Strike

Monday, August 1st, 2022

Company name

Web Application Penetration Test

For more information contact:

Javier Bernardo

Santiago Rosenblatt

Table of Contents

Document Revision History

Revision	Section	Description	Editor	Date
1.0	Whole document	First version of report	Javier Bernardo	July 25, 2022
1.1	Appendix B	Added completion of task	Javier Bernardo	July 31, 2022
1.2	Appendix B	Added coverage of initial test scenarios	Santiago Rosenblatt	August 1, 2022

Strikers

Javier Bernardo

Santiago Rosenblatt

Strike

Executive Summary

Executive Summary

[Test Client] has requested Strike to perform a Penetration Test on [Test APP].

Our main objective is to evaluate the security (overall) of the system from a [Black / Grey / White Box] perspective. This includes:

- Recognizing vulnerable areas that may be exploited by a malicious user.
- Understanding the ability that the application has to withstand common attacks and patterns.

Amount of findings discovered by Strike: [Total Amount of Findings].

Findings filtered by severity:

Critical	1
High	1
Medium	1
Low	0
Informative	0

Remediations and retesting

Strike validates that 2 medium severity findings were fixed and that the severity of:

- 1 high finding was reduced to medium.

By working in the pentest of [Test App], Strike identified that security measure were effective in resisting common attack patterns like:

- Server Misconfiguration.
- Rate Limit.
- Improper Session Management.

Strike

Scope

Scope

The scope of [Test App] Penetration Test was [limited / not limited] to specific components and interfaces. The scope covered the following URL(s):

```
https://testapp.testclient.com/
```

```
https://api.testclient.com/
```

Attack Vectors to Prioritize

The attack vectors that were prioritized by the client were:

- Injection
- Broken Access Control

Assesment Notes

Credentials / Roles where configured to:

prod_client	Client With Stock Hours In Profile, With Credit Card Configured And With All Privileges
stg_testing	Client With Less Privileges (Only 'View' Privileges)
basic_client	Client Without Payment Details And Without Stock Hours

The following points were left out of scope:

- User registration flow

Objectives

> ***Our main objective is to test our platform before launch**. As mentioned previously, we still are in private beta. But in March we will be launching public.*
>
> *Before October we need to:*
>
> - *Have the specified endpoints tested, as those are the core of our system.*
> - *Be able to scale and keep our data safe.*

> *As you may know, many companies define their cybersec budget in June once again. So as a cybersec company, our platform needs to be ready to accept high volumes of traffic (feel free to attempt DOS).*
>
> *Finally, we need to be sure that our payments gateway (logic where clients buy stock hours) works smoothly. We dont want users to get stock hours without paying.*

Strike

Methodology

Methodology

Overview

Strike pentesting methodology is aligned with industry security standards like: OWASP, OSSTMM, ISO27000, NIST 800-115, etc.

Depending on the objectives and the scope of the pentest, our certified Strikers (Pentesters) will follow these methodologies to perform a security assessment against the target, in order to define and prioritize the tasks required to manage enterprise security and to help prepare for different regulatory compliance goals or audits.

Using these methodologies an organization can demonstrate compliance with multiple regulations, including: PCI DSS, HIPAA, Sarbanes-Oxley and global standards, such as GDPR.

These assessments involve a deep automated scan using automated scanning tools to discover common vulnerabilities, as well as manual testing. Manual testing includes validation of all issue types covered under the automated scan as well as checks for problems not typically found by automated scanners such as authentication, authorization and business logic flaws (see: OWASP Top 10 and OWASP Mobile Top 10).

The severity assigned to each vulnerability is calculated using the NIST 800-30 Revision 1 and with industry-standard references including: CVSS, CVE, CWE, etc. This standard determines the risk posed by an application based on the likelihood an attacker exploits the vulnerability and the impact that it would have on the business.

Risk assesment

The severity of a vulnerability will be calculated using the impact that the vulnerability would have on the business if it was successfully exploited, and the likelihood of an attacker exploiting the vulnerability.

Impact

Technical impact can be broken down into factors aligned with the traditional security areas of concern: confidentiality, integrity, availability, and accountability.

The business impact requires a deep understanding of what is important to the company running the application. These are common factors for many businesses: financial damage, reputation damage, non-compliance, privacy violation.

The Impact is rated using the following values:

- **Critical:** Successful exploitation of the vulnerability may cause multiple catastrophic effects over the organization or other organizations.

- **High:** Severe service degradation. The organization is not able to perform primary functions or may cause damage to organizational assets.
- **Medium:** The organization is able to perform its primary functions, but the capabilities are reduced with a negative impact over organizational assets.
- **Low:** Limited degradation of the service, the effectiveness is noticeably reduced and the vulnerability may have minor impact to organizational assets
- **Informative:** The vulnerability could be negligible or just provide information about the systems.

Likelihood

Likelihood stands for the difficulty of exploiting the vulnerability considering the required skill level and the amount of access necessary to exploit the vulnerability.

The likelihood is rated using the following values:

- **Critical:** An attacker is almost certain to exploit the vulnerability.
- **High:** The vulnerability is very obvious, easily accessible and could be exploited without any special skill.
- **Medium:** The vulnerability requires some hacking knowledge to be exploited or access is restricted.
- **Low:** Exploiting the vulnerability requires application access, significant time, resources or a specialized skill set.
- **Minimal:** Adversaries are highly unlikely to leverage the vulnerability.

Severity

The vulnerability severity is calculated with the likelihood and impact weights, using following table:

		Impact				
		Minimal	Low	Medium	High	Critical
Likelihood	Critical	Minimal	Low	Medium	High	Critical
	High	Minimal	Low	Medium	High	Critical
	Medium	Minimal	Low	Medium	Medium	High
	Low	Minimal	Low	Low	Low	Medium
	Minimal	Minimal	Minimal	Minimal	Low	Low

Strike

Findings

Findings

Summary of findings

Finding	Likelihood	Impact	Severity	Status
SQL Injection	Critical	Critical	**Critical**	Open
XML Entity Expansion (Billion Laughs Attacks)	Medium	Critical	**High**	Open
Excessive Session Timeout Duration	Medium	High	**Medium**	Open
Weak Password Policy	Medium	High	**Medium**	Open
Login Page Username Enumeration	High	Low	**Low**	Open
Self-Signed X.509 Certificate (SSL/TLS)	High	Low	**Low**	Open
Lack of Multi-Factor Authentication	Low	High	**Low**	Open
Session Fixation	Low	Medium	**Low**	Open
Vulnerable Third-Party Libraries in Use	Low	Low	**Low**	Open

Critical Severity Findings

1. SQL Injection

Instance

```
https://testapp.testclient.com/test?id=1&order=1
```

Description

The application executes SQL queries generated by dynamically concatenating user-supplied data to static SQL query strings. In this implementation, the database has no way of distinguishing between the developer's intended SQL syntax and any SQL syntax included from user-supplied input. As a result, any user-supplied data containing SQL syntax inserted into the static query will be interpreted and executed. This leaves the application vulnerable to SQL injection since an attacker can inject SQL query syntax that intentionally alters the target query's structure.

SQL injection allows an attacker to modify the structure of a SQL query executed by the application. Depending on the type of SQL server in use, the attacker may be able to modify existing queries or append entirely new queries to the existing query. The modified query can access any portion of the database with the same entitlements that the database connection is granted, potentially leading to:

- Loss of confidentiality when an attacker gains access to unauthorized information.
- Loss of integrity through modification of other users' information, log files, and any other sensitive information.
- Authorization bypass from gaining access to or altering data in ways the application's business logic does not allow.
- Loss of availability when an attacker deletes other users' data, executes commands that take down the database server, or performs a denial of service attack that fills the database and subsequently exhausts the database server's storage.
- Authentication bypass through modification of SQL queries that verify a user's credentials.

Steps to reproduce

1. Setup your browser to direct traffic through the Burp Suite proxy tool.
2. Login to the application as a USER.
3. Navigate to Users.
4. Provide a value for the "Field Name" and select an "Operator".

5. Enter a value containing a single-quote in "Search Text" and click "Search".
6. Send the request to Burp Repeater.
7. Send the request and observe that the response says "Error occurred while trying to search person - Unclosed quotation mark after the character string..." which indicates that user input can alter the backend query.
8. URL encode the following payload which closes the dangling ' and forms a full query.

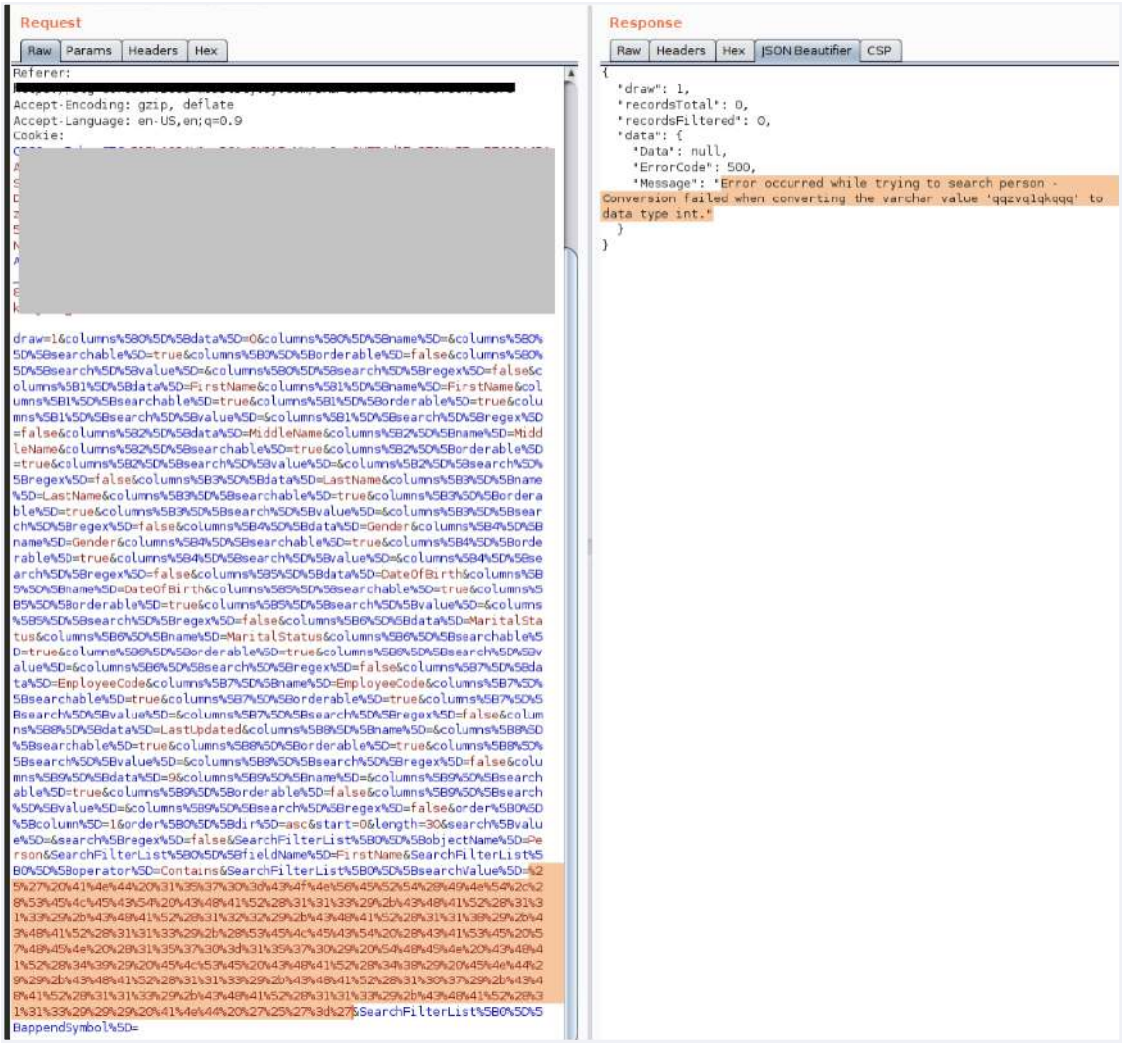
```
s' or SUBSTRING((SELECT 'a'+'b'+'c'), 3, 1)='c
```

9. Paste the encoded payload in the parameter search Value and send the request.
10. Observe that the application responds with valid search values indicating that there were no SQL syntax issues.
11. Repeat steps 9 to 11 with the payload below which produces an error in SQL execution based on which the SQL database is found to be Microsoft SQL Server 2012.

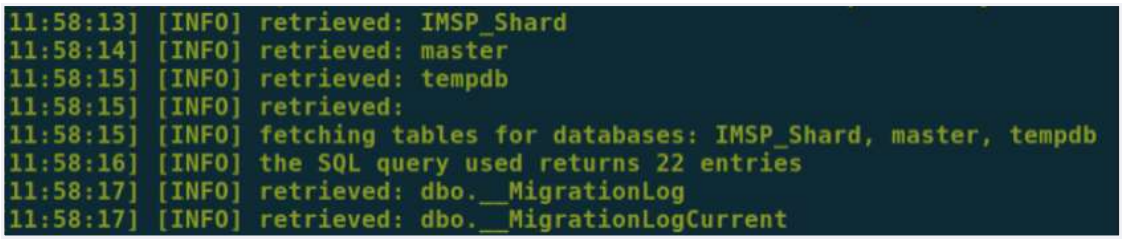
```
%' AND 1570=CONVERT(INT,(SELECT
CHAR(113)+CHAR(113)+CHAR(122)+CHAR(118)+CHAR(113)+(SELECT
(CASE WHEN (1570=1570)
THEN CHAR(49) ELSE CHAR(48)
END))+CHAR(113)+CHAR(107)+CHAR(113)+CHAR(113)+CHAR(113))) AND
'%'='
```

12. Run the sqlmap tool (https://github.com/sqlmapproject/sqlmap/) against the parameter and observe that SQL databases can be enumerated and/or dumped.

Evidences



Evidence 1



Evidence 2

Levels

Impact
Critical

Likelihood
Critical

Severity
Critical

Remediation

Rewrite all SQL queries constructed through dynamic concatenation to use an injection-safe query mechanism such as prepared statements with parameterized queries. Most modern programming languages provide a feature called "parameterized queries" that allow user-supplied data to be inserted safely as values in dynamic SQL queries. Rather than construct the dynamic SQL query by concatenating user-supplied data to static SQL query string fragments, data values are identified in the query by parameter markers or variables. Dynamic data is then passed through a mechanism provided by SQL that prevents the supplied data from changing the meaning of the query.

Note
The exact syntax and use of prepared statements with parameterized queries vary from language to language. The references below details general guidance for secure SQL query construction in .NET, Java, and PHP.

High Severity Findings

1. Stored Cross-Site Scripting (XSS)

Instances

```
https://testapp.testclient.com/Sites/AllItems.aspx
```

```
https://testapp.testclient.com/Sites/Documents/Forms/AllItems.aspx
```

Note
This finding is systemic throughout the application.

Description

A Stored Cross-Site Scripting (XSS) vulnerability occurs when a web application stores a string provided by an attacker and later sends this data to a victim's browser in such a way that the browser executes part of the string as code.

The string contains malicious data that can be interpreted as code by a browser, and this data is initially stored, often in the application's database. The application later retrieves the malicious data and inserts it into a web page without appropriate output encoding. This results in the victim's browser executing the attacker's code within the user's session.

Stored Cross-Site Scripting vulnerabilities give the attacker control of HTML and JavaScript running in the user's browser. The attack can alter page content with malicious HTML or JavaScript code. The attacker can alter any page content displayed to the victim, change page functionality (e.g., rewrite a login form to send credentials to an attacker instead), and log keystrokes entered on the page. If the victim is authenticated to the application, the attacker could hijack their session to read sensitive data accessible to the user and execute application functionality using the victim's identity.

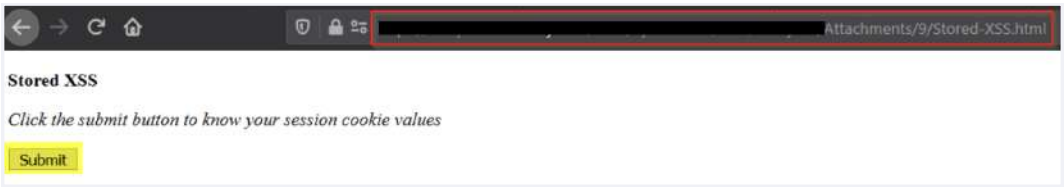
Note
Strike observed that the file-upload functionally allows HTML files to be uploaded. The uploaded files are rendered as part of the same domain. Thus, this functionality causes Stored Cross-Site Scripting and allows JavaScripts within the HTML file to access the domain contents of the application from the browser.

Steps to reproduce

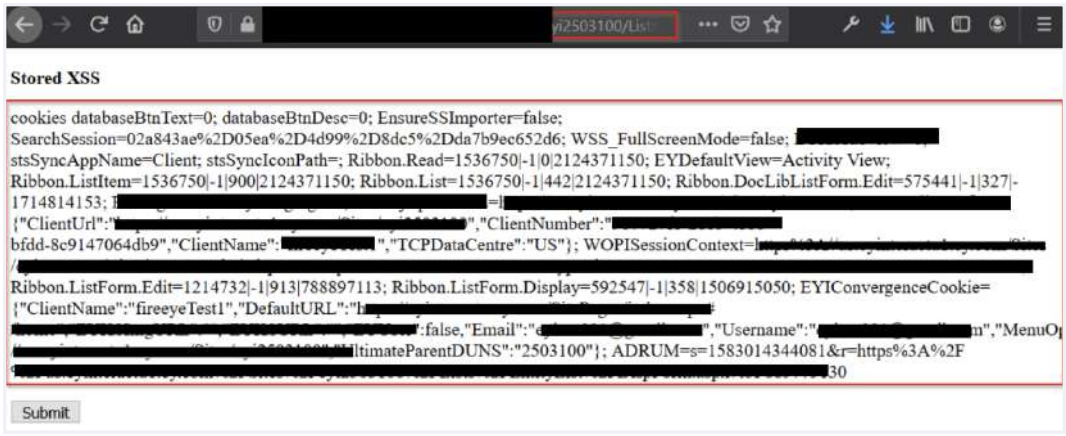
1. Log in to the application.
2. Copy the following contents and create a HTML file:

```
<!DOCTYPE html>
<html>
<body>
<p><b>Stored XSS</b></p>
<p id="name"><i>Click the submit button to know your session cookie values</i></p>
<button onclick="myFunction()">Submit</button>
<script>
 function myFunction() {
 document.getElementById("name").innerHTML = "cookies\n" + document.cookie;
}
</script>
</body>
</html>
```

3. Navigate to the URL (instance 1) mentioned in the Instances section.
4. Click ‘new item’, for instance: ‘Strike’.
5. Complete all the required fields, attach a HTML file and click 'Save'.
6. Click ‘new item’, for instance: 'Strike' .
7. Complete all the required fields, add entity 'Strike' in 'TCPEntityParent' section and click 'Save'.
8. Open the 'TCPEntityParent' assigned to 'Strike', in this case 'Strike'
9. Open the attached html file in the pop-up window and click 'Submit'.
10. Observe that the injected payload is successfully rendered by the browser



The view after clicking the HTML



Notice the injected payload is rendered succesfully by the browser within the application domain context.

Levels

Impact
High

Likelihood
High

Severity
High

Remediation

Stored Cross-Site Scripting (XSS) is prevented by encoding data before inserting it into the generated web page.

Each character of the data is encoded and the result string is then inserted onto the generated web page. This technique of encoding values before inserting them on the web page is called "Output Encoding". Output Encoding libraries exist for most popular programming languages and frameworks.

A web page has seven different output contexts and each output context requires a different encoding scheme. Data must be encoded using the proper scheme. The seven different contexts are:

- HTML Text Element
- HTML Attribute
- URL Parameter
- JavaScript Literal
- HTML Comment
- HTTP Header
- CSS Property

For example, the characters: <, >, ", ' are encoded as `&#60;`, `&#62;`, `&#34;`, `&#39;` for when those characters are inserted into an HTML Text Element. When those characters are inserted as a URL parameter, the same characters are encoded as `%3C`, `%3E`, `%22`, `%27`.

Libraries for implementing the encoding schemes exist for most popular programming languages:

- OWASP Java Encoder: Java only.
- AntiXssEncoder Class (System.Web.Security.AntiXss): .NET languages.
- Ruby - escapeHTML() - only supports HTML Text Encoding.
- Jgencoder in JQuery: for preventing DOM-based XSS.
- Green-field projects can consider the use of other technologies:
  - Google Capabilities based JavaScript CAJA.
  - OWASP JXT- automatically encodes string data with the proper encoding

Input validation is required in addition to output encoding whenever user data ends up in a non-data context, e.g., as the content of the href attribute in an anchor tag. In these situations, encoding will only help to prevent immediate code execution when rendering user data on the page. However, once a malicious link is clicked, the user data can influence the scheme invoked and thereby execute arbitrary JavaScript code by using the javascript: URL scheme. To mitigate this, strict input validation is required to ensure that only http(s) URLs are allowed. It should be noted that input validation still needs to be combined with output encoding. On its own, input validation is not sufficient to mitigate stored cross-site scripting, because two separate applications are often involved: the first stores the malicious string and a second application generates the web page. The application generating the page cannot assume that the application storing the value has filtered out malicious data.

Medium Severity Findings

1. Insufficient Authorization Checks Leading to Horizontal Privilege Escalation

Instances

```
https://api.testclient.com/api/services/app/User/Update
```

```
https://testapp.testclient.com/api/services/app/ProjectListing/GetAll?
uid=6ba61fd2-9e83-4519-bb80-f73db896dfdd&SkipCount=0&MaxResultCount=100
```

Description

The application lacks sufficient controls to prevent a user from accessing data or functionality that would not normally be accessible to them but would be accessible to other users of the same role.

For example, a user may be expected in the normal course of their work to perform transactions or view data for Company A, but the expectation is that there should be controls in place to prevent them from having the same access to Company B's data and functionality. A user is able to elevate their privileges horizontally when the controls in place to prevent this access are not present orthey do not function properly, allowing the user to access data or perform actions related to Company B.

When an application has insufficient authorization checks, this can lead to a horizontal privilege escalation attack yielding read-only access outside a user's authorization boundary and sensitive information available only to other users' accounts may be leaked. An attacker able to execute functionality that acts on unauthorized user assets may have any number of consequences including, but not limited to:

- Loss of data integrity via unauthorized creation, modification or destruction of data

Note
Regarding instance 1 of this finding, it appears that when you change a user's 'userName' it causes an issue with the application where it is no longer accessible to the user.

Steps to reproduce

1. Configure your browser to work with a proxy tool such as Burp Suite.
2. Login to the application.
3. Click on the hamburger menu in the top-right corner of the screen and select 'Userprofile'.
4. Click on the 'Date format' drop down menu and choose an option that is not currently selected, note that this is just to activate the 'Save' button on the page.

5. In Burp turn on 'Intercept'.
6. In the browser click the 'Save' button.
7. In Burp locate the intercepted request and forward it until you see the first instance mentioned in the 'Instances' section of this finding, note that the request we are looking for will use the 'PUT' HTTP method.
8. Right-click this request and choose 'Send to Repeater', then turn off 'Intercept'.
9. In Burp click on the repeater tab and choose the request that was just sent therein the previous step.
10. Change the value of the parameter 'id' to that of a non-existing user and then click 'Go'.
11. Observe in the response a message saying there is no user with that id.
12. Change the value of the parameter 'id' to that of a different existing user, then click 'Go'.
13. Observing the response indicates that the request was successfully made, showing that you can change the user profile of another user.

Levels

Impact
Medium

Likelihood
Medium

Severity
Medium

Remediation

Authorization should always be explicitly checked server-side to verify that the user making a request to view data or perform an action is authorized to do so.

This may be as simple as checking that a user is authorized to view a particular page in the application or checking that the user is authorized to perform an action overall. Many other more fine-grained situations may exist depending on the application context and the complexity of the business functionality. The following are some common situations where a more fine-grained authorization check is necessary:

- A user may be authorized to perform an action, but only against certain entities involved in the transaction. For example, a user may be authorized to perform a funds transfer from one account to another, but they may only be authorized to access certain accounts.
- Authorization checks must be performed to verify that the user is authorized to make a funds transfer as well as verify that the user is authorized to perform that transaction using the supplied accounts.
- Authorization checks must be performed to verify that the user is authorized to view account details as well as verify that the user is authorized to view the details of the account they have requested information for.

Low Severity Findings

No low severity findings

Informative Severity Findings

> No informative severity findings

Strike

Appendix A

Appendix A

Conclusions

[Test Client] suffers a series of security control failures, which lead to compromise the confidentiality and integrity of the application and its related systems. These failures would have had a critical effect on [Test Client] operations if a malicious party had exploited them.

Current security controls are not adequate to mitigate the impact of the discovered vulnerabilities.

The specific goals of the penetration test were stated as:

- Identifying if a remote attacker could penetrate [Test Client]'s defenses.
- Determining the impact of a security breach on:
  - Confidentiality of the company's information.
  - Internal infrastructure and availability of [Test Client]'s information systems.

These goals of the penetration test were met. A targeted attack against [Test Client] can result in a compromise of organizational assets.

It is important to note that this breach in the [Test Client] security infrastructure can be greatly attributed to the lack of input sanitization.

Appropriate efforts should be undertaken to introduce effective security controls to mitigate the effect of identified vulnerabilities.

Strike

Appendix B

Appendix B

Assessment Updates

Assessment Updates are updates that the strikers made to reflect the work they did. These updates are regularly posted by strikers, and are now sorted by the date in which they were posted.

Update #1

These goals of the penetration test were met. A targeted attack against [Test Client] can result in a compromise of organizational assets.

Date: 5 Jul 2022

Description

Task Executed (Black Box):

- RECON (Fuzzing for content + Endpoints, JS files analysis, Technology Enumeration, Port scanning, etc.)
- OSINT (GitHub Leaks, Google Dorks, archived URLs, users enumeration, etc.)
- Web App Business Logic analysis (Test cases, Test users, verification emails, traffic analysis, etc.)
- Source code analysis.
- XSS Tests.
- #1 Critical Vulnerability discovered - SQL Injection.

Regards,
Javier Bernardo

Update #1

Date: 7 Jul 2022

Description

Following initial test scenarios have been covered till now:

- Checked `https://testapp.testclient.com/` and `https://testapi.testclient.com/` for TLS configuration issues. The domains now implement the latest version of TLS with strong cipher suites.

- Performed directory fuzzing and file enumeration for anonymous and sensitive information disclosure.
- Performed Shodan analysis to identify any sensitive information exposed or to gather more information about the domains.
- Completed a website crawl and mapped the pages. Checked JS libraries and software components for missing patches.
- Checked the initial health information filling form for injection flaws like SQL Injection and Cross-site scripting.
- #1 High Vulnerability discovered - Stored XSS.
- #1 Medium Vulnerability discovered - Horizontal Privilege Escalation.
- Attempted captcha bypass by removing token and reusing token value. This was prevented by server-side logic.

Regards,
Santiago Rosenblatt

Strike

Appendix C

Appendix C

Compliance Checklist

Authentication

Password Security Requirements

#	Title	
1.1.1	Verify that user set passwords are at least 12 characters ins length.	☑
1.1.2	Verify that passwords 64 characters or longer are permitted.	☑
1.1.3	Verify that passwords can contain spaces and truncation is not perfomed. Consecutive multiple spaces may optionally be coalesced.	☑
1.1.4	Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 12 emoji or 64 kanji characters should be valid and permitted.	☐
1.1.5	Verify users can change their password.	☑
1.1.6	Verify that password change functionality requires the user's current and new password.	☐
1.1.7	Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new nonbreached password.	☐

General Authenticator Requirements

#	Title	
1.2.1	Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.	☑
1.2.2	Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.	☑
1.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	☑

Session Management

Fundamental Session Management Requirements

#	Title	
2.1.1	Verify the application never reveals session tokens in URL parameters or error messages.	☑

Session Binding Requirements

#	Title	
2.2.1	Verify the application generates a new session token on user authentication.	☑
2.2.2	Verify that session tokens possess at least 64 bits of entropy.	☑
2.2.3	Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.	☑