event of unhandled error; set debug to false.

The SQL injection attacks we have demonstrated in previous section are all dependent on the execution of dynamic SQL statements, which are constructed by the concatenation of SQL with user-entered values. Using parameterized SQL, however, greatly reduces the hacker's ability to inject SQL into programming code. A fragment of this solution using C# code is presented here:

```
private void cmdLogin_Click(object sender, System.EventArgs e)
{
    string strCnx = ConfigurationSettings.AppSettings ["eMCE Sconfig"];
    using (SqlConnection cnx = new SqlConnection (str Cnx))
    {
        SqlParameter prm;
        cnx.Open();
        string strQry =
            "SELECT Count(*) FROM Users WHERE UserName =@ username "+
            "AND Password=@password";
        int intRecs;
        SqlCommand cmd = new SqlCommand(strQry, cnx);
        cmd.CommandType= CommandType.Text;
        prm = new SqlParameter("@username", SqlDbType. VarChar,50);
        prm.Direction=ParameterDirection.Input;
        prm.Value = txtUser.Text;
        cmd.Parameters.Add(prm);
        prm = new SqlParameter("@password",SqlDbType. VarChar, 50);
        prm.Direction=ParameterDirection.Input;
        prm.Value = txtPassword.Text;
        cmd.Parameters.Add(prm);
        intRecs = (int) cmd.ExecuteScalar();
        if (intRecs>0) {
            FormsAuthentication.RedirectFromLoginPage (txtUser. Text, false);
        }
        else {
            lblMsg.Text = "Login attempt failed.";
        }
    }
}
```

A parameterzed command is simply a command that uses placeholders in the SQL text.The placeholders are then added separately and automatically ended [7]. In e-MCES we use this techique instead of dynamic SQL queries.

Savvy users realized there's another potential avenue for attack with web controls. Although parameterized commands prevent SQL injection attacks, they don't prevent attckers from malicious values to the data that's posted back to the server. Left unchecked, this could allow attackers to submit control values that wouldn't otherwise be possible.

For example, imagine we have a list that shows evaluation results made by the current user. A craftly attacker could save a local copy of the page, modify the HTML to add more entries to the list, and then select one of these "fake" entries. If theis attack succeeded, the user will be able to see the evaluation results made by another user, which is an obvious problem.

Fortunately, ASP.NET 2.0 and hiegher versions defends against this attack using an event validation. Event validation checks the data that's posted back to the server and varifies that the values are legitimate. For example, if the POST data indicates the user chose a value that doesn't make sense (because it doesn't actually exist in the control), ASP.NET generayes an error and stps processing.

Programmers should employ stored procedures for the added ability to remove all permissions to the base tables in the database and thus remove the ability to create injection queries like were shown above. An example of this solution as follows:

```
private void cmdLogin_Click(object sender, System. Event Args e) {
    string strCnx = ConfigurationSettings.AppSettings ["cnx_e MCES"];
    using (SqlConnection cnx = new SqlConnection(strCnx))
{
    SqlParameter prm;
    cnx.Open();
    string strAccessLevel;
    SqlCommand cmd = new SqlCommand("procVerifyUser", cnx);
    cmd.CommandType= CommandType.StoredProcedure;
    prm = new SqlParameter("@username",SqlDbType.VarChar,   50);
    prm.Direction=ParameterDirection.Input;
    prm.Value = txtUser.Text;
    cmd.Parameters.Add(prm);
    prm = new SqlParameter("@password",SqlDbType.VarChar, 50);
    prm.Direction=ParameterDirection.Input;
    prm.Value = txtPassword.Text;
    cmd.Parameters.Add(prm);
    strAccessLevel = (string) cmd.ExecuteScalar();
    if (strAccessLevel.Length>0)
    {
        FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
    }
}
else
{
    lblMsg.Text = "Login attempt failed.";
}
```

Here:

- the LoginASP page uses a stored procedure, procVerifyUser, to validate users with two parameters – "@username" and "@password".
- The RedirectFromLoginPage method redirects to the return URL key specified in the query string. If the return key does not exist, RedirectFromLoginPage redirects to Default.aspx. ASP.NET automatically adds the return URL when the browser is redirected to the login page specified in the loginUrl attribute in the <forms> Element configuration directive. The method issues an authentication ticket and does a SetForms with the ticket, using an appropriately configured cookie name for the application as part of the redirect response.

While not directly related to SQL injection attacks, it reasonable to use another best practice security: the encryption of connection strings. Securing the connection string is especially important if it contains an embedded database account password. Since we will need the decrypted version of a connection string to connect to the database, you can't hash a connection string. We will need to encrypt it, instead. Here's what the encrypted connection string stored in Web.config and used by some of the Login.aspx looks like:

```
<add key= "cnx_eMCES"
    value= "AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAcWM Z8XhPz0O
    jHcS1539LAQAAAACAAAAAAADZgAAqAAAABA AAABdodw0YhWfcC6+
    UjUUOiMwAAAAAASAAACgAAAAEAAAALPzjTRnAPt7/W8
    v38ikHL5IAAAAzctRyEcHxWkzxeqbq/V9ogaSqS4UxvKC9zmrXUoJ9mwrNZ/XZ9LgbfcDXIIAX
    m2DLRCGRHMtrZrp9yledz0n9kgP3b3s+X8wFAAAANmLu0UfOJdTc4WjlQQgmZElY7Z8"
/>
```

Login.aspx calls the GetCnxString method from the SecureConnection class, proposed in [6] and shown here:

```
public class SecureConnection {
    static public string GetCnxString(string configKey) {
        string strCnx;
        try {
            // Grab encrypted connection string from web.config
            string strEncryptedCnx = ConfigurationSettings.AppSettings [configKey];
            // Decrypt the connection string
            DataProtector dp = new DataProtector (DataProtector.Store.USE_MACHINE_STORE);
            byte[] dataToDecrypt = Convert. FromBase64String (strEncryptedCnx);
            strCnx = Encoding.ASCII.GetString(dp. Decrypt (dataToDecrypt, null));
        }
        catch {
            strCnx="";
        }
        return strCnx;
    }
}
```

Now we can retrieve the cnx_eMCES AppSetting value and decrypt it with this code:

```
string strCnx = SecureConnection.GetCnxString ("cnx_eMCES");
```

The SecureConnection class in turn calls the DataProtect class library, which wraps calls to the Win32 Data Protection API (DPAPI). One of the nice features of the DPAPI is that it manages the encryption key for developer.

Another important technique of preventing hackers' attacks is an effective exception management procedure. Exception conditions can be caused by configuration errors, bugs in application's code, or malicious input. Without proper exception management, these conditions can reveal sensitive information about the location and nature of application's data source in addition to valuable connection details. Implementing of trap and log ADO.NET exceptions to data access code is an essential way to intercept and handle most possible errors/bugs of the application's code or malicious intents of the user. The best solution is to place data access code within a try/catch block and handle exceptions, when programmer writes ADO.NET data access code, like shown in fragment below:

```
try
{
    // Data access code
}
catch (SqlException sqlex) // more specific
{
}
catch (Exception ex) // less specific
{
}
```

In e-MCES (version 1.1.1) we use an Exception class, which represents errors that occur during application execution. This class is the base class for all exceptions. When an error occurs, either the system or the currently executing application reports it by throwing an exception containing information about the error. Once thrown, an exception is handled by the application or by the default exception handler. We omit the technical aspects of this class due a lack of space. The detail information of it is presented in [8]. A fragment of the C# programming code with our solution is as follows:

```
try
{
. . . . . .
}
catch (Exception err)
{
    err.Data["ASP"] = sASP;
    err.Data["User"] = Session["UserName"].ToString();
    err.Data["UserRole"] = Session["UserRole"].ToString();
    err.Data["Procedure"] = "Getting total evaluation for School/Dept.";
```

```
        err.Data["sp"] = storedProcedureName;
        Session["Exception"] = err;
        . . . . . .
        Response.Redirect("Error.aspx");
    }
    finally
    {
    . . . . . . . . . . . . . . . . . . . . }
```

In this code err Exception class plays role of a container, which accolumulates the information of the ASP page's name, which then generates an exception, UserName, UserRole, a name of the stored procedure, and briefly the definition of the the routine to be executed. The object err stores in the Session object for later populating of the Log table in the database, which accomulates all exceptions that occur during the application' life. This table we use for analysis of the e-MCES errors/bugs, and for registration of all user malicious attempts.

Actually, when an eror occurs during process of execution of the application, the Framework .NET environment generates a standard message on the client machine, which shows a detail information about error event. Below is a fragment of such message:

```
Server Error in '/EMCES2010' Application.
Compilation Error
Description: An error occurred during the compilation of a resource required to service this request.
Please review the following specific error details and modify your source code appropriately.
Compiler Error Message: CS1501: No overload for method 'GetChars' takes '1' arguments
Source Error:
Line 186:     {
Line 187:         FactorID = reader.GetInt16(0);
Line 188:         Weight = reader.GetChars(1);
Line 189:
Source File: c:\Documents and Settings\Administrator\My Documents\Visual Studio
2005\WebSites\EMCES2010\Total2526.aspx.cs            Line: 188

Show Detailed Compiler Output:
C:\Program Files\Microsoft Visual Studio 8\Common7\IDE>
"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\csc.exe" /t:library
/utf8output/R:"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET
Files\emces2010\874e21b9\686c33f1\ App_global.asax.8jtjkur4.dll"

Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.1433
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

c:\Documents and Settings\Administrator\My Documents\Visual Studio
2005\WebSites\EMCES2010\Total2526.aspx.cs(188,26): error CS1501: No overload for method
'GetChars' takes '1' arguments
 c:\WINDOWS\assembly\GAC_32\System.Data\2.0.0.0__b77a5c561934e089\System.Data.dll: (Location of symbol
related to previous error)
. . . . . .
```

We can see some specific and detailed exception information revealing sensitive data – Name of the database (EMCES2010), Location of the application on the server, Name of the ASP.NET page, which generates an exception (Total2526.aspx), etc. All this information is confidential, and shouldn't be shown to the end user. In e-MCES we collect the exception information using an err Exception class, store it in the database and redirect application to the specific Error.aspx page, which generates on the client machine a sandard message, as shown in Fig. 3. As we can see, this information does not present any sensitive data about application and server. It just informs the user, that the system has handled an error and informed the System Administrator.

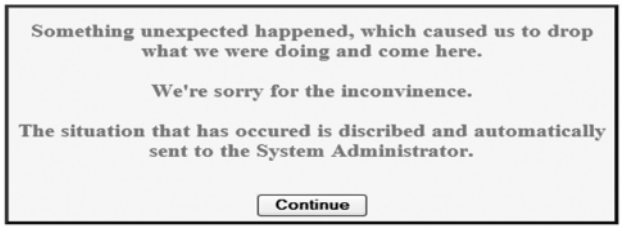


Fig. 3: Message, Generated by Error.aspx on Client Machine

The above mentioned sections showed the top threats to data access code and highlighted the common vulnerabilities. SQL injection is one of the main threats to be aware of. Unless developer uses the correct countermeasures discussed in these sections, an attacker could exploit application's data access code to run arbitrary commands in the database. Conventional security measures such as firewalls and Secured Socket Layer (SSL) provide no defense to SQL injection attacks. The developer should thoroughly validate input information and use parameterized stored procedures as a minimum defense.

4. ROLE-BASED SECURITY APPROACH

The e-MCES has been developed for all members of staff of the educational institution, including Academic/Non-Academic, Administrative, Technical, and Ancillary staff with different areas of responsibility. Some of them are managers and supervisors while others are employees – and all of them are members of the one institution's community. We have to evaluate each member's contribution to the success of the organization by everybody. Because the system serves all of us in the institution, with a single Internet interface and common data storage, it must have restricted access to the resources. Our solution is based on the concept of the role of the currently logged-on user [2, 9]. Using role-based security (or simply RBS), it is possible to programmatically determine the role/roles of the current user interacting with a given type or type member. In our system we associate one user with one or more roles.

The process of getting access to the system by user is comprised of the following two steps:

1. Authentication. The StartPage.aspx asks user to enter a University ID and checks whether the user is a member of the university community using the main data storage of the system—MS SQL Server. If the data entered by the user does not match the list of UTech IDs, it informs the user of this and does not allow entry into the system.
2. Authorization. At that level, we use a standard ASP.NET 2.0 Login Control, which provides an out-of-the-box Web User Interface (UI) for the purpose of validation of credential. Beyond offering the traditional UI, the Login control makes use of the specified membership provider to perform validation. Given all of this intrinsic functionality, we built a Login.aspx file with no code whatsoever, providing and nesting subelements into <asp:Login> tag, that maps on to the contain control.

To programmatically obtain the identity of the current user via the RBS model, we must obtain a principal object from the current thread of exception via Tread. CurrentPrincipal object. Technically speaking, a principal object is some type implementing the System.Principal.IPrincipal interface:

```
public interface IPrincipal {
    IIdentity Identity { get; }
    bool IsInRole(string role);
}
```

Evidentially, the read-only IPrincipal.Identity property returns an object implementing System.Security. Principal. IIdentity, which is defined as:

```
public interface IIdentity {
    string AuthenticationType { get; }
    bool IsAuthenticated { get; }
    string Name { get; }
}
```

Before obtaining a principal object via Thread.CurrentPrincipal, the calling assembly needs to inform the Common Language Runtime (CLR) of the principal policy. .NET provides four possible principal policies:

1. Forms: A RBS implementation for ASP.NET;
2. Generic: Enables us to define our own custom RBS system;
3. Passport: A RBS implementation for MS .NET Passport;
4. Windows: A RBS implementation for Win32 user account system.

Because the Form-based principal policy is used extensively when securing ASP.NET in web applications, we have decided to use it in our solution. The .NET security model enables us to restrict access to type allocation and type member invocation using Imperative RBS, which types directly into the code, making run-time demands and decisions where needed. With this approach we gain the capability to monitor access violation gracefully in code via try/catch constructs and/or simply deny a given course of action.

5. AUTHENTICATION AND AUTHORIZATION IN THE E-MCES

Process of the user's authentication involves several application's, network's and server's procedures. It works with a cookie on the client machine, which carries out the authentication ticket - specific and hidden information about current session. The authentication ticket is a unique ID, which is associated with current client machine and server. Fig. 4 shows a series of exchanges that occur between a web browser and a server when user attempts to access a page that is protected by forms-based authentication [10].

In e-MCES, we use several standard server controls, which are offered by .NET Framework and make programming security-related aspects of web applications easier than ever before: Login, PasswordRecovery, ChangePassword, etc. These controls rely entirely on the membership API (Application Programming Interface) and selected provider to execute standard operations such as validating credentials, displaying error messages, and redirecting to the originally requested page (in e-MCES it is a Welcome.aspx page), in case of successful login.

The membership API provides a set of classes with wide range of methods to let developer manage users and their roles: - adding a new user and editing any associated user information such as – e-mail and password, creating and managing association between users and roles.

The Membership class defaults to a provider that stores user's confidential access information to a SQL Express database in predefined format, through its property Provider.

This property returns a reference to the membership provider currently in use and selected from the configuration files web.config (located in root directory of the application) and machine.config (located in C:\WINDOWS\Microsoft.NET\Framework\vvv\CONFIG on the server, where vvv is the version of ASP.NET).

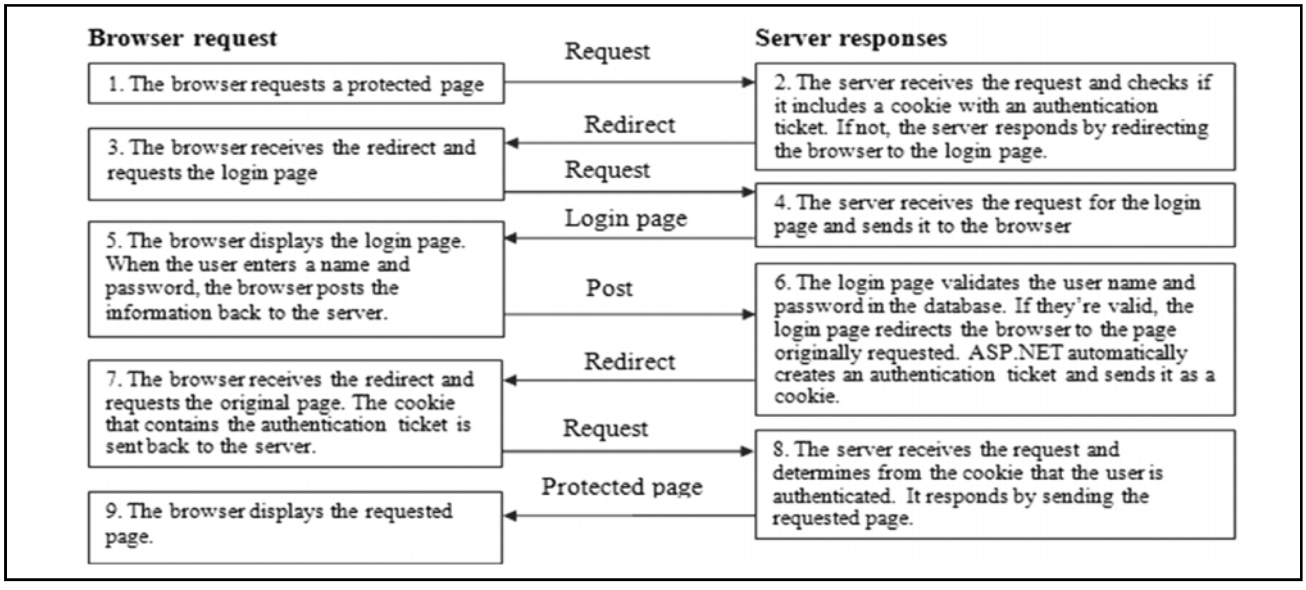


Fig. 4: HTTP Requests and Responses with Forms-based Authentication

An Example of a Child <provider> Element Under Which Provider Configures is as Follows:

```
<membership>
    <providers>
      <add name="AspNetSqlMembershipProvider"
        type="System.Web.Security.SqlMembershipProvider, System.Web, Version=2.0.0.0,
        Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
        connectionStringName="LocalSqlServer"
```

```
            enablePasswordRetrieval="false"
            enablePasswordReset="true"
            requiresQuestionAndAnswer="true"
            applicationName="/" requiresUniqueEmail="false"                    (1)
            applicationName="/" requiresUniqueEmail="false"
            passwordFormat="Hashed" maxInvalidPasswordAttempts="5"
            minRequiredPasswordLength="7"
            minRequiredNonalphanumericCharacters="1"
            passwordAttemptWindow="10"
            passwordStrengthRegularExpression=""/>
        </providers>
    </membership>
```

The content of the configuration file (1) describes a security policy of the application by values of its attributes. Some of them are:

- connectionStringName="LocalSqlServer" describes that application uses a standard .mdf security database, predifined by ASP.NET, which originally is located in App_Data folder. We have improved this solution, removed this database to the SQL Server from this application directory. It has given us a number of benefits, including its better maintenance.
- passwordFormat="Hashed" declares that confidential information about users access (user name, password, etc.) will be encrypted. Usually ASP.NET provides several hashing algorithms with 128-bits strength, for instance – Massage Digest method (MD5) and Secure Hash Algorithm (SHA) with different modifications [11, 12]. The e-MCES does modify security algorithm using HashAlgorithm object. The e-MCES does modify security algorithm using HashAlgorithm object. A portion of C# code is as follows:

```
static void Main(string[] args) {
    // Open a local configuration file on the C drive
    FileStream fs = new FileStream(@"C:\eMCES_ Config_ File.txt", FileMode.Open);
    // now generate a hash code for this file using MD5 hashing //algorithm
    HashAlgorithm alg = HashAlgorithm.Create ("MD5");
    byte[] fileHashValue = alg.ComputeHash(fs);
    ..........................................
}
```

- maxInvalidPasswordAttempts="5" – declares a number of unseccessful attempts are allowed
- minRequiredPasswordLength="7" – declares a minimum length of the password
- minRequiredNonalphanumericCharacters="1" – describes that at least one nonalphabetivcal charcter it must be presented in password.

The configuration information from web.config file overrides the same information in machine.config file.

6. NAVIGATION AND JUST-IN-TIME MENU GENERATOR

Navigation is a fundamental aspect of the e-MCES. In conjunction with security policy, it allows for the system to provide set of various Menus for different categories of users and their roles. The menu grants to a user his/her permissions. In an educational institution, like University of Technology, Jamaica, with number of employees more than 1,300, this is a big problem. We need to keep and maintain Menus for all categories of staff and this is enormous job. Traditional maintenance of these processes by updating an application (redesign and recompiling) is waste of time and energy. To create one very big and universal menu predicting new positions is also not possible. We have invented a novel solution for it in our system. Actually we do not keep the set of Menus in the system and in some supported configuration files at all. Instead, we keep it in the database as the fragments of any menu called as tokens. The system generates each virtual menu for each user Just in Time Compilation (JITC) of his request and keeps it during the current session of the user [13]:

- collecting essential tokens from the database, using our specific algorithm,

- at run-time generating a JavaScript menu program
- renders HTML code to the final DHTML code, and sends it to the client browser.

Accordingly [14], Just-In-Time compilation process includes any translation performed dynamically, after a program has started execution and is used to gain the benefits of static compilation and interpretation [7]. It generates HTML document, to be interpreted by the browser of client machine. These two factors provided stimulation for us to create our appropriate JITC technique.

In e-MCES, we consider menu at two levels – parent and its subordinate - child. We have created a Just In Time Menu Generator (JITMG) – a C# procedure, which uses tokens of menu as input information. Under tokens, we consider all necessary elements of future menu – texts of items, which will be seen by user and touchable (in terms of ability to be hyperlinks) by user, destination addresses (URLs) associated with these items, and link to the user's role. Fig. 5 shows a fragment of database, which holds tokens for JITMG. Table Roles receives a text value with User's role. Then two SQL queries retrieve set of tokens associated with this particular role from five linked tables:

- SELECT L1.[ID], L1.[Text] AS Text1 FROM mnuLevel1 AS L1 WHERE RoleName = '" + uRole + "'"
- SELECT L2.[Location], L2.mnuLevel1ID, L2.[Text] AS Text2 FROM mnuLevel2 AS L2 INNER JOIN mnuLevel1 AS L1 ON L2.mnuLevel1ID = L1.[ID] "WHERE L1.RoleName = '" + uRole + "'"

Here, an uRole is a text variable, which holds the name of user's role.

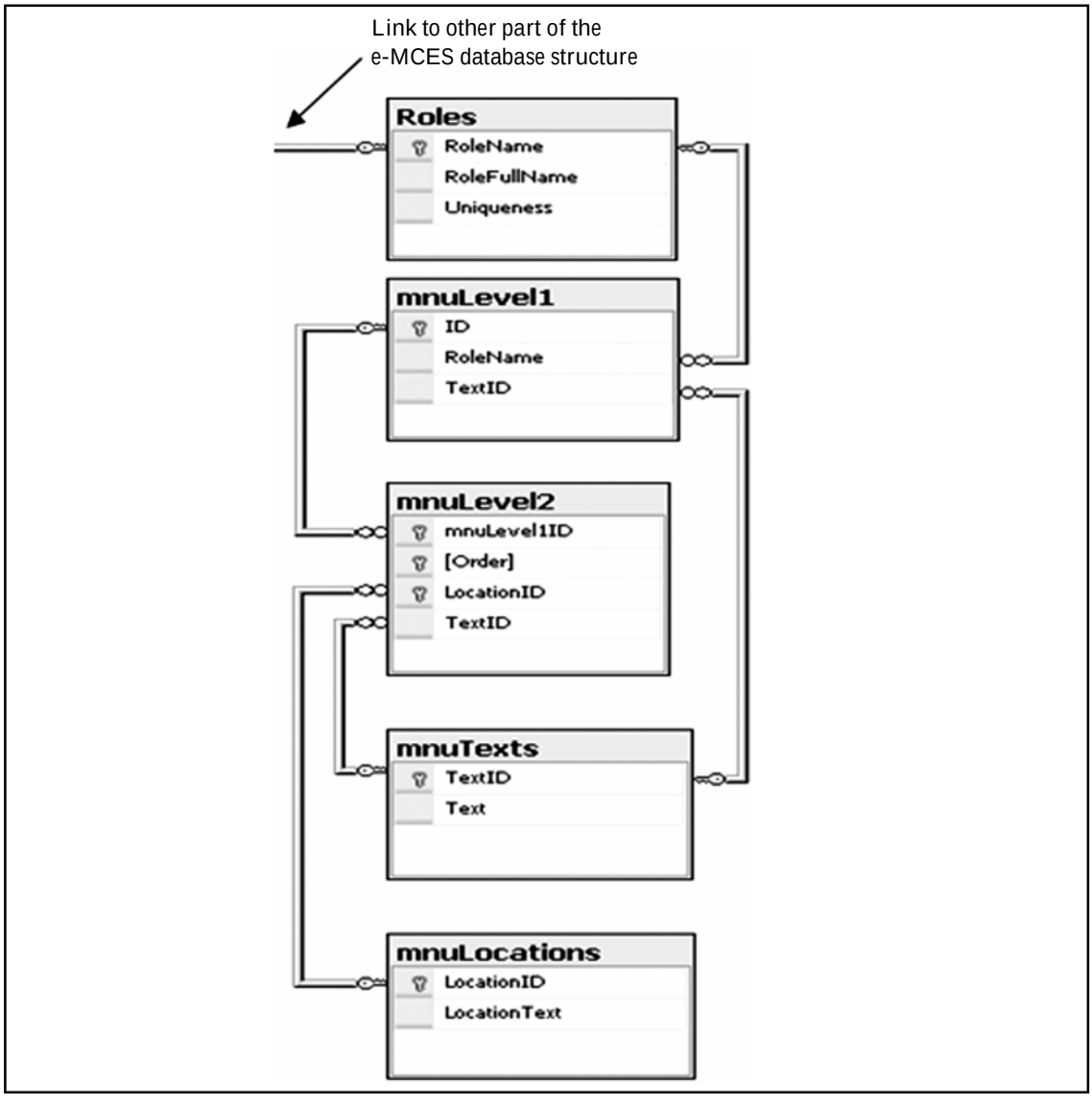


Fig. 5: Fragment of Database, Which Holds Tokens for JITMG

After that, a C# programming procedure of the Welcome.aspx page code-behind class uses the set of tokens, which were retrieved by SQL queries mentioned above, Navigation object from .NET Framework and generates a JavaScript program, which resides into the HTML response as its integral part. We believe that this solution of including C# and ASP.NET source code is an intellectual property of the University of Technology, Jamaica hence additional details are not presented here.

Appendix A shows a snapshot of the browser's screen with menu, generated for user Mr. Arnett Ford, who is in role of a "Head of School of Computing & Information Technology".

Appendix B shows fragments of JavaScript menu, generated automatically by JITMG. For the first session of the user System Administrator provides a temporary password, which is combination of two strings of user's University's ID, separated by dot symbol. This temporary password must be updated by the user as soon as possible due the security reason. The web.config file of the application (located in root directory of the application) holds a number of these attempts and can be modified by the System Administrator without recompiling the application. Default value of this number is 5. As mentioned above, the confidential information (User ID, password, etc.) kept in the database is an encrypted value and cannot be restored even by System Administrator. If the number of attempts exceeded 5, the user must be registered again with the system. This approach ensures a high level of security.

The JITMG solution has many advantages. Two of the main advantages are:

- By using this technology, it does not matter how big organisation is and how many different positions are there;
- To add a new position or new user, assign a new user to some position, or change position for some particular user is the responsibility of the person from HRD, who is assigned for these functions – he/she can do it through e-MCES interface which provides these functions, without recompiling the application. It is a usual routine for administrator of the system.

Appendix C shows a full content of menu (44 items, e-MCES version 1.1.1), generated by JITMG for Head of SCIT (Appendix B shows some choices, which have been made by the user).

7. CONCLUSION

The security solutions of the e-MCES, presented in this paper are the most valuable part of the process of building an Integrated Management System for Educational Institutions. Our solutions do not depend on size of the institution and on diversity of Academic and non-Academic roles. We have developed a new "Just-In-Time (JIT)" menu generator for granting permissions to logged-on users, with highest level of security. e-MCES does not depend on hierarchy, number and diversity of existing positions and roles in the institution. It grants new person to a new position with some role, provides the permission and creates a menu "on the fly", without storing it in the database. A practical experience of using e-MCES in the University of Technology, Jamaica proves that we have selected an effective way for improving the management processes at the University.

REFERENCES

[1] Howard M., and LeBlanc D., "Writing Secure Code, Practical Strategies and Proven Techniques for Building Secure Applications in a Networked World", 2nd ed., Microsoft Press, Redmond, Washington, 2003.

[2] Pougatchev V., Glasgow G., Ellis J., and Johnson N., "Online Performance Based Management and Evaluation System as an Instrument to Manage the Quality of Institutional Performance at the University of Technology, Jamaica", Paper Presented at the 11th IASTED International Conference on Computers and Advanced Technology for Education CATE-2008, (September 29 – October 1, 2008), Crete, Greece http://www.actapress.com/Abstract.aspx?paperId= 34196.

[3] Meier J.D., Mackman A., Dunner M., Vasireddy S., Escamilla R. and Murukan A., "Improving Web Application Security: Threats and Countermeasures. Patterns & Practices.", Microsoft Corporation, 2003 http://msdn.microsoft.com/en-us/practices.

[4] Henderson K., "The Guru's Guide to Transact-SQL", Addison-Wesley, Pearson Education, Indianapolis, IN, 2000, ISBN: 0201615762.

[5] Sceppa D., "Programming Microsoft ADO.NET 2.0. Core Reference", 2006, Microsoft, Microsoft Press, Redmond, Washington, USA, 2006.

[6] Litwin P., "Data Security: Stop SQL Injection Attacks Before They Stop You", MSDN2004 Magazine, Microsoft Press, Redmond, Washington.

[7] MacDonald M., Szpuszta M., Pro ASP.NET 2.0 in C#2005, Special Edition, APRESS®, 2006, ISBN: 1-59059-768-0

[8] Microsoft Software Development Network Library (MSDN), 2005.

[9] Pougatchev V., Glasgow G., Ellis J., and Johnson N., "Online Performance-based Management and Evaluation System as an Instrument to Manage the Quality of Institutional Performance at the University of Technology", Jamaica", Journal of Research in Innovative Teaching, National University, San Diego, California, USA, 2(1), March 2009, pp. 53-78, http://www.nu.edu/assets/resources/pageResources/7638_JournalofResearch09.pdf

[10] Pougatchev V., Kulkarni A., "Online Operationalization of Processes in e-Management Control and Evaluation System for a University", International Journal of Computer Science and Communication, ISSN 0973-4414, 4(2), (2010, June)

[11] "National Institute of Standards and Technology", FIPS 180: Secure Hash Standard, (1993 May), Available from: http://csrc.nist.gov.

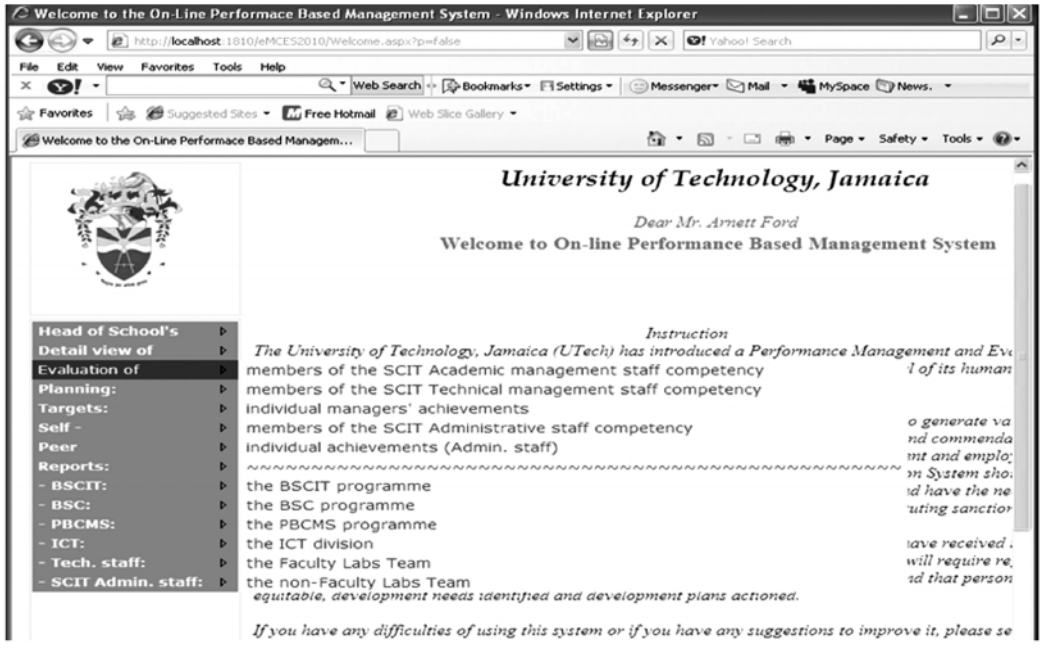
[12] FIPS 180-1., Secure hash standard. (1996). NIST, US Department of Commerce, Washington D.C., Springer-Verlag.

[13] Pougatchev V., (2008, September 29 – October 1). "Online Performance Based Management and Evaluation System at the University of Technology", Jamaica: Information Resources and Security Solutions. Paper presented at the 11th IASTED International Conference on Computers and Advanced Technology for Education (CATE-2008) Crete, Greece, http://www.actapress.com/ Abstract.aspx?paperId = 34196

[14] Aycock J., (2003, June), "A Brief History of Just-In-Time", ACM Computing Surveys, 35(2), pp. 97–113.

APPENDIX A

A snapshot of the browser's screen with menu, generated for the user Mr. Arnett Ford, which is in role a Head of School of Computing & Information Technology.



APPENDIX B

```
// java script program handling
<script type="text/javascript">
//<![CDATA[
var theForm = document.forms['fPBMS'];
if (!theForm)
  {
    theForm = document.fPBMS;
}
function__doPostBack(eventTarget, eventArgument)
{
   if(!theForm.onsubmit || (theForm.onsubmit() != false))
  {
    theForm.__EVENTTARGET.value = eventTarget;
    theForm.__EVENTARGUMENT.value = event Argument;
    theForm.submit();
  }
}
  //]]>
  /script>
   ......
  <scriptsrc="/EMCES2010WebResource.axd?
    d=GJ56476UwNAULC srfu P4Q2 &amp; t=634203
    587922187500"
    type="text/javascript"></script>
  <scriptsrc="/EMCES2010 WebResource. axd?d = 5v
    uYZmLBPR4bkOEVL2FIA2&amp;t=63420358
    7922187500"
    type="text/javascript"></script>
    ......
  //Items of the parent menu
    <td style="white-space:nowrap;width:100%;"><a
    class="NavigationMenu_1NavigationMenu_3
```

```
NavigationMenu_8"href=”javascript:__doPostBack
('Navigation Menu','~*|*Choice Error.aspx? P=HR1
')”>University's</a></td>
......
<tdstyle=“white-space:nowrap;width:100%;”><a
class=“NavigationMenu_1 NavigationMenu_3
NavigationMenu_8" href= “javascript:__
doPostBack('NavigationMenu',' ~*|*ChoiceError.
aspx? P=HR2')”>
HR evaluation</a></td>
......
//Items of child menus
<td style=“white-space:nowrap;width:100%;”><a
class=”NavigationMenu_1 NavigationMenu_3
NavigationMenu_8" href=” javascript:__doPostBack
('NavigationMenu','~*|* ChoiceError.aspx? P=HR4
_31')”> Reports:</a></td>
........
<td style=“white-space:nowrap;width:100%;”><a
class=”NavigationMenu_1 NavigationMenu_5"
href=”javascript:__doPostBack('NavigationMenu','
~*|*ChoiceError.aspx?P=HR2
President*|*Redirect President.aspx?D=DeptEval
&amp;F=HR&amp;G=T')”>totalscore</a></td>
.........
<td style=”white-space:nowrap;width:100%;”><a
class=”NavigationMenu_1 NavigationMenu_5"
href =” javascript:__ doPostBack ('NavigationMenu
','~*|* ChoiceError. aspx?P=HR3\\~*|*President
*|*RedirectPresident.aspx? D=DeptEval &amp;
F=HR&amp;G=D')”>the Human Resources score
evaluation</a></td>
........
<td style=“white-space:nowrap;width:100%;”><a
class=“NavigationMenu_1 NavigationMenu_5"
href = “javascript:__doPostBack ('NavigationMenu
','~*|* ChoiceError. aspx? P=HR4 _31 \\~*|* HR *|*
ExclusiveAccess*|*RedirectHRExclusive.aspx ?
D=reports&amp;F=totalScoreAdmStaffDCS
&amp;E=0')”>View a Scores list of the Day Care
Centre staff</a></td>
........
// java script program handling
<script type=”text/javascript”>
//<![CDATA[
var NavigationMenu_Data = new Object();
NavigationMenu_Data.disappearAfter = 500;
NavigationMenu_Data.horizontalOffset = 2;
NavigationMenu_Data.verticalOffset = 0;
NavigationMenu_Data.hoverClass = 'Navigation
Menu_17';
NavigationMenu_Data.hoverHyperLinkClass=
'NavigationMenu_16';
NavigationMenu_Data.staticHoverClass=
'NavigationMenu_15';
NavigationMenu_Data.staticHoverHyperLinkClass
='NavigationMenu_14';
//]]>
</script>
```

Appendix C

Content of menu, generated by JITMG for Head of School of Computing and Information Technology (e-MCES, version 1.1.1)

	Content of menu Level 1 (“parent”)	Content of menu Level 2 (“child”)
1.	Head of School's	total score
2.	Detail view of	the score of the School of Computing & Information Technology
3.		the Head of School of Computing & Information Technology's competency
4.		the Faculty of Engineering & Computing Score
5.		the Academic Division score
6.		the University corporate score
7.	Evaluation of	members of the SCIT Academic management staff competency
8.		members of the SCIT Technical staff competency
9.		individual manager's achievements
10.		members of the SCIT Administrative staff competency
11.		Individual achievements (Admin. staff)
12.		~~~~~~
13.		the BSCIT programme
14.		the BSC programme
15.		the PBCMS programme

16.			the Information and Communication Technology division
17.			the Faculty Labs Team
18.			the non-Faculty Labs Team
19.	Planning:		maintenance of SCIT operational plan (last year)
20.			maintenance of SCIT operational plan (next year)
21.			maintenance of Admin. office operational plan (last year)
22.			maintenance of Admin. office operational plan (next year)
23.	Targets:		assigning objectives/targets to the School (last year)
24.			assigning objectives/targets to the School (next year)
25.			assigning objectives/targets to the Admin. office (last year)
26.			assigning objectives/targets to the Admin. office (next year)
27.	Self		evaluation
28.	Peer		evaluation assigning for Academic staff
29.	Reports:		view of scores of the Academic staff managers
30.	-	BSCIT:	view of scores of the Academic staff (Professors/Assistant Professor/Principal lecturers
31.			view of scores of the Academic staff (Senior lecturers/Lecturers/Assistant lecturers
32.			view of scores of the Administrative staff
33.	-	BSC:	view of scores of the Academic staff (Professors/Assistant Professor/Principal lecturers
34.			view of scores of the Academic staff (Senior lecturers/Lecturers/Assistant lecturers
35.			view of scores of the Administrative staff
36.	-	PBCMS:	view of scores of the Academic staff (Professors/Assistant Professor/Principal lecturers
37	.		view of scores of the Academic staff (Senior lecturers/Lecturers/Assistant lecturers
38.			view of scores of the Administrative staff
39.	-	ICT:	view of scores of the Academic staff (Professors/Assistant Professor/Principal lecturers
40.			view of scores of the Academic staff (Senior lecturers/Lecturers/Assistant lecturers
41.			view of scores of the Administrative staff
42.	-	Tech.	view of scores of the Technical staff of the Faculty Labs Team
43.			view of scores of the Technical staff of the non-Faculty Labs Team
44.	-	SCIT	view of scores of the Administrative staff