{Client Name}

{Month} {Year}

{Client Name .Net} Code Review

Client Details

Company Name: {Client Name}

Contact Person: {Person Name}

Address: {Address Name}

Email: {Email Address}

Telephone: {Telephone Number}

Document History

Version	Date	Author	Remark
1.0	{Date}	{Author}	Document Creation

Sample Report

Table of Contents

Sample Report

1. About Payatu

Payatu is a research-powered security testing service organization specialized in IoT and embedded products, web, mobile, cloud & infrastructure security assessments, and in-depth technical security training. Our state-of-the-art research, methodologies, and tools ensure the safety of our client's assets.
At Payatu, we believe in following one's passion, and with that thought, we have created a world-class team of researchers and executors who are bending the rules to provide the best security services. We are a passionate bunch of folks working on the latest and leading-edge security technology.
We are proud to be part of a vibrant security community and don't miss any opportunity to give back. Some of the contributions in the following fields reflect our dedication and passion.

- nullcon - nullcon security conference is an annual security event held in Goa, India. After years of effort put in the event, it has become a world-renowned platform to showcase the latest and undisclosed research.
- hardwear.io - Hardware security conference is an annual hardware security event held in The Hague, Netherlands. It is being organized to answer emerging threats and attacks on hardware. We aim to make it the largest platform, where hardware security innovation happens.
- Dedicated fuzzing infrastructure - We are proud to be one of the few security research-based companies to own an in-house infrastructure and hardware for distributed fuzzing of software such as browsers, client, and server applications.
- null - It all started with null - The open security community. It's a registered non-profit society and one of the most active security community. null is driven totally by passionate volunteers.
- Open source - Our team regularly authors open-source tools to aid in security learning and research.
- Talks and Training: Our team delivers talks/highly technical training in various international security and hacking conference, i.e., DEFCON Las Vegas, BlackHat Las Vegas, HITB Amsterdam, Consecwest Vancouver, nullcon Goa, HackinParis Paris, Brucon Belgium, zer0con Seoul, PoC Seoul to name few.

We are catering to a diverse portfolio of clients across the world, who are leaders in banking, finance, technology, healthcare, manufacturing, media houses, information security, and education, including government agencies. Having various empanelment and accreditations, along with a strong word of mouth, has helped us win new customers. Our thorough professionalism and quality of work have brought repeat business from our existing clients.
We thank you for considering our security services and requesting a proposal. We look forward to extending the expertise of our passionate, world-class professionals to achieve your security objectives.

Sample Report

2. Document Map

This report consists of the following sections:

- **Introduction**

  In this section, we will discuss about the general information about project, including the scope duration and findings.

- **Architecture/Configuration Overview**

  This section will provide brief overview of application design/architecture/configuration and will also highlight the shortcomings of application architecture/design/configuration from security point of view.

- **Executive Summary**

  This section is meant to provide a general understanding of the security status of the application, for management team. High level view of the information gathered during the assessment, usually using graphs, tables or comparative numbers will be provided.

  This section will also provide, a glance of the vulnerability found during **Whitebox** security assessment of **<Application Name>** and will hold overview of the found issues/vulnerabilities sorted by their severity.

- **Finding Details**

  For each issue, this section includes all relevant details, including a detailed security advisory, and all variants, examinations, vulnerability details, associated Risk (impact), screenshots, code snippet, proof of vulnerability, and proof of concept and fix recommendations. This section is used both to educate on the nature and impact of the different issues, and to guide their remediation.

Sample Report

3. Introduction

This report holds the results of a **Whitebox Security Assessments** performed on the **<Application Name>** by the **Payatu** security team.

3.1 Scope and Objective

The scope of the security assessment was limited to auditing **<Application Name>** source code for **backdoors and exposed vulnerabilities** on the target system.

The objective was auditing the source code for unknown vulnerabilities and any backdoors that can subvert the whole system or send critical information to a third party without the authorization of the Client. There are **17 modules** comprising of **VB.NET, ASPX, PL/SQL** and **JavaScript** code. Out of which a few are not to be audited as specified by the Client. The modules include:

1. Source code audit of <Application Name> .NET engine.
2. Source code audit of the backend database PL/SQL

As per the SOW/Contract the assessment was requested to be performed on the below mentioned list of modules.

Source Code Audit

S. No.	Assets	VB.NET Files	ASPX Files	JS Files	Total Files	Approx. Total Size (KB)	Comments
1	<Module 1>					564	
2	<Module 2>					41	
3	<Module 3>					85	
4	<Module 4>					22	
5	<Module 5>					6500	
6	<Module 6>					400	
7	<Module 7>					125	

Sample Report

Database Audit

S. No.	Object Type	Count of Objects	LOC	Size (KB)	REMARK
1	Index				Out of scope
2	Lob				Out of scope
3	Package				
4	Package Body				
5	Procedure				0 PROCEDURES INITALLY, THEY GET ADDED AS AND WHEN RULES ARE CERATED. CONSIDER OUT OF SCOPE
6	Queue				
7	Sequence				
8	Table				Total 12487 columns
9	Trigger				
10	Type				
11	View				

3.2 Time Duration

The security assessment performed for a period of **60 days** from **<Start Date>** to **<End Date>** by two consultants as requested by the customer.

3.3 Findings

The Payatu security team performs manual, automated code review and real-time security assessments on web applications. These assessments aim is to uncover any security issues in the assessed web application, explain the impact and risks associated with the found issues, and provide guidance in the prioritization and remediation steps.

The security assessment revealed **1 critical**, **21 high**, **4 medium**, **6 low** and **4 informational** severity issues in this application. The consolidate summary of assessment has been presented in Executive Summary section. Additional information is contained within the Finding Details Information section of this report.

Sample Report

4. <Application Name> Architecture/Configuration Overview

4.1 Application Architecture Overview

<Application Name> consists of multiple VB.Net Project/Modules. Each module generates a separate executable file which works independently to serve any application specific purpose. Each module contains one configuration file. The name of the configuration file is either **web.config** or **app.config** depending on whether the application is web based or thick client. These configuration files contain different parameters and their values which are used by the application to perform certain tasks.

4.2 Application Configuration Security Consideration

During our security assessment we noticed that some of the configured parameters are not safe from the application security perspective. For example, the XML processed by the XML HTTP Interface is accepted without any form of authentication because the authentication setting is disabled in **web.config** file.

We also noticed that <Module Name> stops responding while performing a scan with automated web scanning tools.

Most of the configuration issues, related to each module, have been described in the Finding Details section of the detailed report.

4.3 Database Architecture Overview

<Application Name> database has a simple architecture. It contains **176 tables excluding all <Table Name> tables** (<Table Name> tables are generated dynamically based on total number of <Rule Name> created). All the tables are independent of each other. There is no relationship between the tables (No Foreign Key constraint).

There are **254 *views* excluding <View Name1> and <View Name2> *views***. The views whose name starts with AX$XXX and LX_XXX (XXX is a place holder) and have been created for ease of writing database queries. Instead of writing complex queries to extract data from multiple tables, the data can be extracted using a simple query with the help of these views. The views whose name starts with LX_XXX (XXX is a place holder) are the exact copy of table name PX_XXX (XXX is a place holder) and the only records which are marked as "**deleted=0**" (Not deleted) in the tables have been stored in these views.

Total **166 *triggers*** have been created in database schema and all the *triggers* are "**BEFORE Triggers**". Each *trigger* is associated with one table and gets executed before the *insert* operation on the table. There are **182 *sequences*** in total and each sequence is associated with one *trigger*. All the *triggers* have the same functionality i.e., just take "**<Value 1>**" from associated *sequence* and assign it to one specific column of the table named XXX_ID (XXX is a placeholder). The *Sequence* increases the ID value by 1 each item there is an *insert* operation in the table.

Sample Report

There are **44 *packages* excluding *packages* created for <Package Name1>** (<Package Name1> related packages are created dynamically). Each *package* contains multiple *procedures*. Most of the *procedures* accept below 3 *parameters*

(<Parameter 1>, InXML CLOB, OutXML OUT CLOB)

All these *procedures* get called from the VB.Net code and take *InXML* as Input, perform specific operation, or execute SQL query on specific table and return *OutXML* as output. Few of the important *packages* which contain the core of <Application Name> logic is mentioned below.

1. <Package Name 1>
2. <Package Name 2>
3. <Package Name 3>
4. <Package Name 4>
5. <Package Name 5>
6. <Package Name 6>
7. <Package Name 7>
8. <Package Name 8>
9. <Package Name 9>

Below given are two *packages* which contain the *procedures* to perform actions like parsing of XML file, execute SQL query, and log all the executed queries in database before/after execution etc.

1. <Package Name 10>
2. <Package Name 11>

There are **11 *queues*** in total. *Procedures* inside the *packages* i.e. <Package Name 1>, <Package Name 2> use these queues to en-queue SMS, Email, IVR messages, event data, and workflow to be consumed by various applications.

For example, "***Class <Class Name>***" inside "***<Directory Name>\<File Name>.vb***" calls "***<Package Name.Procedure Name>***" procedure to send mail to the intended recipient. It gets the recipient details from the same *procedure* via ***OutXML*** file.

4.4 Database Security Consideration

Apart from the password hash, no encryption has been used to store the data within the database. Information such as <Column Name 1>, <Column Name 2> is also stored in plain text format.

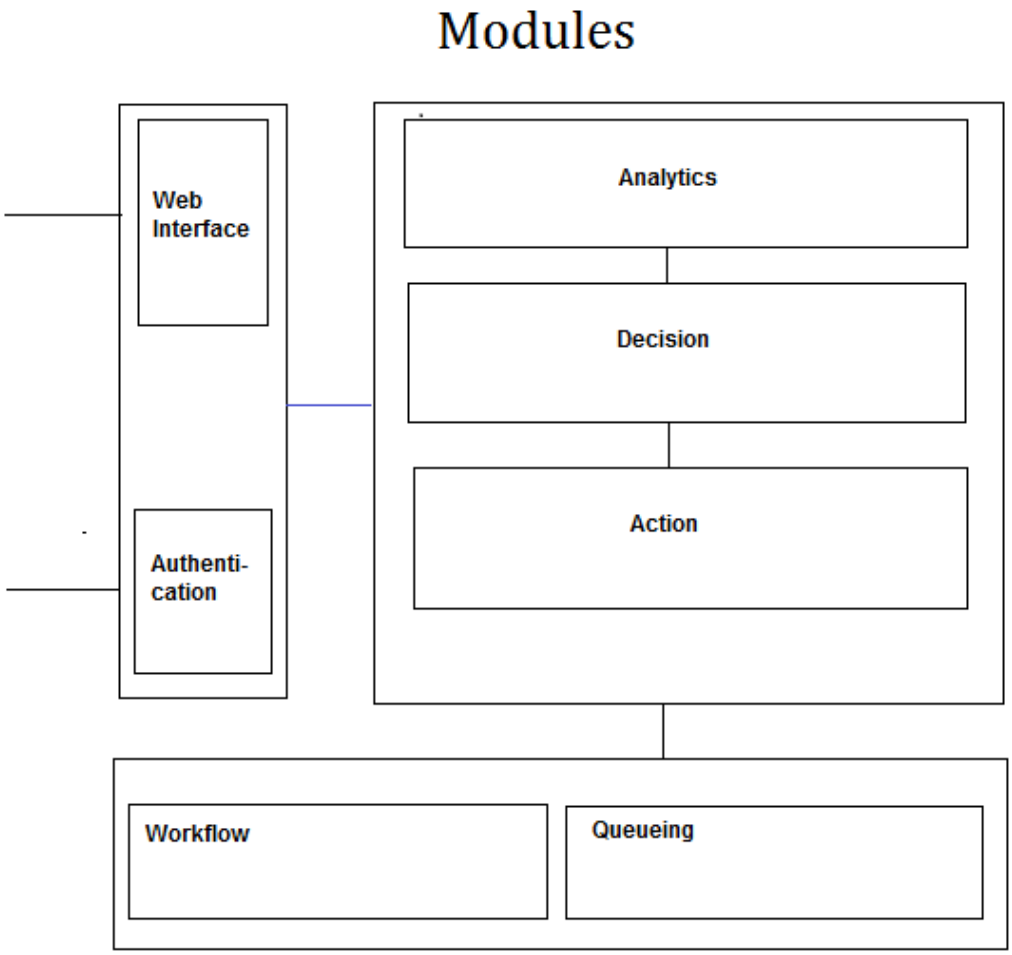
Most of the PL/SQL queries written inside the *procedure*s have been written using "***bind variable technique***" which prevents SQL injection attacks. However, there are places where string concatenation has been used to create a SQL *query*, which might lead to SQL injection attack if the attacker is able to control the parameter passed to these SQL *queries*.

Sample Report

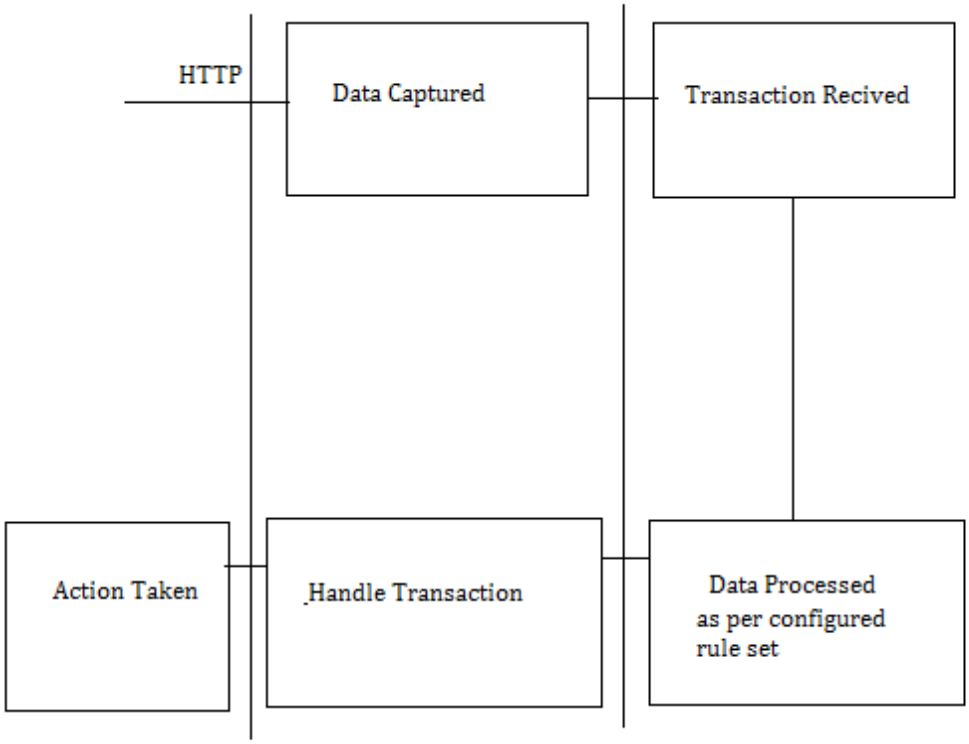
We found one specific place where it is possible to execute any procedure or run arbitrary DBMS code, if the attacker can control the XML file provided to **<Procedure Name>** *Procedure* defined inside **<Package Name>** *Package*.

4.5 Application Architecture and Transaction Flow Diagram

Below are the diagrams, which show the Application Modules interaction, Technical Architecture Details and Transaction flow.



Transaction Flow



Sample

5. Executive Summary

5.1 Review Summary

Whitebox Security Assessment of **<Application Name>** has been performed, considering below security issues:

- Is any intentional backdoor present in the code?
- Appropriate Access Control Mechanism and User Right Management are in place or not.
- Proper Input Validation and Sanitization has been incorporated in the application or not.
- Proper Session Management has been implemented or not.
- Are there any configuration mistakes that might leak important information about the application?

Though we didn't find any intentional backdoor in code, overall security postures of the **<Application Name>** is not very strong, most of the security controls/measures have not been properly thought of/implemented during the design and coding of the application. The application has been developed/ coded based on consideration that it will be used in internal environment by trusted users. Only protection against SQL injection vulnerability has been implemented in most of the places to prevent database hacking.

The Application has most of the common vulnerabilities like SQL injection, Unauthenticated XML Processing, Cross Site Scripting (XSS), XML External Entity (XXE) attack, Cross Site Request Forgery (CSRF), Broken/Improper Access Control, No Transport Layer Protection, Sensitive Information Leakage Through Log Files, Security Misconfiguration, ASP.NET Misconfiguration, Improper Error Handling, Un-validated Redirect and Weak Hashing Algorithm used for password storage, Non-Exploitable XSLT Injection etc.

While testing <Application Name> application we discover that application is not able to handle automated request generated by scanning tools. It goes in the state of DOS (Denial of Service) and stops responding after few seconds.

Note: Please note that some vulnerabilities mentioned, exist at multiple pages/parameters and for brevity we have consolidated the similar vulnerabilities into a single one in the detailed findings, for example Multiple Stored Cross Site Scripting (XSS), XML External Entity (XXE). The developers, however, will be required to fix each one of them separately.

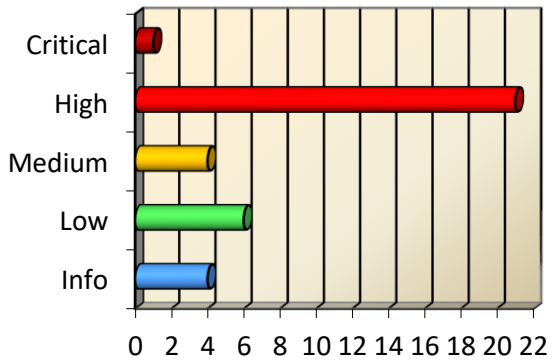
We have developed "**Proof of Concept**" (PoC) codes for few of the vulnerabilities that we found during the security assessment to show the impact of the vulnerability.

5.2 Summary of Findings

Findings by Risk Rating

Sample Report

The discovered finding table and chart illustrated below, provides a snapshot view of the number and severity of issues discovered during this security assessment.



Critical - This issue can impact the application severely and should be addressed immediately. Attackers can gain root or super user access or severely impact system operation.

High - Serious impact to the application security, this issue can cause a problem like unprivileged access and should be addressed as soon as possible.

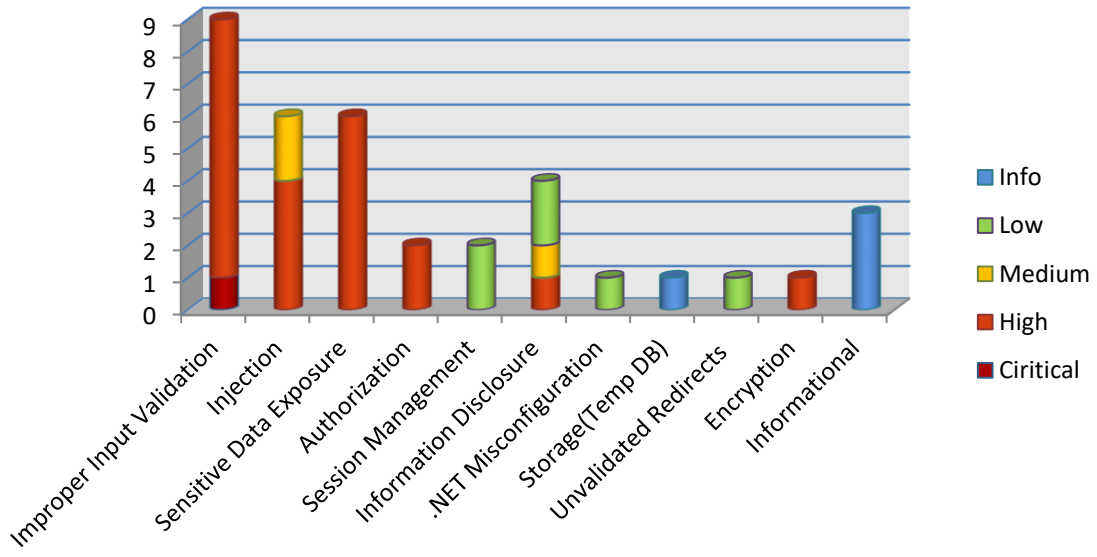
Medium - Notable impact to the application security, this issue may pose a significant threat over a longer period of time.

Low - Potential impact to the application security, this issue is more likely an information disclosure and may be an acceptable threat.

Info - Does not directly make the code less secure, but bad coding practice.

Findings by Category

Below given chart shows the vulnerability matrix based on category of vulnerabilities.



Sample Report

5.3 Table of Findings

<Module Name 1>

No exploitable vulnerability found in XXXXXXX 2FA module.

<Module Name 2>

Ref	Finding	Severity
1	**XML External Entity (XXE) Processing:** Processing of an external entity containing tainted data may lead to disclosure of confidential information and other system impacts.	HIGH
2	**No Transport Layer Protection:** Application is using plain text protocol (HTTP) to send and receive data from external source/destination.	HIGH

<Module Name 3>

Ref	Finding	Severity
1	**Cross-Site Scripting (Stored XSS):** There are multiple **Stored XSS** vulnerabilities in the application. Most of the parameters are vulnerable to XSS.	CRITICAL
2	**Improper Access Control:** There is no server side validation of the user action. **<Function Name 1>** and **<Function Name 2>** function does not validate the user access right before performing action.	HIGH
3	**XML External Entity (XXE) Processing:** Processing of an external entity containing tainted data may lead to disclosure of confidential information and other system impacts.	HIGH
4	**SQL Injection: <Function Name 4>, <Function Name5> and <Function Name 6>** functions are vulnerable to SQL injection attack because it does not sanitize the input data.	HIGH
5	**Weak Hashing Algorithm: <Function Name 7>** uses MD5 hashing algorithm to hash the passwords which has known weakness. Application is using salt with no value/constant value which makes security of password weaker.	HIGH
6	**No Transport Layer Protection: <Module Name 1>** is configured to run on HTTP. It is possible to sniff the login sequence.	HIGH
7	**Open Redirect: <Function Name 1>** and **<Function Name2>** functions does not have a page redirection policy and redirects the	LOW

Sample Report

Ref	Finding	Severity
	user to the page as specified by **<Parameter Name>** parameter without validation and this can be controlled by attacker.	
8	**AUTOCOMPLETE Attribute:** AUTOCOMPLETE attribute is not disabled in HTML Form/Input element containing username, password. Passwords/access code may be stored in browsers and retrieved.	**LOW**
9	**Missing Secure Attribute in Encrypted Session (SSL) Cookie:** Application does not set "**Secure**" flags for the session cookies.	**LOW**
10	**Database Procedure Name Disclosure:** The **<Function Name>** web service function throws exception to the response, thus, revealing database procedure name.	**LOW**
11	**Insufficient CSRF Protection: <Module Name>** is vulnerable to CSRF attack (conditional) as it depends on VIEWSTATE functionality of .NET framework, which is not meant for CSRF protection.	**LOW**
12	**Web Server Fingerprint:** Web server version and type knowledge allows attacker to determine known vulnerabilities and the appropriate exploits to use during exploitation.	**INFO**

<Application Name> Database Finding

Ref	Finding	Severity
1	**SQL Injection: <Procedure Name>** is suspected to be vulnerable to SQL injection as it creates dynamic SQL queries by string concatenation.	**HIGH**
2	**Encryption Not Used:** Application does not use encryption before storing critical data like Bank Account Number, Primary Account Number, etc. in the database.	**HIGH**
3	**Suspected Second Order SQL Injection attack:** We suspect few of the PL/SQL statement are vulnerable to second order SQL injection, as it takes output of one query and use the same output to create another query by concatenating the output value with new query.	**MEDIUM**

Sample Report

6. Finding Details

This section provides details about the specific weaknesses that were found during the review. These details are designed to provide the developers with proof that the stated weaknesses exist as well as to provide examples that the developers can use to find and fix similar areas of the code. Development team should use the information in these findings as an opportunity to improve the entire code base.

6.1 <Module Name 1>

No exploitable vulnerability found in this module.

6.2 <Module Name 2>

6.2.1 XML External Entity (XXE) Processing (Severity: High)

Advisory

Improper Input Validation: XML Injection

XML External Entities attacks benefit from an XML feature to build documents dynamically at the time of processing. An XML entity allows including data dynamically from a given resource. External entities allow an XML document to include data from an external URI. Unless configured to do otherwise, external entities force the XML parser to access the resource specified by the URI, e.g., a file on the local machine or on a remote system. This behaviour exposes the application to XML External Entity (XXE) attacks, which can be used to perform denial of service of the local system, gain unauthorized access to files on the local machine, scan remote machines, and perform denial of service of remote systems.

Vulnerability Details

S. No.	Source File	Line No.	Description	Risk Rating
1	<File Name>.vb	53	The application takes and XML from database *procedure* name <procedure name> and does not verify the XML before passing it to <Function Name> function. If attacker can control the database queue and push the malicious XML then it is possible to exploit this vulnerability.	**HIGH**

Sample Report

Associated Risk (Impact)

- If the attacker can include a crafted DTD and the default entity resolver is enabled, the attacker may be able to access arbitrary files on the system.
- The attacker will be able to launch DOS attack on the application consuming XML data from untrusted source.

Code Snippet

Serial No. 1 – Line No. 53

```
Dim ResponseXML As String
Dim ResponseErrorStack As String = ""
Dim XMLDoc As New XmlDocument
XMLDoc.LoadXml(EventXML)
Log.Info("XML Loaded.")
```

Proof of Concept

Proof of concept code for XXE (XML External Entity) attack has been provided into Appendix section.

Proof of vulnerability

N/A

Fix Recommendations

- XML parser should not follow URIs to External Entities, or make it only follow known good URIs (whitelisted URIs).
- Do not retrieve external DTDs or interpret inline DTDs by default.
- Do not support external entity by default.
- Follow the article for more information on defending against XXE attack on VB.NET version 4.0. Link: http://msdn.microsoft.com/en-us/magazine/ee335713.aspx

6.3 <Module Name 3>

6.3.1 Cross-Site Scripting (Stored XSS) (Severity: Critical)

Advisory

Improper Input Validation: Cross Site Scripting

Cross-Site Scripting attacks are a type of injection problem in which malicious scripts are injected into the web page on the browser via a vulnerable web application. Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious

Sample Report

code, generally in the form of a client-side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread; this vulnerability occurs when a web application uses inputs from a user directly, without validating or encoding it, in the response.

Vulnerability Details

S. No.	Source File	Line No.	Description	Risk Rating
1.	<File Name 1>.vb	25, 80	**<Function Name>** function is vulnerable to XSS because it does not sanitize the output returned by **<Procedure Name>** *procedure* which take the data (column name **<Column Name>**) from **<Table Name>** table **(<View Name>** view).	**CRITICAL**
2.	<File Name 2>.vb	30	**<Function Name>** function is vulnerable to XSS attack because it does not sanitize the parameter **<Parameter Name>** (column) coming from table **<Table Name>**	**CRITICAL**

Associated Risk (Impact)

- The application does not sanitize the data before storing on database and displaying on the web page. An attacker can use this vulnerability to launch different kinds of attack on the uses of the web portal.
- Attacker can steal user session cookie and perform user impersonation.
- Attacker can change the password of victim user without his knowledge.

Code Snippet

Serial No. 1 – Line No. 25

```
CXXXXXXXXus = dsXXXXXXX.ExXXXXdure(PXXXXXe, "XXXXXXX_SXXXEM.XXXXXXXXX")
GXXXXXl.PXXXXXe(CXXXXXXXus)
UsXXXXXXXe.Text = objXML.ExtractValue(CXXXXXXXXXus, "//NAME") & " " &
objXML.EXXXXXe(CXXXXXus, "//CREATED_DATE")
SXXXXXo.Text = objXML.EXXXXXXue(CuxxXXXXs, "//MXXXO")
```

Serial No. 2 – Line No. 30

```
If Objxml.EXXXXXXue(sXXXML, "//EXXXXXXE") = "1" Then
    lXXXXXXXXa.Text = objParser.TXXXXXXXXn(Objxml.EXXXXXe(strInXML, "//RXXXXE"))
Else
    lXXXXXXa.Text = "There is no XXXXXX Definition."
```

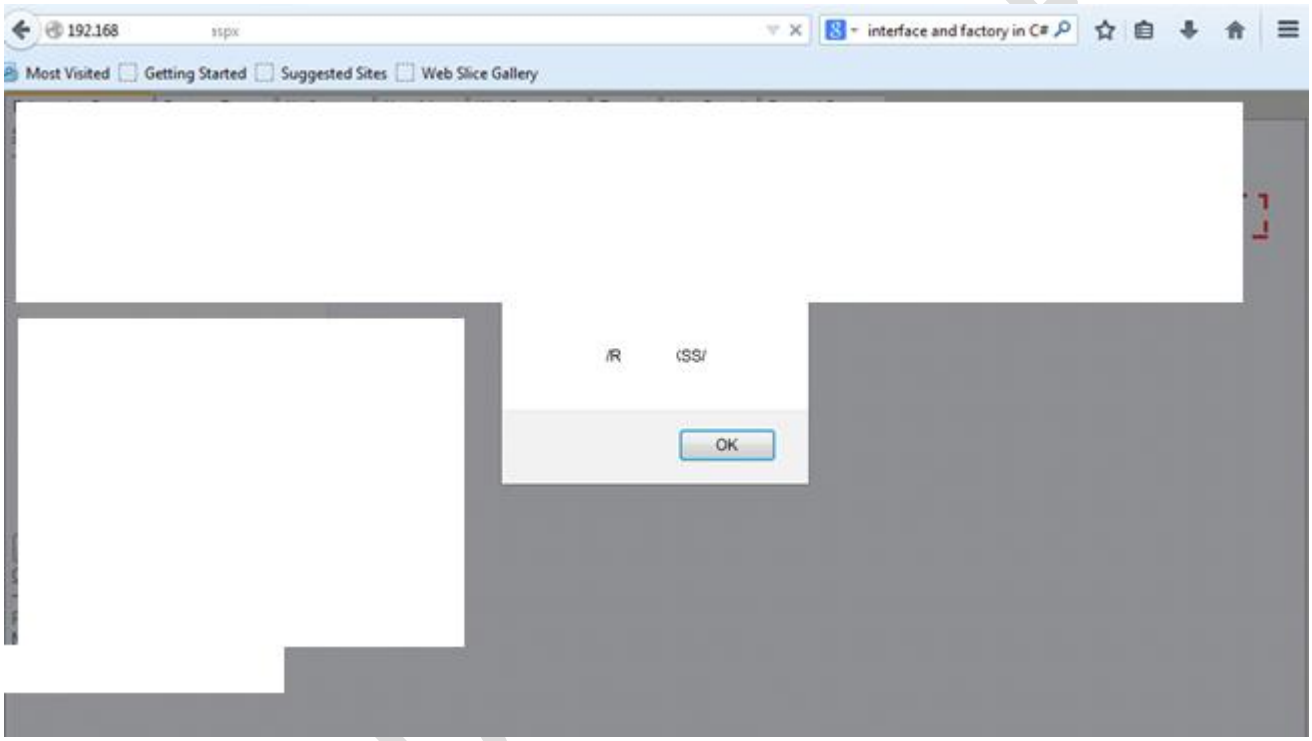
Sample Report

End If

Proof of Concept

Payload: %uff1cscript%uff1ealert(String(/XXXXXXX XSS/))%uff1c/script%uff1e

Proof of vulnerability



Fix Recommendations

- Do not trust client-side input even if there is client side validation.
- Sanitize potentially danger characters in the server side. Very often filtering the **<**, **>**, **", characters** prevent injected script to be executed in most cases. However, sometimes other danger meta-characters such as **'** , **(** ,) , **/** , **&** , ; etc. are also needed.
- In addition, encode HTML meta-characters in the response. For example, encode **<** as **<**
- Always generate new Anti-CSRF tokens for each request/page and validate against the same request/page while user POST the data.

Sample

6.3.2 Improper Access Control (Severity: High)

Advisory

Authorization: Broken Access Control

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication and govern what 'authorized' users are allowed to do.

Many of these flawed access control schemes are not difficult to discover and exploit. Frequently, all that is required is to craft a request for functions or content that should not be granted. Once a flaw is discovered, the consequences of a flawed access control scheme can be devastating. In addition to viewing unauthorized content, an attacker might be able to change or delete content, perform unauthorized functions, or even take over site administration.

Vulnerability Details

S. No.	Source File	Line No.	Description	Risk Rating
1.	<File Name 1>.aspx.vb	201-255	There is no server-side validation of the user action. **<Function Name 1>** function does not validate the user access right before performing the changes password action. Currently client-side restriction has been implemented, which can be bypassed very easily.	**HIGH**
2.	<File Name 2>.aspx.vb	150-175	There is no server-side validation of the user action. **<Function Name 2>** function does not validate the user access right before performing the revoke action. Currently client-side restriction has been implemented which can be bypassed very easily.	**HIGH**

Associated Risk (Impact)

- It is possible to revoke and change password of other ADMIN users even though this functionality is disabled on the web portal.

Code Snippet

Serial No. 1 – Line No. 150-175

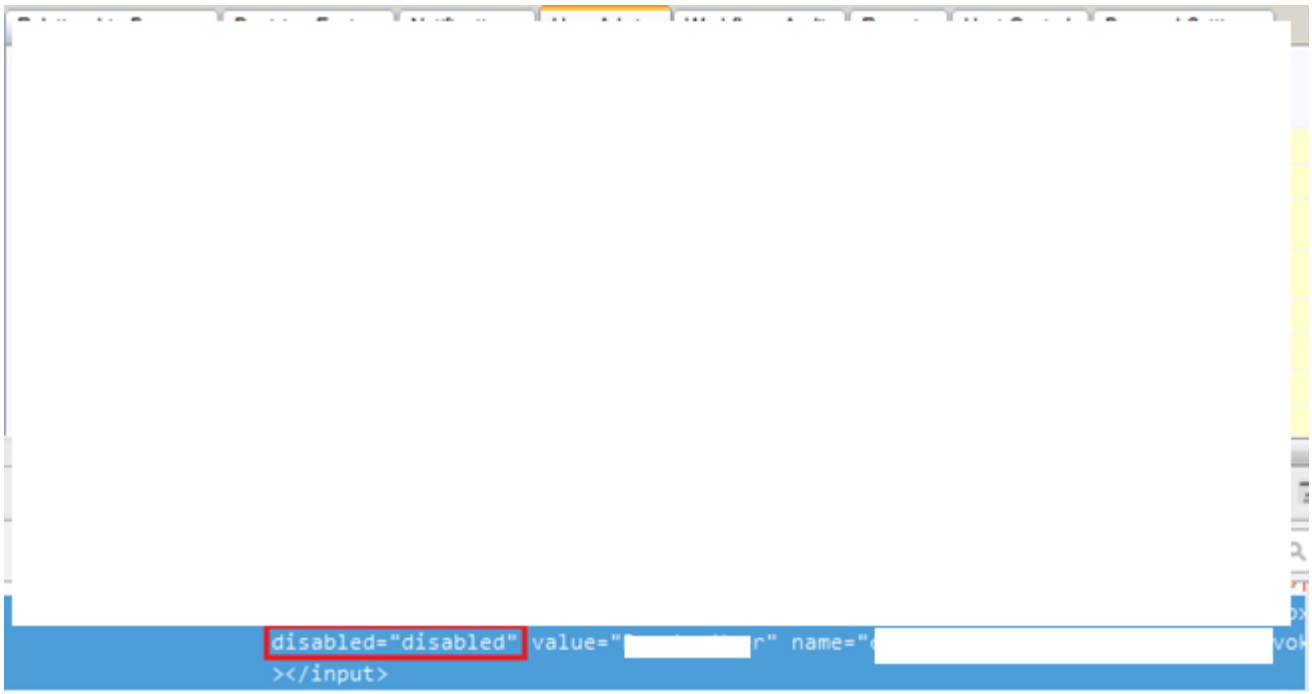
```
Private Sub bxxxxxe_Cxxxx(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
bXXXXXke.Click
    Try
        Dim dsXXXXXXX As New XXXXXXXXXXXXXes
    Const constXXXXXXvoke As String = "SXXXXXXXXXXXXXXXXXXXXXXXXXXXE"
```

Sample Report

```
Dim objXML As New XmlProcedures

objXML.AddXmlData(New XmlData)
objXML.XmlData(0).AddXmlDataField(New XmlDataField("SXXXXXXXXXD", SXXXXXXD.Value))
dsXXXXXXX.EXXXXXXXure(PXXXXXXXX.Name, coXXXXXXoke, objXML.GetXml)
```

Proof of Concept



Proof of vulnerability

Fix Recommendations

- It is suggested to implement server-side access control mechanism instead of relying client side access control implementation.

7. Appendix

7.1 PoC (Proof of concept) code for XXE (XML External Entity) vulnerability

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Sample Report