

Sample Report for Web Application

Penetration Test

<Target_Name>

December 2021

Taskmaster: <Your Company's Name>

Performed and Reported by: Raspina Net Pars Co. Ltd.



<https://rnpg.ir>



Raspina Net Pars Co. First Floor, No.13, Bozorgmehr Street, Valiasr Street, Tehran, Iran



info@rnpg.ir

Document Specifications

Title	Penetration Test Final Report - <Target_Name>
Version	1.0
Classification	Classified
Created by	Mohammad Reza Ismaeli Taba (Senior Penetration Tester)
Reviewed by	Alireza Mostame (CTO)
Approved by	Mohammad Reza Mostamea (CEO)

Table of Contents

Contents	Page Number
Document Specifications.....	2
Project's Specification.....	4
1.Project's Contact Information	4
2.Project's Timetable	4
3.Project's Scope.....	4
Introduction	5
Management Summary.....	6
Summary of Detected Vulnerabilities	7
Methodology and Approach	8
1-Information gathering	8
2- Configuration and Deployment Management Testing.....	8
3-Assessing the identity management mechanisms in the application	8
4-Authentication testing	8
5-Authorization testing.....	9
6-Assessing Security in Session Management	9
7-Assessing input validation security mechanisms	9
8-Error Handling Testing	9
9-Testing for Weak Cryptography	9
10-Business logic Testing	9
11-Client-Side Security Testing	9
12- Summarizing and reporting.....	9
1. Management Summary.....	10
2. Details of discovered vulnerabilities	10
Details of discovered vulnerabilities	11
Standard Query Language Injection (Blind SQLi)	11
Authentication Bypass Through Insecure JWT	18
XML Injection (XXE).....	23
Cross Site Scripting (Reflected XSS)	29

Project's Specification

1. Project's Contact Information

Product Owner (Employer)	Contractor
Project Manager Mr. / Ms. <Your Project Manager's Name>@<Your Company's Domain>.com	Project Manager Mohammad Reza Mostamea CEO@rnpg.ir
Project's Supervisor Mr. / Ms. Auditor@<Your Company's Domain>.com	Project's Technical Manager Mr. / Ms. Example@rnpg.ir

2. Project's Timetable

Project's Objectives	Start Date	Finish Date
Conducting <Target_Name> Penetration Test	12/22/2021	02/07/2022

3. Project's Scope

In this section, the agreed upon scope for the execution of the penetration test/security assessment project is specified. In the following table, agreed upon URLs for conducting project, are demonstrated.

Project's Scope
Agreed upon project scope

Introduction

Given the increasing prevalence of cyber-attacks targeting the organizations' critical infrastructure through IT platforms, the focus of IT department of the organizations must be on identifying and neutralizing threats and vulnerabilities in this perimeter, providing a safe and secure environment for both users and organizations to thrive in. The penetration test is in fact an authorized attempt to help assessing the level of cyber security in information technology infrastructures.

The purpose of web application penetration test is to identify all of the possible vulnerabilities in a web application and then exploit the identified vulnerabilities. Also in this process, an attempt is made to perform a detailed evaluation of the effectiveness and efficiency of the defense mechanisms used for securing the web application.



The goal of Raspina Net Pars's penetration testing service is to assist organizations in identifying and addressing cyber security threats. One of the most distinctive features of our service is its close resemblance to real-world cyber threats imposed by real-world hackers through the touch of our sophisticated penetration testing team: It's not just a vulnerability assessment! Besides, penetration testing and assessment process are iterated after the vulnerability reports are submitted and issues are patched, at no extra cost, to ensure the utter neutralization of found vulnerabilities.



The purpose of this document is to acquaint the prospective customers of Raspina Net Pars Company with an example of the final report for web application penetration testing projects. Therefore, all the vulnerabilities mentioned in this document belong to our dedicated laboratory web applications and are not related in any ways to any organization or company.

Management Summary

In this section of the final report, the characteristics and impacts of the most sensitive and dangerous vulnerabilities are briefly described. In the following, results of the penetration testing project are displayed in the form of graphs and tables. Other notable items in this section include the total number of all vulnerabilities detected during the project and the methodology and standards used in the process. Finally in Figure 1, the overall status of the tested environment in terms of vulnerability to cyber security threats is displayed.

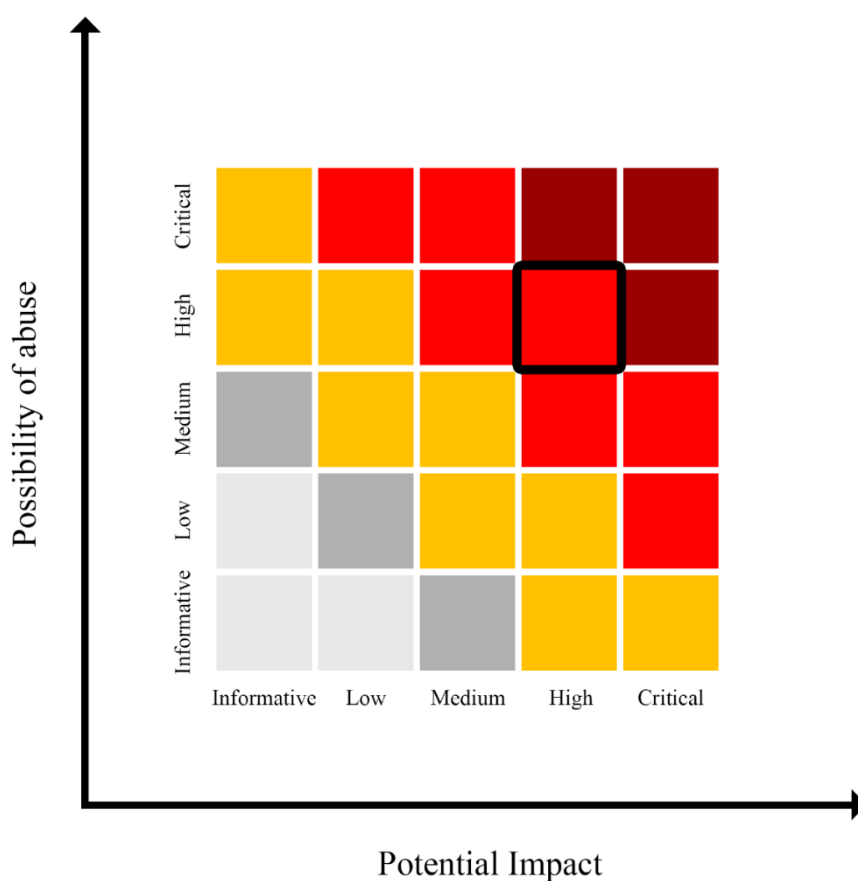


Figure 1 - <Target_Name> Overall Status in terms of vulnerability to cyber security threats

Summary of Detected Vulnerabilities

In this section, the titles of the discovered vulnerabilities along with their corresponding severities are shown.

		2	2	0	0	4
		Critical	High	Medium	Low	Total
Title						Severity
Blind SQL Injection						Critical
Authentication Bypass Through Insecure JWT						Critical
XML Injection (XXE)						High
Cross Site Scripting (Reflected XSS)						High

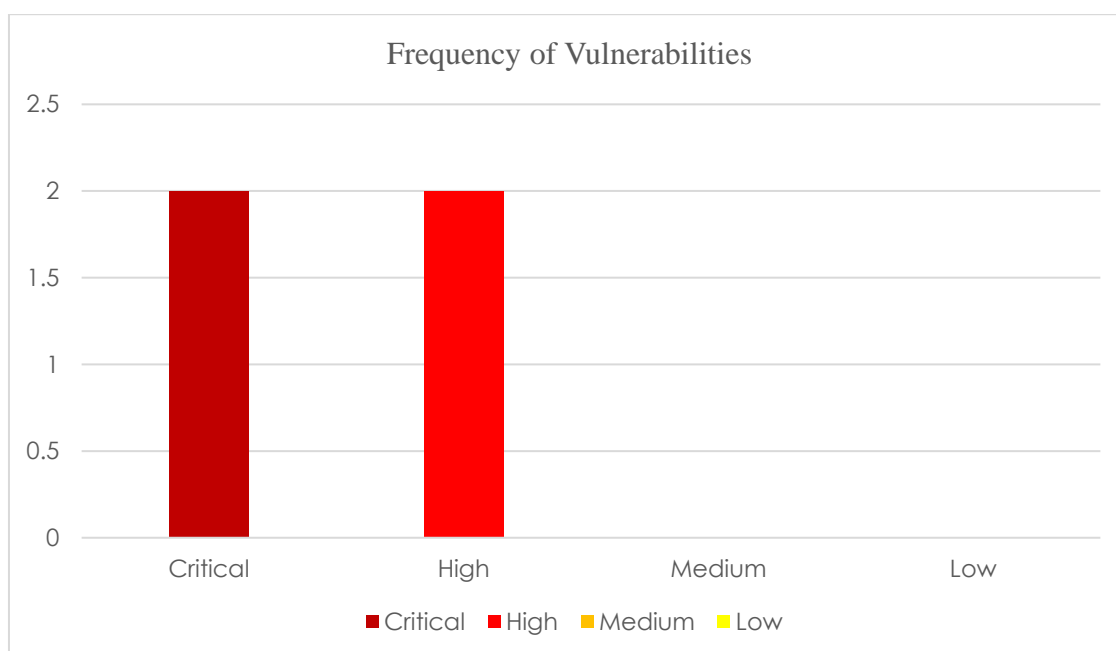


Figure 2- Frequency of discovered vulnerabilities with respect to their severity

Methodology and Approach

Web application penetration testing service is designed using a comprehensive, risk-based approach to identify application's vulnerabilities at all levels. This approach consists of 12 general steps and 105 sub-steps, which is designed by following a variety of standards and best practices, including WSTG (version 4.2) and ASVS (2019 version).

Experience has shown us that automated assessment tools are effective for performing the early stages of penetration testing, while a comprehensive, complete and accurate security assessment can only be performed using human intelligence and manual assessments. Accordingly, about 90% of the tests are done manually and 10% of the evaluations are done by commercial and automated tools. This approach focuses on manual assessments to identify logical and complex vulnerabilities as well as the use of automated tools to achieve the best results in the shortest time.

In the following, we describe each component of the intended approach.

1-Information gathering

The first step in performing a penetration test is to accurately collect information about the target. The attacker can use collected information to further expand his/her attacks.

2- Configuration and Deployment Management Testing

Knowing the settings of the web hosting server, the ports that can be connected to it, as well as the security issues observed in them is very important, which is examined in this step.

Among the actions that can be done in this section are examining how different parts of the application communicate with each other including databases, identify vulnerabilities due to misconfiguration in the operating system, identify unreferenced files, check for existence of old backup files and check for HTTP methods used.

3-Assessing the identity management mechanisms in the application

The processes associated with registering users with different access levels and the roles defined for each of them can create various security risks, including the intruder entering the system illegally and stealing sensitive information, which we will examine in this step.

4-Authentication testing

In this step, the application authentication mechanism is fully tested and in case of existing vulnerability we try to exploit it in order to gain unauthorized access to other parts of the application.

5-Authorization testing

Authorization is a process in which each user accesses a certain number of resources according to the level of access defined for him/her. Ensuring the correct implementation of the Authorization is one of the most important issues. Therefore, in this step, several tests are performed to identify such vulnerabilities.

6-Assessing Security in Session Management

One of the key parts in designing any web application is to use a mechanism to handle the sessions and keep the user connected to it. Misuse of authorization, increase of access level or unwanted actions are among the cases that we will examine in this step so that attackers cannot endanger the security of application users.

7-Assessing input validation security mechanisms

The most common weakness of web application is placed in this section. Data entry by users is an inevitable part of any application that must be designed properly to prevent attackers from accessing to unauthorized information. Attackers can use these type of security vulnerabilities to compromise the security of information in the system as well as the security of other users.

8-Error Handling Testing

In case of an error occurring in the application, it is necessary to detect the error in a managed manner and respond to it otherwise there is a possibility of information leakage. In this step we will examine applications error handling mechanism.

9-Testing for Weak Cryptography

The use of insecure cryptographic algorithms can lead to the leakage of sensitive information. In this step, we test the weaknesses in the cryptographic algorithms used in the application.

10-Business logic Testing

Each application consists of several sections, and each section follows its own logic that is related to other sections. Insecure design or lack of proper communication between different sections can lead to the inability of application to provide services. In this section, we examine the security vulnerabilities associated with application design logic.

11-Client-Side Security Testing

Today, various items are controlled in Client-side and also various items are stored, processed and executed by browsers, This, in addition to the many advantages, has created risks for users and the resources they use, which in this step we will identify these risks.

12- Summarizing and reporting

In Raspina Net Pars Company, this step is considered very serious and important. We provide all the steps of the penetration testing service process with the experiences, results and findings obtained by the technical team in a technical and customized report to our customers.

Vulnerability reports will be presented to our customers based on the OWASP standard and in an agreed upon format, along with stating the reasons and presenting practical evidence in an accurate manner. Each report contains multiple effective suggestions to remediate and neutralize the corresponding vulnerability.

The final report consists of two general sections, which include the following:

1. Management Summary

- A brief description of the results obtained
- Displaying the results in the form of statistics
- A brief description of the process performed and the used methodology

2. Details of discovered vulnerabilities

- Specifications of each vulnerability
- Description the vulnerability Detection process and how to exploit it.
- Providing Remediation to Patch the vulnerability.

In order to provide the best service to the customer, after delivering penetration test reports, Raspina Net Pars specialists will cooperate with technical teams on the customer side by providing efficient solutions to solve discovered security issues.

Details of discovered vulnerabilities

Vulnerability Table No. 1

Critical	Standard Query Language Injection (Blind SQLi)
Vulnerable Path	http://rnpg.lab/sqli/example1.php?name=user1
Vulnerable Param.	user
Payload	sqlmap -u "rnpg.lab/sqli/example1.php?name=root" -T users --dump
Attack Result	<pre>[06:19:27] [INFO] the back-end DBMS is MySQL back-end DBMS: MySQL ≥ 5.0.12 [06:19:27] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries [06:19:27] [INFO] fetching current database [06:19:27] [INFO] fetching columns for table 'users' in database 'exercises' [06:19:27] [INFO] fetching entries for table 'users' in database 'exercises' Database: exercises Table: users [4 entries] +-----+-----+-----+-----+-----+ id groupid age name passwd +-----+-----+-----+-----+ 1 10 10 admin admin 2 0 30 root admin21 3 2 5 user1 secret 5 5 2 user2 azerty +-----+-----+-----+-----+ [06:19:27] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output/rnpg.lab/dump/exercises/users.csv' [06:19:27] [INFO] fetched data logged to text files under '/root/.sqlmap/output/rnpg.lab' [*] ending @ 06:19:27 /2020-01-26/</pre>
CVSS 3.1 Score	10.0
CVSS 3.1 Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
CWE Code	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
OTG Code	Testing for SQL Injection (OTG-INPVAL-006)
CAPEC Code	CAPEC-7: Blind SQL Injection

Vulnerability Summary

According to *Vulnerability Table No. 1*, the vulnerability is easily identifiable and exploitable on the Raspina Net Pars's Experimental Lab, enabling attackers to access all information in the victim's database without having the necessary permissions. Basically, the attacker will be able to view, edit or even delete the information residing in the database. In addition, the attacker can exploit this vulnerability to remotely execute commands on the victim's operating system, which might have irreparable consequences.

Attack Description

SQL Injection is one of the most dangerous web application vulnerabilities that enables attackers to use different techniques (such as Union based, Blind, Boolean based, Error based, Time based, etc.) and execute malicious queries and consequently view, edit or delete information in the database of the vulnerable system. Such attacks occur when user input is not properly sanitized and the values entered by users are received without proper restrictions.

To implement this attack, the attacker first navigates to *Path No.1* and examines the user parameter (Figure 3).

Path No.1:

`http://rnpg.lab/sqli/example1.php?name=user1`

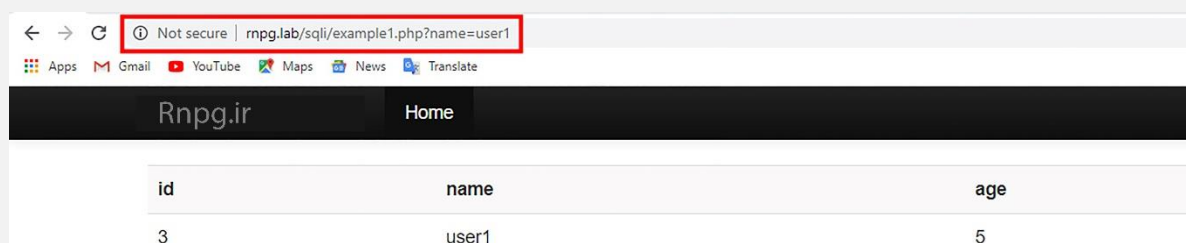
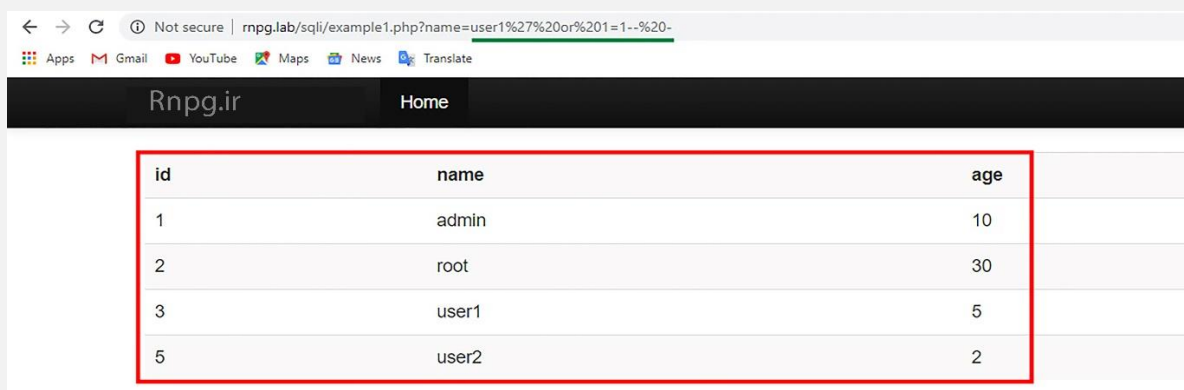


Figure 3 - Vulnerable parameter's path

In order to verify the vulnerability of the suspicious parameter, the attacker sends the *Payload No.1* to the database via the user input. This payload works in such a way that it extracts all the information in the database table using a condition that is always True. As shown in Figure 4, all of the user information is displayed in the table. With this in mind, the attacker ensures that the user parameter is vulnerable.

Payload No.1:

```
user1' or 1=1- --
```



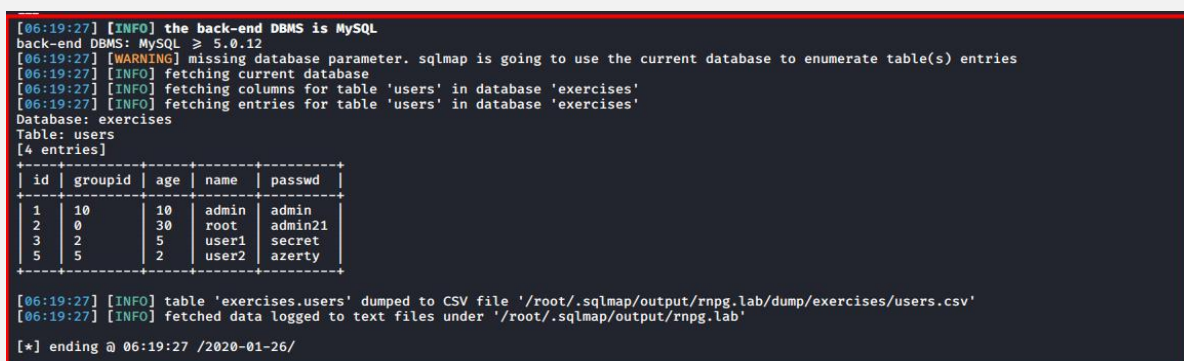
id	name	age
1	admin	10
2	root	30
3	user1	5
5	user2	2

Figure 4 – Verifying the vulnerability of “name” parameter

For the next stage, the attacker uses the “sqlmap” tool to extract more information about the underlying system. For this purpose, the attacker executes *Command No.1* to extract all the information in the User table. As shown in Figure 5, all of the database’s information including *id*, *groupid*, *age*, *name* and *passwd* have been successfully extracted from the table.

Command No.1:

```
sqlmap -u "rnpq.lab/sqli/example1.php?name=root" -T users --dump
```



```
[06:19:27] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12
[06:19:27] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[06:19:27] [INFO] fetching current database
[06:19:27] [INFO] fetching columns for table 'users' in database 'exercises'
[06:19:27] [INFO] fetching entries for table 'users' in database 'exercises'
Database: exercises
Table: users
[4 entries]
+----+-----+-----+-----+-----+
| id | groupid | age | name | passwd |
+----+-----+-----+-----+
| 1  | 10      | 10  | admin | admin  |
| 2  | 0       | 30  | root  | admin21 |
| 3  | 2       | 5   | user1 | secret |
| 5  | 5       | 2   | user2 | azerty |
+----+-----+-----+-----+
[06:19:27] [INFO] table 'exercises.users' dumped to CSV file '/root/.sqlmap/output/rnpq.lab/dump/exercises/users.csv'
[06:19:27] [INFO] fetched data logged to text files under '/root/.sqlmap/output/rnpq.lab'
[*] ending @ 06:19:27 /2020-01-26/
```

Figure 5 - Extracting information from the database

Remediation

1. Using Prepared Statements, along with binding the variable prevents this vulnerability and ensures that the attacker does not intend to change the query. Parametered queries force the developer to first define the SQL code and then pass each parameter to the query.

The .Net language uses parameterized queries such as SqlCommand () or OleDbCommand (). This method of coding allows the database to distinguish between code and data, regardless of what the user enters.

example:

Sending parameters into the Query, by calling. Parameters.Add ()

```
String query = "SELECT account_balance FROM user_data WHERE user_name
=?";
try {
    OleDbCommand command = new OleDbCommand(query, connection);
    command.Parameters.Add(new OleDbParameter("customerName",
CustomerName Name.Text));
    OleDbDataReader reader = command.ExecuteReader();
    // ...
} catch (OleDbException se) {
    // error handling
}
```

In Java, you can specify the type of the request using Prepared Statements.

```
String custname = request.getParameter("customerName");
// Checking user input
String query = "SELECT account_balance FROM user_data WHERE user_name =
? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

In PHP, you can use PDO to work with different entities of the database in the form of an api.

```
<?php
$stmt=$conn->prepare(INSERT INTO
MyGuests(firstname,lastname,email)VALUES(?,?,?));
$stmt->bind_param("sss",$firstname,$lastname,$email);
//set paramters and execute
$firstname="John";
$lastname="Doe";
$email="john@example.com";
$stmt->execute();
$firstname="Mary";
$lastname="Moe";
$email="mary@example.com";
$stmt->execute();
```

2. Sometimes developers use Stored Procedures for creating SQL Statements using automatically parameterized parameters. The difference between prepared statements and stored procedures is that the SQL code in the stored procedures is pre-defined and stored in the database and then called via the application. Although this approach does not always prevent SQL Injection, it is as effective as parameterized queries

Example:

In .NET:

```
Try
    Dim command As SqlCommand = new SqlCommand("sp_getAccountBalance",
connection)
    command.CommandType = CommandType.StoredProcedure
    command.Parameters.Add(new SqlParameter("@CustomerName",
CustomerName.Text))
    Dim reader As SqlDataReader = command.ExecuteReader()
    '...
Catch se As SqlException
    'error handling
End Try
```

In Java:

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
    CallableStatement cs = connection.prepareCall("{call
sp_getAccountBalance(?) }");
    cs.setString(1, custname);
    ResultSet results = cs.executeQuery();
```

```
// ... result set handling
} catch (SQLException se) {
// ... logging and error handling
}
```

In VB .NET:

```
Try
    Dim command As SqlCommand = new SqlCommand("sp_getAccountBalance", c
onnection)
    command.CommandType = CommandType.StoredProcedure
    command.Parameters.Add(new SqlParameter("@CustomerName",
CustomerName.Text))
    Dim reader As SqlDataReader = command.ExecuteReader()
    '...
Catch se As SQLException
    'error handling
End Try
```

3. Using NamedQuery in Java, you can receive a request as a Prepared Statement and create the according query for it.

```
Query query=em.createNamedQuery('User.findByUserId');
query.setParameter('userId', user);
List users=query.getResultList();
```

4. By validating the parameters, SQL Injection can be prevented. For example, the application can only accept the desired parameters including and limited to letters, numbers and spaces. One of the best ways to prevent this vulnerability is to identify unauthorized characters within the string and convert them to allowed characters.

For example, developer can define a function that receives user input and converts characters that lead to a database error to allowed characters in an array.

```
<?php
function BlockSQLInjection($str)
{
return str_replace(array("'", '"', "'", '"'), array("'", '"', '"', $str));
}
?>
```


It should be noted that the symbols that lead to database errors, such as (' , ">, |,;), (etc.), must be checked on the server side.


5. A short-term method to prevent some SQL injections is to apply type casting to data types. That is, specify the type of data you intend for the variable, for example: (integer, string, Boolean, etc.)

Example:

```
protected void Page_Load(object sender, EventArgs e)
{
    var connect =
    ConfigurationManager.ConnectionStrings["NorthWind"].ToString();
    var query = "Select * From Products Where ProductID = @ProductID";
    using (var conn = new SqlConnection(connect))
    {
        using (var cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.Add("@ProductID", SqlDbType.Int);
            cmd.Parameters["@ProductID"].Value =
            Convert.ToInt32(Request["ProductID"]);
            conn.Open();
            //Process results
        }
    }
}
```

6. One of the ways to reduce the consequences of this vulnerability is to allocate the user with the minimum level of access to the database account. DBA or admin access level should not be defined as the main and current user of the database.
7. It is recommended to use ORM and frameworks such as Dapper and Ado.net to exchange information. Because these frameworks and ORMs, considering the above, provide you with facilities and capabilities in order to transfer information in a secure environment.

Vulnerability Table No. 2

Critical Authentication Bypass Through Insecure JWT	
Vulnerable Path	Incoming Traffic to User Panel
Vulnerable Param.	jwt
Payload	eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0=.eyJ1c2VybmFtZSI6ImFkbWluIn0.LEQyK0K8LTVTi81LSaYm3oiEK56EFim5YHpI-zFRLRI
Attack Result	
CVSS 3.1 score	9.8
CVSS 3.1 Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
CWE Code	CWE-302: Authentication Bypass by Assumed-Immutable Data
OTG Code	Testing for Weak Encryption (OTG-CRYPST-004)
CAPEC Code	CAPEC-461: Web Services API Signature Forgery Leveraging Hash Function Extension Weakness

Vulnerability Summary

According to *Vulnerability Table No. 2*, an attacker could log in to the victim's account (high-privileged user) and view, edit, or even delete the victim's information. In the following, we will discuss how to exploit the vulnerability and implement the attack.

Attack Description

JWT is a token format that provides the exchange of system and web service information on a secure platform. The JWT signs the exchanged value with the Secret (using the HMAC algorithm) or the Private/Public key pair (using the RSA or ECDSA algorithms).

This vulnerability is emerged due to the lack of proper implementation of the JWT. In this way, the attacker is able to access the admin user area by changing the value of the algorithm (alg) and the "username" in the token.

To execute an attack, the attacker first navigates to the user login page. On this page, users can click on the "Login as Guest" button and login to the user area with the minimum access level (Figure 6).

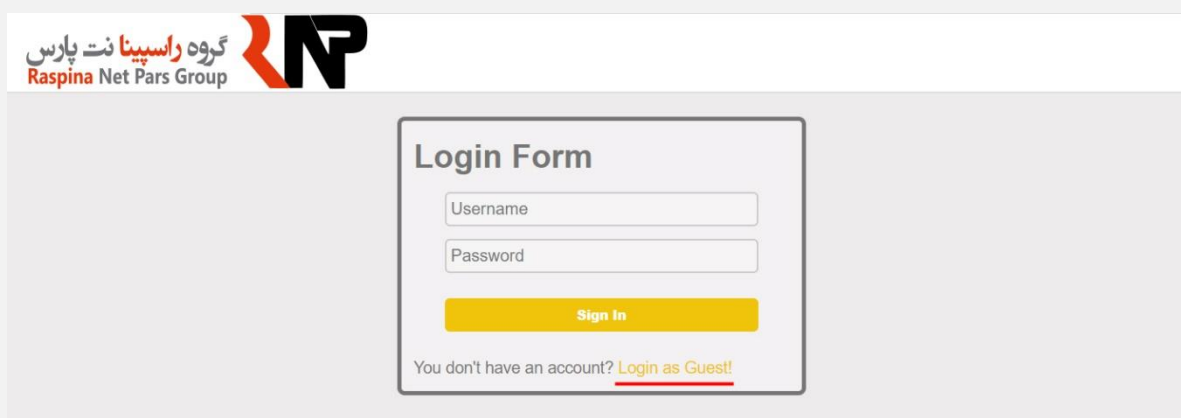


Figure 6 – Login to user panel

Now the attacker clicks on "Login as Guest!" button and receives the traffic using the Burp Suite tool. Attacker then sends the traffic to the Repeater tab of the tool (Figure 7).

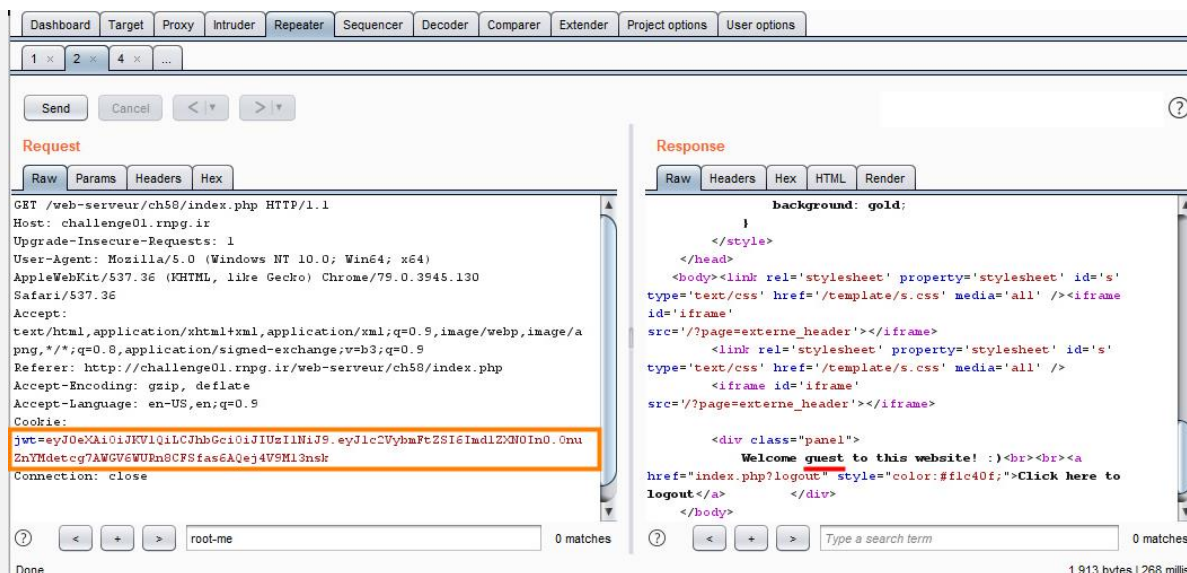


Figure 7 – Interception of incoming traffic to guest's panel

After interception of the traffic, the attacker sends the JWT value to the Burp's Decoder, decoding the first segment of the token. Attacker then changes the “alg” value to “none” and encodes the new value with the Base64 mechanism (Figure 8).

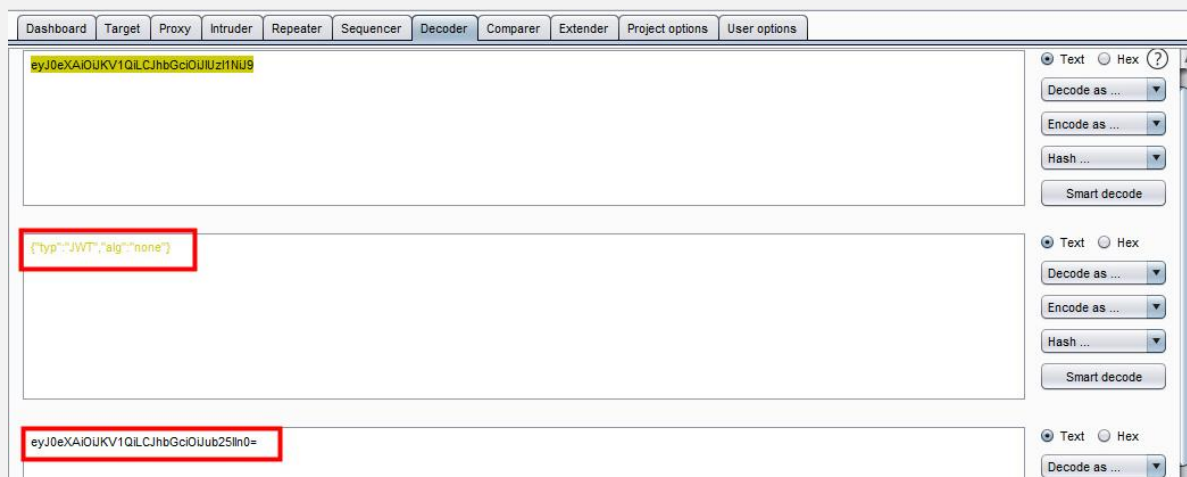


Figure 8 – changing the first segment of the JWT

In this stage, attacker sends the send segment of the jwt value to the Burp's Decoder and after decoding this value, he/she changes the username value to "admin" and encodes the new value with the Base64 mechanism (Figure 9).



Figure 9 - changing the second segment of the JWT

The attacker replaces the first and second sections with the original JWT value. As shown in Figure 10, the attacker successfully logged into the user panel with the admin access level and bypassed the authentication mechanism.



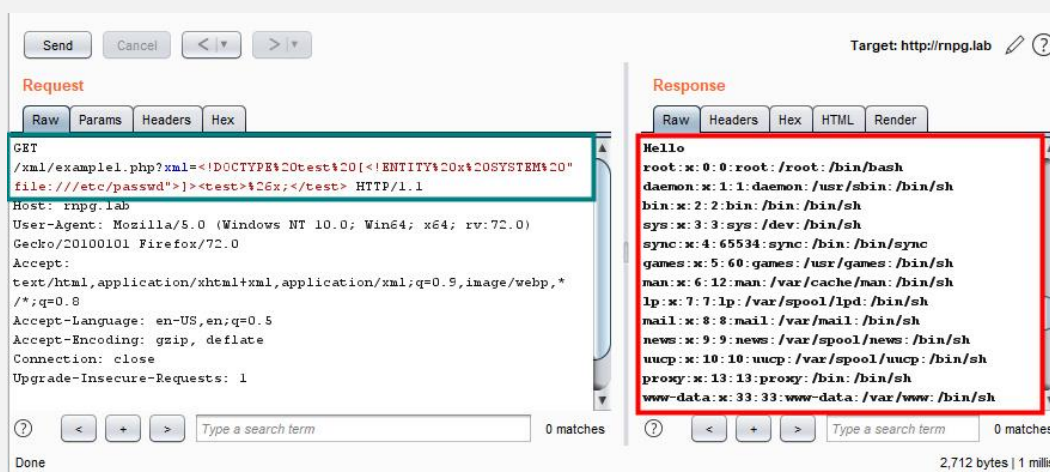
Figure 10 – Successful login with admin access level

Remediation

JWTs carry two JSON objects (Header and Payload) that contain valuable information. The header contains information about the algorithm used by the JWT to sign and encrypt the data.

1. Received JWTs must be reviewed and validated at the server before processing. In other words, only tokens containing known algorithms should be processed and “none” should not be accepted for the “alg” parameter.
2. JWT protects the exchanged value using Secret (using HMAC algorithm) or Private / Public key pairs (using RSA or ECDSA algorithm). Due to this, it is recommended to use asymmetric algorithms (RSA or ECDSA algorithms) instead of symmetric algorithms (HMAC algorithms).
3. Avoid using keys with small length.
4. Consider a complex and strong phrase for the Secret value.

Vulnerability Table No. 3

High	XML Injection (XXE)
Vulnerable Path	http://rnpng.lab/xml/example1.php?xml=%3Ctest%3ERaspinaNetPars%3C/test%3E
Vulnerable Param.	xml
Payload	!>DOCTYPE%20test%20[<!ENTITY%20x%20SYSTEM%20"file:///etc/passwd">]><test%26x;</test>
Attack Result	
CVSS 3.1 Score	8.6
CVSS 3.1 Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N
CWE Code	CWE-611: Improper Restriction of XML External Entity Reference
OTG Code	Testing for XML Injection (OTG-INPVAL-008)
CAPEC Code	CAPEC-221: XML External Entities Blowup

Vulnerability Summary

According to *Vulnerability Table No. 3*, the attacker is able to view all files on the victim's operating system, which violates the principle of confidentiality of information. In the following report, we will discuss how to implement and exploit this vulnerability.

Attack Description

XML injection vulnerability occurs when information entered by a user is placed in an XML file on the server side without any restrictions. This enables the attacker to inject XML documents into the user's input and execute these commands by an XML parser and consequently extract the desired information.

In order to exploit the vulnerability, the attacker first navigates to *Path No.2* (Figure 11).

Path No.2:

`http://rnpg.lab/xml/example1.php?xml=%3Ctest%3ERaspinaNetPars%3C/test%3E`



Figure 11 – Vulnerable Path

Attacker then intercepts the corresponding traffic using the Burp Suite tool Intercept and sends it to the Burp's Repeater (Figure 12).



Figure 12 – Interception of request to visit vulnerable parameter

Now the attacker inserts the *Payload No.2* inside the vulnerable xml parameter to extract the usernames of the Raspina Net Pars operating system and sends the corresponding request. As depicted in (Figure 13), all of the usernames of the system have been extracted.

Payload No.2:

```
!>DOCTYPE%20test%20[<!ENTITY%20x%20SYSTEM%20"file:///etc/passwd">]><test%26x;</test>
```

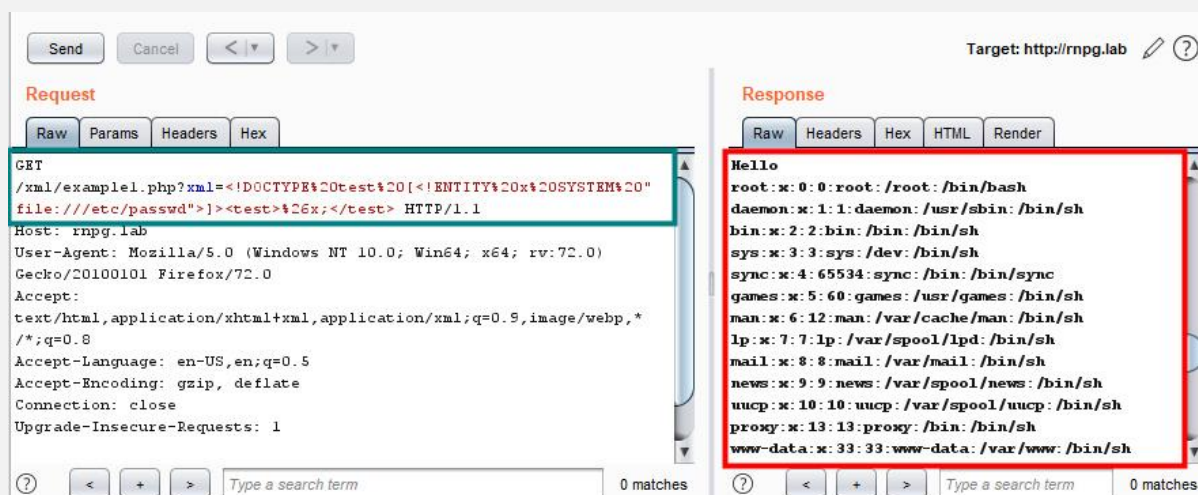


Figure 13 – Inserting malicious payload into vulnerable user input parameter

Remediation

To prevent this vulnerability, you should do the following according to your programming language:

C/C++

libxml2

- Enum xmlParserOptio: Should not have the following settings:
 - XML_PARSE_NOENT
 - XML_PARSE_DTDLOAD

libxerces-c

- Use XercesDOMParser to prevent these attacks:

```
XercesDOMParser *parser = new XercesDOMParser;
parser->setCreateEntityReferenceNodes(false);
```

- Use SAXParser to prevent these attacks:

```
SAXParser* parser = new SAXParser;
parser->setDisableDefaultEntityResolution(true);
```

- Use SAX2XMLReader to prevent these attacks:

```
SAX2XMLReader* reader = XMLReaderFactory::createXMLReader();
parser->setFeature(XMLUni::fgXercesDisableDefaultEntityResolution,
true);
```

Java

StAX parsers, such as XMLInputFactory, allow us to configure many features and options.

- To protect XMLInputFactory against XXE, do the following:

```
xmlInputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, false);
// disable external entities
xmlInputFactory.setProperty("javax.xml.stream.isSupportingExternalEntities", false);
```

- To protect javax.xml.transform.TransformerFactory against XXE, do the following:

```
TransformerFactory tf = TransformerFactory.newInstance();
tf.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
tf.setAttribute(XMLConstants.ACCESS_EXTERNAL_STYLESHEET, "");
```

- To protect javax.xml.validation.Validator against XXE, do the following:

```
SchemaFactory factory =
SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
Schema schema = factory.newSchema();
Validator validator = schema.newValidator();
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
```

- To protect javax.xml.validation.SchemaFactory against XXE, proceed as follows:

```
SchemaFactory factory =
SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
factory.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
factory.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
Schema schema = factory.newSchema(Source);
```

- To protect javax.xml.transform.sax.SAXTransformerFactory against XXE, proceed as follows:

```
SAXTransformerFactory sf = SAXTransformerFactory.newInstance();
sf.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
sf.setAttribute(XMLConstants.ACCESS_EXTERNAL_STYLESHEET, "");
sf.newXMLFilter(Source);
```

- To protect org.xml.sax.XMLReader against XXE, do the following:

```
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
// This may not be strictly required as DTDs shouldn't be allowed at all, per previous line.
reader.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
reader.setFeature("http://xml.org/sax/features/external-general-entities", false);
reader.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
```

- To protect org.dom4j.io.SAXReader against XXE, proceed as follows:

```
saxReader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
saxReader.setFeature("http://xml.org/sax/features/external-general-entities", false);
saxReader.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
```

- To protect org.jdom2.input.SAXBuilder against XXE, do the following:

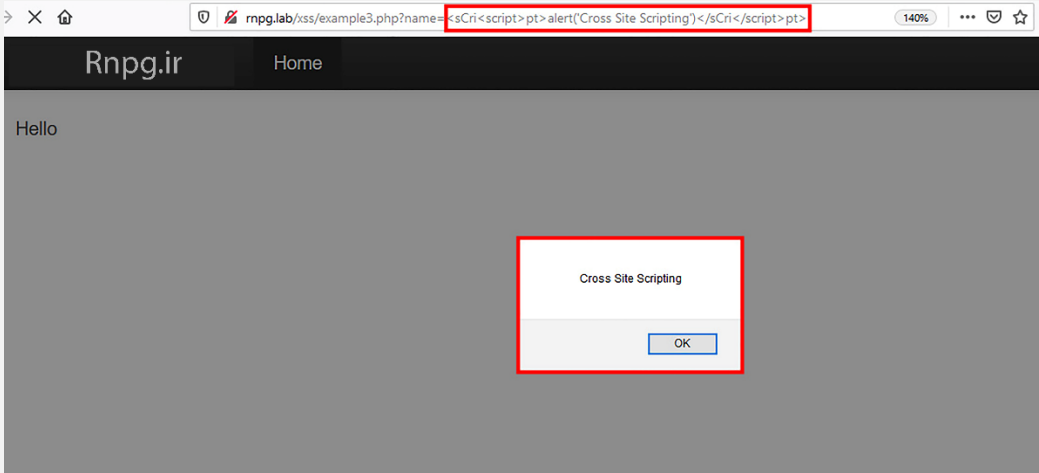
```
SAXBuilder builder = new SAXBuilder();  
builder.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);  
builder.setFeature("http://xml.org/sax/features/external-general-entities", false);  
builder.setFeature("http://xml.org/sax/features/external-parameter-entities", false);  
Document doc = builder.build(new File(fileName));
```

PHP

When using PHP XML parser, the following code should be applied to each PHP document to prevent such attacks:

```
libxml_disable_entity_loader(true);
```

Vulnerability Table No. 4

High	Cross Site Scripting (Reflected XSS)
Vulnerable Path	http://rnpng.lab/xss/example3.php?name=RaspinaNetPars
Vulnerable Param.	name
Payload	sCri<script>pt>alert('Cross%20Site%20Scripting')</s> <Cri</script>pt
Attack Result	
CVSS 3.1 Score	7.5
CVSS 3.1 Vector	CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:H
CWE Code	CWE-87: Improper Neutralization of Alternate XSS Syntax
OTG Code	Testing for Reflected Cross Site Scripting (OTG-INPVAL-001)
CAPEC Code	CAPEC-591: Reflected XSS

Vulnerability Summary

According to, *Vulnerability Table No. 4* the attacker is able to exploit the vulnerability on the "Raspina Net Pars experimental system" to attack all the victims of this system and edit, delete or even steal their information. In the following, the report will discuss how to exploit the vulnerability and implement the attack and also means to remediate this vulnerability are discussed.

Attack Description

A Reflected Cross site scripting attack occurs when malicious JavaScript code is injected into a victim browser. To implement the attack scenario, the attacker uses a variety of social engineering methods (sending emails, text messages, etc.) to send the links containing malicious code to the victim and also to trick the victim into clicking on the malicious link.

To implement the attack, the attacker first navigates to *Path No.3* (Figure 14).

Path No.3:

`http://rnpq.lab/xss/example3.php?name=RaspinaNetPars`

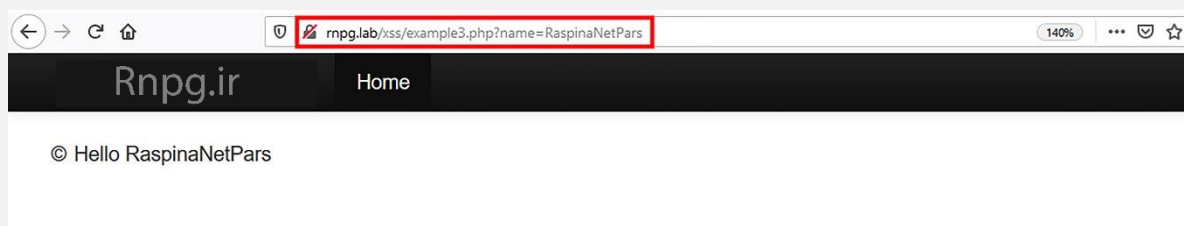


Figure 14 – Vulnerable Path

The attacker then checks the "name" parameter. During the checks, it turns out that the "name" parameter is vulnerable to Cross Site Scripting (XSS) attack. Due to this, the attacker creates a malicious payload (*Payload No.3*) by considering the filter that has been applied to this parameter, and puts it in the vulnerable parameter, as shown in Figure 15.

Payload No.3:

```
<sCri<script>pt>alert('Cross%20Site%20Scripting')</sCri</script>pt>
```

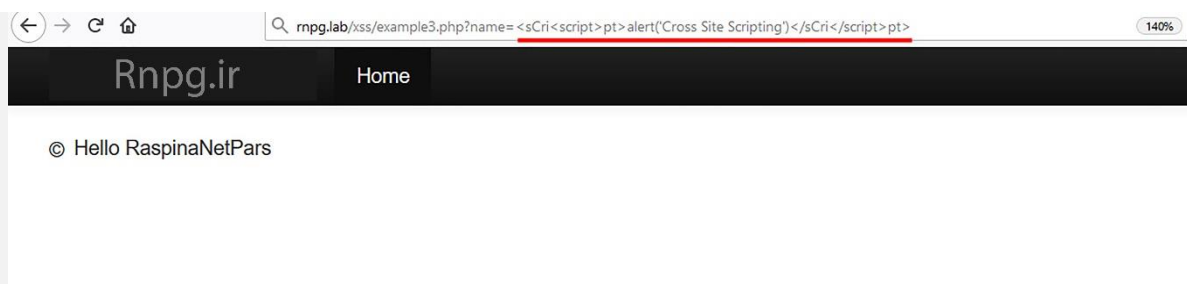


Figure 15 – Inserting malicious payload

The attacker now sends the link containing the malicious code to the victim using one of the social engineering techniques. As soon as the malicious link is opened, the payload is executed and the victim will see a pop-up message like the one in Figure 16, indicating that aforementioned parameter is vulnerable.

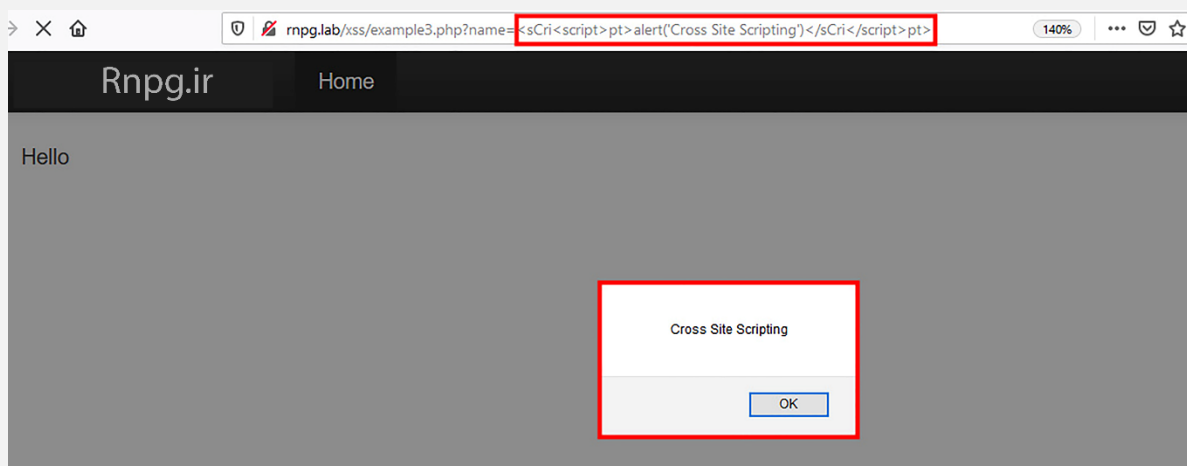


Figure 16 – Successful execution of malicious payload in victim's browser

Remediation

1. Consider the appropriate location for storing user input data inside the main body of the application is mandatory. If the data is not stored properly, the possibility of XSS vulnerabilities increases. for example:

The data entered by the user should not be placed between known tags such as <script>, <style>, etc.

```
<script>...User_Input...</script>
```

2. Before placing the user input data, inside the known tags such as <body>, <div>, , etc., the input data must be checked and sanitized properly. Only if the data is valid, it should be placed between the mentioned tags.

example:

```
<div>  
...Sanitize and check carefully before inserting user input data here...  
</div>
```

3. Data that contains symbols such as %, *, +, -, /,;, <, =,>,etc. must be checked for not being malicious prior to insertion into intended locations.
4. Before insertion of URL or data inside the “href” attribute by the user, the input data must be checked.

example:

```
<a href="http://www.somesite.com?test=...make sure the input data is  
sanitized before inserting it in this location..."> link </a>
```