Common Vulnerabilities
&
Recommended Solutions

Moosa Al Subhi IDS216

←Shift left

Shift left refers to moving security sooner in the development process.

Your application is first line of defence

شركة تنمية نفط عمان
Petroleum Development Oman

Most Common Vulnerabilities

MOST COMMON VULNERABILITIES WITH RECOMMENDED SOLUTIONS

- Open Redirect
- Cross Side Scripting XSS (DOM, persistent & reflected)
- SQL Injection
- Path Manipulation
- Mass Assignment: Insecure Binder Configuration
- Header manipulation: cookies
- Shoulder Surfing
- XML External Entity Injection
- Server-Side Request Forgery
- Insecure Data Transport

شركة تنمية نفط عمان
Petroleum Development Oman

OPEN REDIRECTION ISSUE / URL Redirection

A link which can be manipulated to redirect to another website

Built In IsLocalURL() Method For MVC and latest versions of Dot Net.	**Extension Methods For ASP .NET Webforms and technologies that does not support Built In Methods**

Returns a value that indicates whether the URL is local. A URL is considered local if it does not have a host / authority part and it has an absolute path. URLs using virtual paths ('~/') are also local
 	
Example : The following code instructs the user's browser to open a URL parsed from the dest request parameter when a user clicks the link.
 	
`String redirect = Request["dest"];`
`Response.Redirect(redirect);`
 	
Implementation:
`if(Url.IsLocalUrl(returnurl))`
`{`
`return Redirect(returnurl);`
`}`
`else`
`{`
`return redirectToAction("Index","Home");`
`}`
	`public class RequestExtensions`
	`    {`
	`        System.Web.HttpRequest request = HttpContext.Current.Request;`
	`        public bool IsLocalUrl(string url)`
	`        {`
	`            if (string.IsNullOrEmpty(url))`
	`            {  return false; }`
	`              Uri absoluteUri;`
	`            if (Uri.TryCreate(url, UriKind.Absolute, out absoluteUri))`
	`            { return String.Equals(request.Url.Host, absoluteUri.Host,`
	`StringComparison.OrdinalIgnoreCase);}`
	`            else`
	`            {`
	`                bool isLocal = !url.StartsWith("http:",`
	`StringComparison.OrdinalIgnoreCase) && !url.StartsWith("https:",`
	`StringComparison.OrdinalIgnoreCase) && Uri.IsWellFormedUriString(url,`
	`UriKind.Relative);`
	`                return isLocal;`
	`            }`
	`        }`
	`    }`

CROSS SIDE SCRIPTING XSS (DOM)

The attack payload is executed as a result of modifying the DOM "environment" in the victim's browser used by the original client-side script

Example 1:

The following jQuery code segment reads data object from an api and displays it to the user.

```
function loadPartialToDOM(DOM, obj, api) {
    $.ajax({
        url: window.location.origin + getRootFolderPath() + api ,
        data: obj[0],
        cache: false,
        type: "POST",
        dataType: "html",
        success: function (data, textStatus, XMLHttpRequest) {
            $('#' + DOM).html(data);
}
    });
}
```

Recommendation:

Using variables in this case causes the issue, since fortify expected unsafe value in variables.
Make div name fixed , and the object definition inline will do the work.

```
var obj = [];
obj.push({ LogKey: $('#hdnlogkey').val()});
```

CROSS SIDE SCRIPTING XSS (DOM)

The attack payload is executed as a result of modifying the DOM "environment" in the victim's browser used by the original client-side script

Example 2:	**Example 3:**

The method sends unvalidated data to a web browser, which can result in the browser executing malicious code
 	
`window.open(window.location.origin + getRootFolderPath() + '/ShutdownPlanning/NewShutDownPlan', "_self");`
 	
Recommendation:
Use JavaScript URL Builder as appending URL Directly in Windows.Open is an insecure way
 	
`Const params = new URL(window.location.origin + getRootFolderPath() + '/ShutdownPlanning/NewShutDownPlan');`
`window.open(myUrlWithParams);`
	`var initial_nav = window.location.hash;`
	`var scrollto = $(initial_nav).offset().top - scrolltoOffset;`
	`$('html, body').animate({`
	`scrollTop: scrollto }, 1500, 'easeInOutExpo'`
	`);`
	Recommendation:
	Implemented hash value encoding has resolved the issue
	
	`var encodedItem = encodeURIComponent(window.location.hash);`
	`if (encodedItem && encodedItem.length > 0) {`
	`var scrollto = $(encodedItem).offset().top - scrolltoOffset;`
	`$('html, body').animate({`
	`scrollTop: scrollto`
	`}, 1500, 'easeInOutExpo');`
	`}`

شركة تنمية نفط عمان
Petroleum Development Oman

CROSS SIDE SCRIPTING XSS (Persistent)

In the case of persisted (also known as stored) XSS it is typically a database or other back-end data store.

Example:	**Recommendation:**

```
DataTable dts = dt;
List<DataItem> studentList = new List<DataItem>();
studentList = (from DataRow dr in dt.Rows
          select new DataItem()  {
             Id = (Convert.ToInt32(dr["Id"])),
             Name = (dr["Name"].ToString())
          }).ToList();
```
	Encode data source using AntiXssEncoder.HtmlEncode Method (System.Web.Security.AntiXss)
	```
	DataTable dts = dt;
	List<DataItem> studentList = new List<DataItem>();
	  studentList = (from DataRow dr in dt.Rows
	          select new DataItem() {
	             Id = Html.Encode(Convert.ToInt32(dr["Id"])),
	             Name = Html.Encode(dr["Name"].ToString())
	          }).ToList();
	```

CROSS SIDE SCRIPTING XSS (Reflected)

In the case of reflected XSS, the untrusted source is typically a web request

Example:	**Recommendation:**

protected void ddlAsset_SelectedIndexChanged(object sender, EventArgs e)
{
try
{
lblAssetID.Text = ddlAsset.SelectedValue;
BindAreas(selectedIndex: -1, pageIndex: 0);
}
catch (Exception ex)
{
new LogException().MailException(ex);
messageBox.AddMessage(ConstantVariables.ErrorMessage, uscMsgBox.enmMessageType.Error);
}
}
	Encode data source using AntiXssEncoder.HtmlEncode Method (using System.Web.Security.AntiXss)
	
	lblAssetID.Text = Html.Encode(ddlAsset.SelectedValue);
	BindAreas(selectedIndex: -1, pageIndex: 0);
	
	OR
	Use the code behind to make a validation, this is what we did, this way will be efficient to get rid of the Vulnerability of the HTML input.
	
	If (user.length > 0 && rg.IsMatch(txtIWUserId.Text))
	{
	lblAssetID.Text = (ddlAsset.SelectedValue);
	BindAreas(selectedIndex: -1, pageIndex: 0);
	}

SQL Injection

A SQL injection attack involves the injection/insertion of untrusted data into a SQL query, causing a transition from the data context into the query context.

Example:

```
string searchTerm = "AW00011010";

// BAD CODE. DO NOT USE!!
string sql = $@"SELECT EmailAddress
            FROM dbo.DimCustomer
            WHERE CustomerAlternateKey = '{searchTerm}'";
var emails = new List<string>();
using (var connection = new SqlConnection(connectionString))
{
   connection.Open();
   using (var command = new SqlCommand(sql, connection))
   using (var reader  = command.ExecuteReader())
   {
      while (reader.Read())
      {
         string email = reader.GetString(0);
         emails.Add(email);
      }
   }
```

The SQL that is generated by the code will look as follows.

```
SELECT EmailAddress
FROM dbo.DimCustomer
WHERE CustomerAlternateKey = 'AW00011010’
```

The dangers

The bad code works, but why is it bad?

It is easy to demonstrate the dangers of the code by changing the searchTerm variable as follows.

```
string searchTerm = "' OR 1=1--";
```

The SQL that is generated will now look as follows.

```
SELECT EmailAddress
FROM dbo.DimCustomer
WHERE CustomerAlternateKey = '' OR 1=1--
```

This query is returning over 18,000 records, a potential data leakage!

……

By adding OR 1=1 we have specified a condition that will always evaluate to true, thus essentially removing any filtering and returning all results.

……

The -- text indicates a comment. If the SQL query string in the code contained additional conditions, the double-dash would ‘comment out’ the rest of the SQL code so that it would have no effect on the preceding injection SQL.

SQL Injection

A SQL injection attack involves the injection/insertion of untrusted data into a SQL query, causing a transition from the data context into the query context.

Example:	
Good code To fix this issue, we'll need to introduce SQL parameters.	**What fixes the issue** Instead of the search term being injected into the SQL string (i.e. WHERE CustomerAlternateKey = '{searchTerm}') the SQL string now references a parameter name instead (i.e. WHERE CustomerAlternateKey = @SearchTerm). The @SearchTerm parameter value is specified by adding a SQL parameter before executing the query, as follows. command.Parameters.Add(new SqlParameter("@SearchTerm", searchTerm)); As you can see, we didn't have to make very many changes to greatly improve the safety of the code. If you now try changing the value of the searchTerm variable to ' OR 1=1-- you will find that no results are returned. This is because the SQL parameter is encapsulating the search term data such that it is sent to the server separate from the SQL query text. This prevents an unintended transition from the data context into the query context.

```
string searchTerm = "AW00011010";

string sql = $@"SELECT EmailAddress
        FROM dbo.DimCustomer
        WHERE CustomerAlternateKey = @SearchTerm";

var emails = new List<string>();

using (var connection = new SqlConnection(connectionString))
{
  connection.Open();
  using (var command = new SqlCommand(sql, connection))
  {
    command.Parameters.Add(new SqlParameter("@SearchTerm", searchTerm));
    using (var reader = command.ExecuteReader())
    {
      while (reader.Read())
      {
        string email = reader.GetString(0);
        emails.Add(email);   }}}
```

PATH MANIPULATION

Path manipulation errors occur when the following two conditions are met when An attacker can specify a path used in an operation on the file system or by specifying the resource, the attacker gains a capability that would not otherwise be permitted.

Example:

Example 1: The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker may provide a file name like "..\\..\\Windows\\System32\\krnl386.exe", which will cause the application to delete an important Windows system file.

```
String rName = Request.Item("reportName");
File.delete("C:\\users\\reports\\" + rName);
```

Recommendations:

- Validate File path against Invalid Characters like below

```
invlaidCharList.AddRange(new List<Char>() { '<', '>', ':', '"', '/', '|', '?', '*' });
```

- Restrict user to list of accepted directories

```
        string dirname = new FileInfo(path).Directory.FullName;
        string filename = Path.GetFileName(path);
        var diInfo = new DirectoryInfo(dirname);
        foreach (var fi in di.GetFiles("*", SearchOption.AllDirectories))
        { fi.Delete(); }
```

- Use FileInfo to create or read file

```
        using (FileStream fs = files[0].OpenRead())
        {
        }
```

- if(!fullpath.StartsWith(Application Path))

```
{
   throw Exception();
}
```

- Use IsPathRooted

Mass Assignment: Insecure Binder Configuration

The Issue arises when the framework binder used for binding the HTTP request parameters to the model class has not been explicitly configured to allow, or disallow, certain attributes.

Example:

Example 1: With no additional configuration, the following ASP.NET MVC controller method will bind the HTTP request parameters to any attribute in the RegisterModel or Details classes:

```
public ActionResult Register(RegisterModel model)
{
   return View(model);
}
………………………………………………………………………
…………………………
public class RegisterModel
{
public string UserName { get; set; }
public string Password { get; set; }
public Details Details { get; set; }
}
………………………………………………………………………
…………………………
public class Details
{
public bool IsAdmin { get; set; }
}
```

Recommendations:

Depending on the framework used there will be different ways to control the model binding process:

Example 5: It is possible to control the ASP.NET MVC model binding process using opt-in approach, decorate the class with the [DataContract] attribute. If this attribute is present, members are ignored unless they have the [DataMember] attribute:

```
[DataContract]
public class Details
{
public Details()
{
IsAdmin = false;
}

[DataMember]
public int Id { get; set; }

public bool IsAdmin { get; set; }
...
}
```

Petroleum Development Oman

Mass Assignment: Insecure Binder Configuration

The Issue arises when the framework binder used for binding the HTTP request parameters to the model class has not been explicitly configured to allow, or disallow, certain attributes.

Recommendations:	**Recommendations:**

Example 2: It is also possible to control the ASP.NET MVC model binding process using opt-out approach, decorate the members with the [IgnoreDataMember] attribute. If this attribute is present then, those members are ignored during the model binding process:
`public class Details`
`{`
`public Details()`
`{`
`IsAdmin = false;`
`}`
`[IgnoreDataMember]`
`public bool IsAdmin { get; set; }`
`}`
	Example 3: In ASP.NET Web Form applications or Web API, you can use [BindRequired] and [BindNever] to specifically state whether a property or entire class should be bound:
	
	`public class Employee`
	`{`
	`public Employee()`
	`{`
	`IsAdmin = false;`
	`IsManager = false;`
	`}`
	`[BindRequired]`
	`public string Name { get; set; }`
	`[BindRequired]`
	`public string Email { get; set; }`
	
	`[BindNever]`
	`public bool IsManager { get; set; }`
	`[BindNever]`
	`public bool IsAdmin { get; set; }`
	`}`

شركة تنمية نفط عمان
Petroleum Development Oman

Header Manipulation: Cookies

Cookie Manipulation vulnerabilities occur when Data enters a web application through an untrusted source, most frequently an HTTP request and the data is included in an HTTP cookie sent to a web user without being validated.

Problem:	**Recommendations:**
Example: The following code segment reads the name of the author of a weblog entry, author, from an HTTP request and sets it in a cookie header of an HTTP response. string author = Author.Text; Cookie cookie = new Cookie("author", author); Assuming a string consisting of standard alphanumeric characters, such as "Jane Smith", is submitted in the request the HTTP response including this cookie might take the following form: HTTP/1.1 200 OK ... Set-Cookie: author=Jane Smith ……. If an attacker submits a malicious string, such as "Wiley Hacker\r\nHTTP/1.1 200 OK\r\n...", then the HTTP response would be split into two responses of the following form: From Hacker: HTTP/1.1 200 OK ... Set-Cookie: author=Wiley Hacker …. HTTP/1.1 200 OK ...	• Sanitize the input • AntiXssEncoder.HtmlEncode Method (System.Web.Security.AntiXss) | Microsoft Docs • Change mode to httponly myHttpOnlyCookie.HttpOnly = true;

Petroleum Development Oman
شركة تنمية نفط عمان

Shoulder Surfing

Some APIs that gather sensitive information can mishandle it by echoing it back to the user as he or she enters it at the input prompt.

Problem:	**Recommendations:**

Example 1: The following code demonstrates a model in ASP.NET that has a password property, but does not specify the DataType as a password, meaning that by default it will be shown when displayed:
 	
public class User
{
[Required]
public int ID { get; set;
[Required]
public string Username { get; set; }
 	
[Required]
public string Password { get; set; }
}
 	
Since the property Password in Example 1 did not specify the attribute [DataType(DataType.Password)], it will not be hidden by default when displayed in the UI.
	Always ensure that the appropriate flags and attributes are set correctly for any APIs that gather sensitive information, such that the sensitive information is not echoed back to the user as he or she enters it at the input prompt.
	
	Example 2: The following shows the corrected version of Example 1, such that the Password property will be hidden in the UI:
	
	public class User
	{
	[Required]
	public int ID { get; set; }
	
	[Required]
	public string Username { get; set; }
	
	[Required]
	[DataType(DataType.Password)]
	public string Password { get; set; }
	}

XML External Entity Injection

XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data.

Problem:	**Recommendations:**

XML parser configured does not prevent nor limit external entities resolution
`XmlDocument xDoc = new XmlDocument();`
`xDoc.LoadXml(samldata);`
	The XmlDocument object has an XmlResolver object within it that needs to be set to null in versions prior to 4.5.2. In versions 4.5.2 and up, this XmlResolver is set to null by default.
	
	`XmlDocument xDoc = new XmlDocument();`
	`xDoc.XmlResolver = null;`
	`xDoc.LoadXml(samldata);`

Server-Side Request Forgery

A Server-Side Request Forgery (SSRF) attack involves an attacker abusing server functionality to access or modify resources

Problem:

Example: In the following example, an attacker can control the URL to which the server is connecting.

```
string url = Request.Form["url"];
HttpClient client = new HttpClient();
HttpResponseMessage response = await client.GetAsync(url);
```

The attacker's ability to hijack the network connection depends on the specific part of the URI that can be controlled, and on the libraries used to establish the connection. For example, controlling the URI scheme lets the attacker use protocols different from http or https like:

- up://
- ldap://
- jar://
- gopher://
- mailto://
- ssh2://
- telnet://
- expect://

Recommendations:

Limit the user to specific resources by using whitelists

Example:

```
private readonly st
ring[] whiteList = { "www.example.com", "example.com" };


URI remoteUrl     = new Uri(url);
        string remoteHost = remoteUrl.Host;

        // Match the incoming URL against a whitelist
        if (!whiteList.Contains(remoteHost))
        {
           return BadRequest();
        }
```

Petroleum Development Oman

Insecure Data Transport

The application communicates with its database server over unencrypted channels and may pose a significant security risk to the company and users of that application. In this case, an attacker can modify the user entered data or even execute arbitrary SQL

Problem:	**Recommendations:**

The application communicates with its database server over unencrypted channels and may pose a significant security risk to the company and users of that application. In this case, an attacker can modify the user entered data or even execute arbitrary SQL commands against the database server.
 	
Example 1: The following configuration causes the application to communicate with its database server over unencrypted channels:
 	
`<connectionStrings>`
`<add name="Test" connectionString="Data Source=210.10.20.10,1433; Initial Catalog=myDataBase;User ID=myUsername;Password=myPassword;" providerName="System.Data.SqlClient" />`
`</connectionStrings>`
	Most database servers offer encrypted alternatives on different ports that use SSL/TLS to encrypt all the data being sent over the wire. Always use these alternatives when possible.
	
	Most database servers offer encrypted alternatives on different ports that use SSL/TLS to encrypt all the data being sent over the wire. Always use these alternatives when possible.
	`<connectionStrings>`
	`<add name="Test" connectionString="Data Source=210.10.20.10,1433; Initial Catalog=myDataBase;User ID=myUsername;Password=myPassword; Encrypt=yes;" providerName="System.Data.SqlClient" />`
	`</connectionStrings>`
	
	OR
	
	Encrypt the Connection string using command prompt

Petroleum Development Oman

شركة تنمية نفط عمان

Libraries

- **Jquery**
  jquery is a package that makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.
  Affected versions of this package are vulnerable to Cross-site Scripting (XSS), Prototype Pollution and Denial of Service (DoS) due to removing a logic that lowercased attribute names.
  Recommendation: Upgrade to version 3.5.1 or higher

- **jquery-form**
  jquery-form is a jQuery plugin that allows you to upgrade HTML forms to use AJAX.
  Affected versions of this package are vulnerable to Cross-site Scripting (XSS). The package does not sanitise AJAX responses before rendering them. Therefore, an attacker could potentially inject malicious HTML into the page.
  Recommendation: Use Customer Jquery Forms instead of using jQuery-form plugin

- **Adminlte**
  admin-lte is a Responsive open source admin dashboard and control panel.
  Affected versions of this package are vulnerable to Cross-site Scripting (XSS) via the IFrame and Sidebar Search classes, which do not sanitize input data.
  Recommendation: Upgrade admin-lte to version 3.1.0 or higher

- **dat.gui**
  dat.gui is an A lightweight graphical user interface for changing variables in JavaScript.
  Affected versions of this package are vulnerable to Regular Expression Denial of Service (ReDoS) via specifically crafted rgb and rgba values.
  Recommendation: Upgrade dat.gui to version 0.7.8 or higher

Libraries

- **Bootbox**

  bootbox is a JavaScript library which allows you to create programmatic dialog boxes using Bootstrap modals, without having to worry about creating, managing, or removing any of the required DOM elements or JavaScript event handlers.
  Affected versions of this package are vulnerable to Cross-site Scripting (XSS). The package does not sanitize user input in the provided dialog boxes, allowing attackers to inject HTML code and execute arbitrary JavaScript.

- **JQuery Validation Plugin**

  jquery-validation is a Client-side form validation made easy
  This Library is vulnerable to ReDoS
  Recommendation: Upgrade to version 1.19.5

- **qunit 1.23.1**

  qunit is a JavaScript unit testing framework
  Affected versions of this package are vulnerable to Cross-site Scripting (XSS).
  Recommendation: Upgrade 2.9.0 or higher

- **date.js**

  This package is no longer supported and has been deprecated. To avoid malicious use Change Library , Library is deprecated since 2017

- **CkEditor**

  ckeditor is a A highly configurable WYSIWYG HTML editor.
  Recommendation: Upgrade ckeditor4 to version 4.18.0 or higher.

Libraries

- **JSPDF**
  jspdf is a PDF Document creation from JavaScript
  Known Issues:
  Affected versions of this package are vulnerable to Regular Expression Denial of Service (ReDoS) and Cross Site Scripting (XSS)
  - ReDoS: version < 2.3.1
  - Cross Site Scripting (XSS): version < 2.0.0

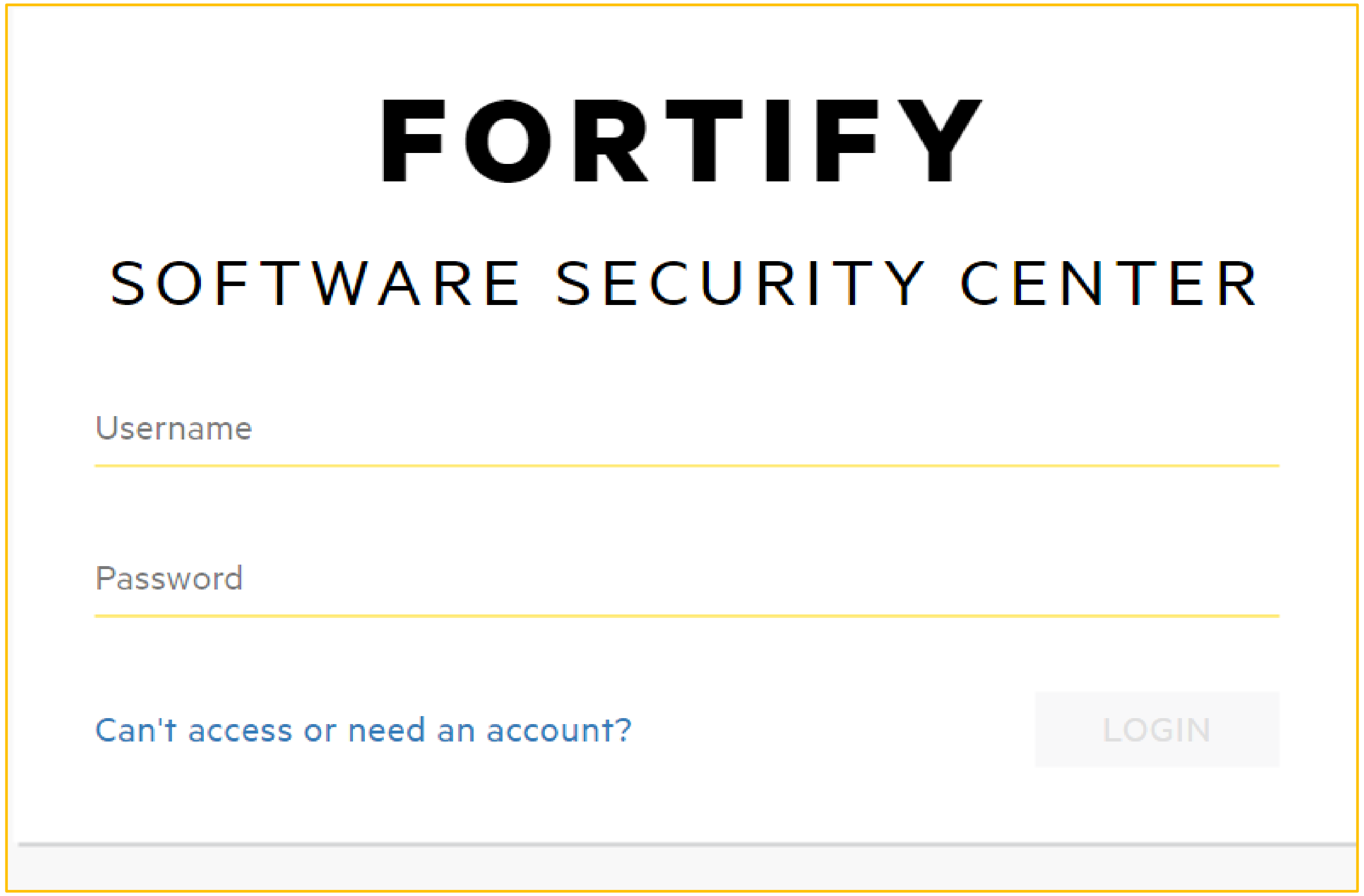
  Recommendation: Upgrade to version 2.3.1 (released on 9 Mar, 2021) or higher

- **bootstrap-daterangepicker**
  bootstrap-datepicker is vulnerable to a cross-site scripting (XSS) attack. The library does not properly handle the jQuery for the date container, allowing a malicious user to inject arbitrary Javascript.
  Recommendation: Upgrade to bootstrap-daterangepicker@2.1.18

Recommendations

- Upgrade libraries to latest version
- Storing encryption keys in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the encryption key.
- processName.StartInfo.Arguments = " -user " + input + " -role user";
- AntiXssEncoder.HtmlEncode Method (System.Web.Security.AntiXss) | Microsoft Docs
- Restrict user
- Sanitize input
- Encode before sending data to browser
- Dynamic Code Evaluation

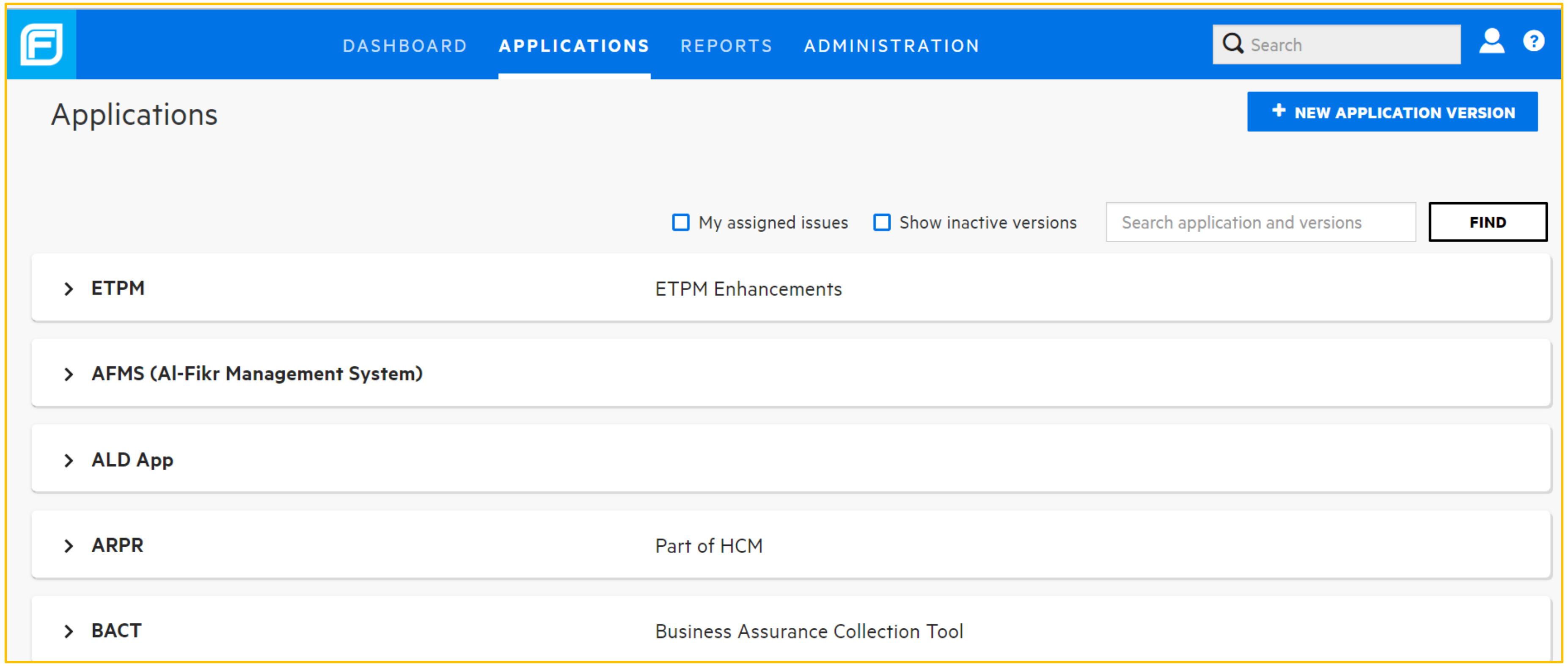
Logging in to Fortify Software Security Center



1- In a web browser, type the URL (**https://mus-as-531/ssc/**) and enter your windows account credentials:

Petroleum Development Oman

Accessing Scanned Applications in SSC Portal



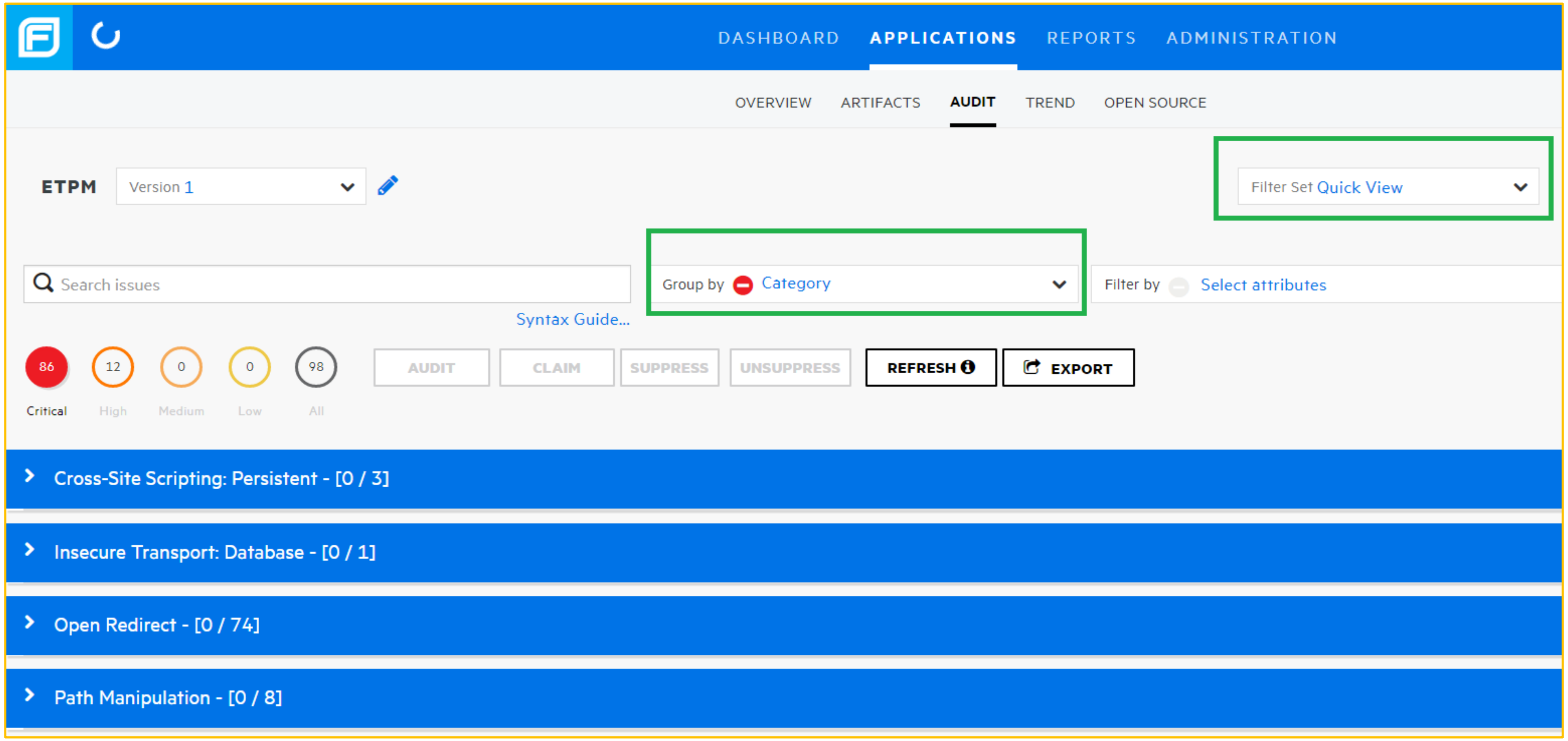
Applications

ETPM	ETPM Enhancements
AFMS (Al-Fikr Management System)	
ALD App	
ARPR	Part of HCM
BACT	Business Assurance Collection Tool

2- After log in, all the current user's applications will be available

شركة تنمية نفط عمان
Petroleum Development Oman

Vulnerabilities in Application



Cross-Site Scripting: Persistent - [0 / 3]

Insecure Transport: Database - [0 / 1]

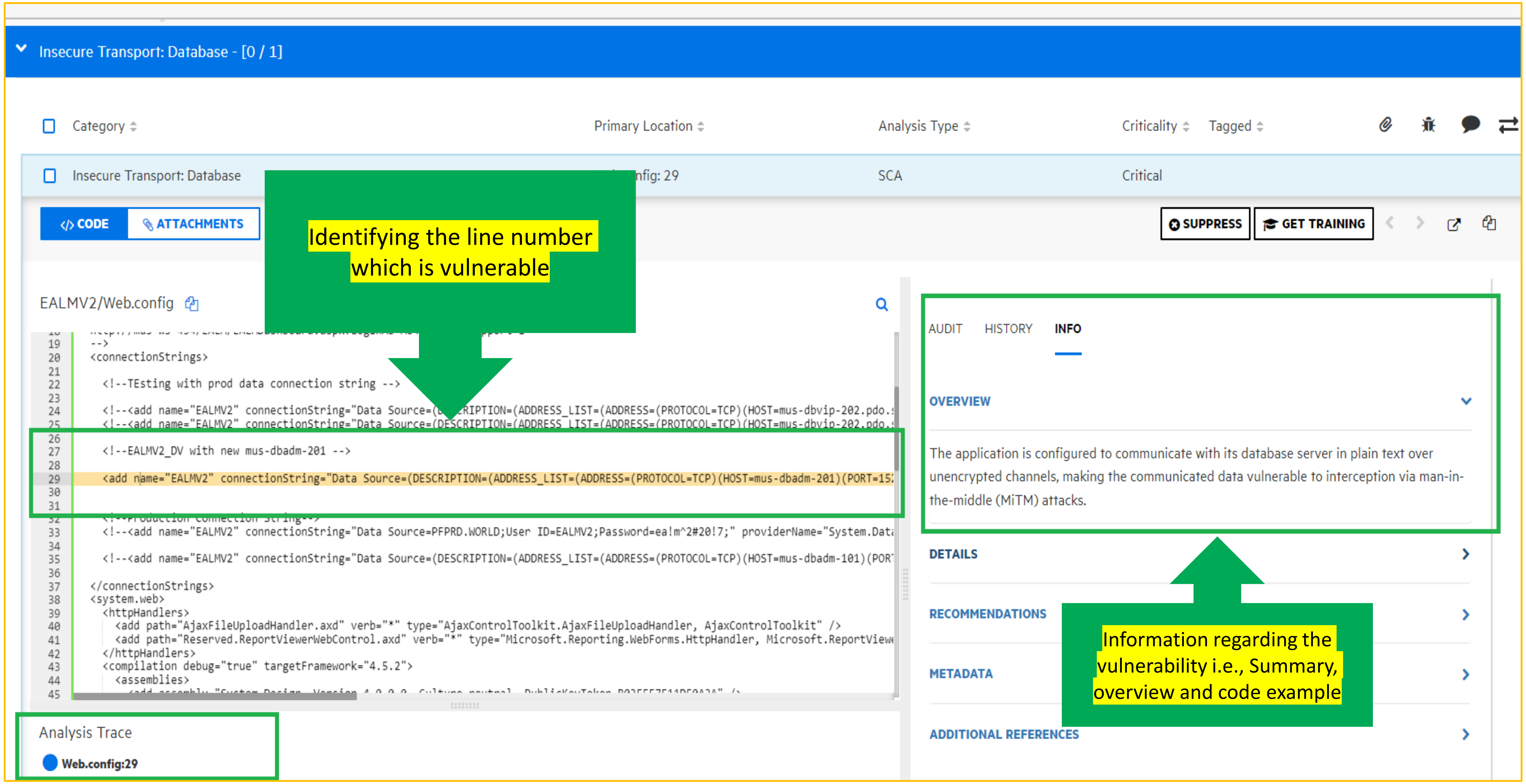
Open Redirect - [0 / 74]

Path Manipulation - [0 / 8]

3- Expand the latest version of application

شركة تنمية نفط عمان
Petroleum Development Oman

Vulnerability Details



Insecure Transport: Database - [0 / 1]

Category	Primary Location	Analysis Type	Criticality	Tagged
Insecure Transport: Database	…nfig: 29	SCA	Critical	

Identifying the line number which is vulnerable

EALMV2/Web.config

```
19     -->
20     <connectionStrings>
21
22       <!--TEsting with prod data connection string -->
23
24       <!--<add name="EALMV2" connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=mus-dbvip-202.pdo.
25       <!--<add name="EALMV2" connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=mus-dbvip-202.pdo.
26
27       <!--EALMV2_DV with new mus-dbadm-201 -->
28
29       <add name="EALMV2" connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=mus-dbadm-201)(PORT=15
30
31
32       <!--Production Connection String-->
33       <!--<add name="EALMV2" connectionString="Data Source=PFPRD.WORLD;User ID=EALMV2;Password=ea!m^2#20!7;" providerName="System.Data
34
35       <!--<add name="EALMV2" connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=mus-dbadm-101)(PORT
36
37     </connectionStrings>
38     <system.web>
39       <httpHandlers>
40         <add path="AjaxFileUploadHandler.axd" verb="*" type="AjaxControlToolkit.AjaxFileUploadHandler, AjaxControlToolkit" />
41         <add path="Reserved.ReportViewerWebControl.axd" verb="*" type="Microsoft.Reporting.WebForms.HttpHandler, Microsoft.ReportViewe
42       </httpHandlers>
43       <compilation debug="true" targetFramework="4.5.2">
44         <assemblies>
```

Analysis Trace

● Web.config:29

OVERVIEW

The application is configured to communicate with its database server in plain text over unencrypted channels, making the communicated data vulnerable to interception via man-in-the-middle (MiTM) attacks.

DETAILS

RECOMMENDATIONS

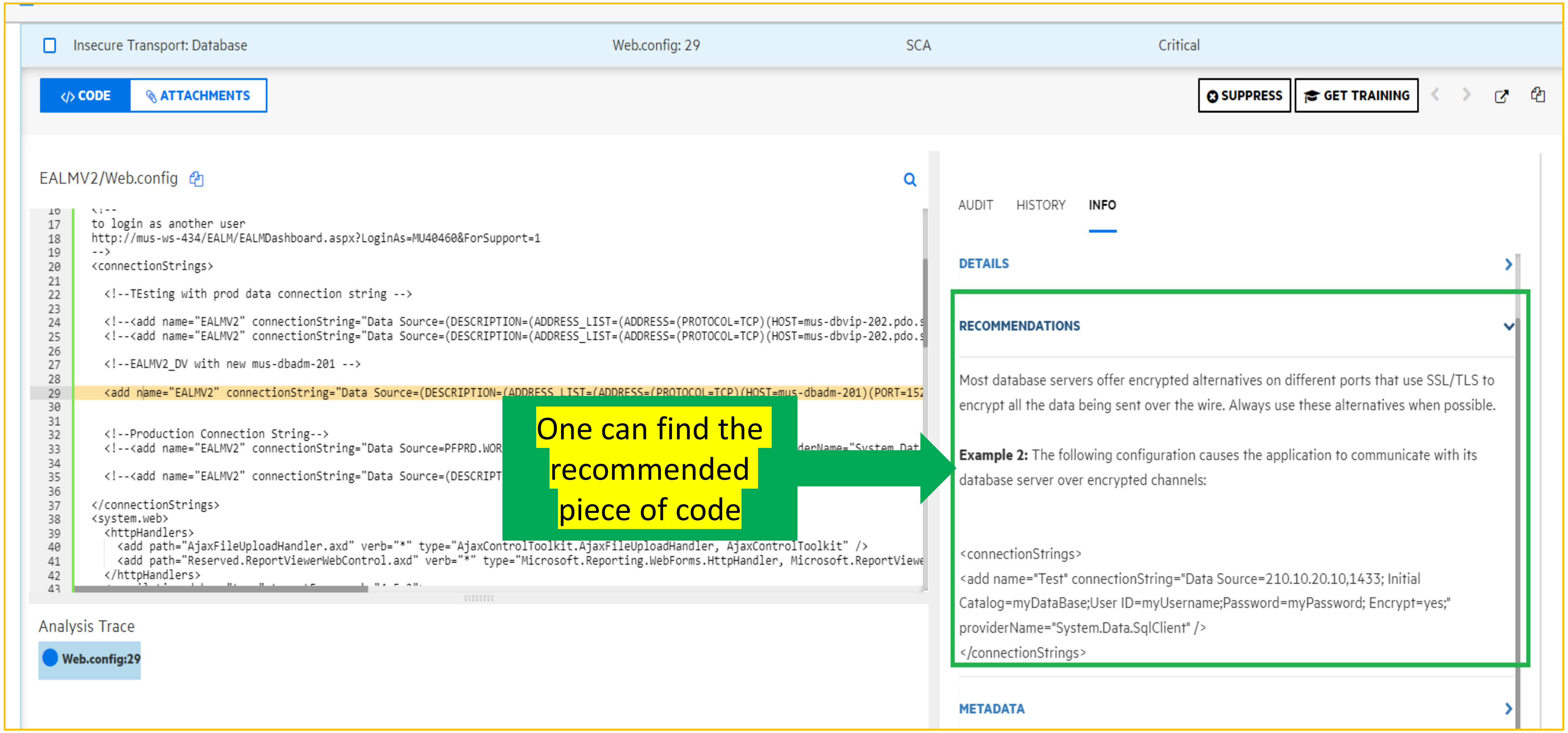
METADATA

ADDITIONAL REFERENCES

Information regarding the vulnerability i.e., Summary, overview and code example

Petroleum Development Oman

Vulnerability Recommendations



Insecure Transport: Database | Web.config: 29 | SCA | Critical

EALMV2/Web.config

```
16   <!--
17   to login as another user
18   http://mus-ws-434/EALM/EALMDashboard.aspx?LoginAs=MU40460&ForSupport=1
19   -->
20   <connectionStrings>
21
22     <!--TEsting with prod data connection string -->
23
24     <!--<add name="EALMV2" connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=mus-dbvip-202.pdo.s
25     <!--<add name="EALMV2" connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=mus-dbvip-202.pdo.s
26
27     <!--EALMV2_DV with new mus-dbadm-201 -->
28
29     <add name="EALMV2" connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=mus-dbadm-201)(PORT=152
30
31
32     <!--Production Connection String-->
33     <!--<add name="EALMV2" connectionString="Data Source=PFPRD.WOR                              derName="System.Dat
34
35     <!--<add name="EALMV2" connectionString="Data Source=(DESCRIPT
36
37   </connectionStrings>
38   <system.web>
39     <httpHandlers>
40       <add path="AjaxFileUploadHandler.axd" verb="*" type="AjaxControlToolkit.AjaxFileUploadHandler, AjaxControlToolkit" />
41       <add path="Reserved.ReportViewerWebControl.axd" verb="*" type="Microsoft.Reporting.WebForms.HttpHandler, Microsoft.ReportViewe
42     </httpHandlers>
43
```

Analysis Trace

Web.config:29

RECOMMENDATIONS

Most database servers offer encrypted alternatives on different ports that use SSL/TLS to encrypt all the data being sent over the wire. Always use these alternatives when possible.

Example 2: The following configuration causes the application to communicate with its database server over encrypted channels:

```
<connectionStrings>
<add name="Test" connectionString="Data Source=210.10.20.10,1433; Initial Catalog=myDataBase;User ID=myUsername;Password=myPassword; Encrypt=yes;" providerName="System.Data.SqlClient" />
</connectionStrings>
```

One can find the recommended piece of code

Petroleum Development Oman

Thank you

شركة تنمية نفط عمان
Petroleum Development Oman