**Final Latek(overleaf) code ns project**

\documentclass{report}

\usepackage{graphicx}

\title{Network Security Project}

\reportstage{1}

\degree{Bachelor In Computer Science}

\degreespecialization{Network Security}

\department{Department of Computer Science}

\academicyear{2021}{2025}

\hod{Prof Faisal Iradat}

\author{Shahmeer Khan (25156)}

\coauthor{Nosherwan Sheikh (25237)}

\coauthor{Tabish Imran (17905)}

\declarationdate{Friday}{17\textsuperscript{th} November, 2023}

\declarationplace{Karachi, Sindh}

\graphicspath{ {images/} }

\begin{document}

\maketitle{A03:2021 – Injection}

\begin{dedication}

To my University..

\end{dedication}

\makecertificate

\makedeclaration

\begin{abstract}

This report specifies the various processes and techniques used in gathering requirements, implementing and testing for the project on ‘A03:2021 – Injection’ which deals with one of the vulnerabilities from OWASP top 10 ie Injection. This project aims to use BURP suite to test the vulnerabilities and display the results of various tests.

\end{abstract}

\tableofcontents

\newpage

\section{Introduction}

We will focus on the Injection Vulnerability by performing BURP SUITE tests on possible attacks and flaws, test result findings will be displayed accordingly, the aim is to further innovate and discover in the field of network security and also emphasize the benefit of BURP SUITE for network testing for candidates that have just gathered an interest in the field.

\newpage

\section{SQL Injections and the threat they propose}

\newline

One significant and frequent kind of web application security flaw is SQL injection. Attackers can manipulate input to insert malicious SQL code when an application fails to properly validate or sanitise user input before utilising it in SQL queries. This may result in data tampering, unauthorised database access, and in certain situations, total system compromise.

\newline

\newline

\textbf{SQL injection's risks include:}

\newline

\newline

\textbf{Unauthorised Entry:}

\newline

SQL injection is a tool that attackers can use to get around authentication restrictions and access private information without authorization. Passwords, usernames, and other private information may fall under this category.

\newline

\newline

\textbf{Data manipulation:}

\newline

Data loss or corruption can result from the use of SQL injection to add, remove, or modify data in databases. This is especially worrisome when handling sensitive or important data.

\newline

\newline

\textbf{Code Execution:}

\newline

SQL injection may occasionally be used to run arbitrary code on the server, which could result in the host system being fully compromised.

\newpage

\section{Why are these issues overlooked?}

There are a number of technological and organisational reasons why security flaws, such as SQL injection vulnerabilities, are often ignored. These are some explanations for why these problems might go unnoticed:

\newline

\newline

\textbf{Budget limit: }

\newline

Security measures often require additional resources in terms of both time and money. Organizations with limited resources may use resources to meet functional requirements, ignoring security considerations.

\newline

\newline

\textbf{False beliefs regarding frameworks:}

\newline

Security is not guaranteed if a specific programming language or framework is all that is used. Developers may fail to implement security properly because they believe that utilising a well-known framework renders their application automatically secure.

\newline

\newline

\textbf{Lack of Security Education:}

\newline

It's possible that many developers don't get enough instruction or training on safe coding techniques. Developers run the risk of unintentionally introducing vulnerabilities if they lack a thorough awareness of potential security risks.

\newline

\newpage

\section{Why choose Injections}

SQL Injection is a malicious technique and poses serious security threats however it is not given the emphasis it deserves due to which attackers often exploit its effectiveness. Additionally, SQL Injection attacks can be particularly lethal because they exploit weaknesses in authentication mechanisms, for instance, by injecting a payload like '101 OR 1=1' into a login form, an attacker can manipulate the SQL query to always evaluate to true, effectively bypassing authentication checks and not needing a password, with this the attacker has access to all your protected data and with the fact that data is now more valuable than money itself, the damage can be critical. Another reason we chose SQL injections in particular is that we currently have a database course and we are working on a web app connecting to a database. By learning more about SQL injections we can find ways of securing not only our own , but databases all over the world, we hope to bring light to the damage that can be done by this and thus highlight the importance of security against it.

\section{Why choose Burp Suite}

Burp Suite is an ideal tool for testing SQL Injection vulnerabilities due to its features and user-friendly interface. It is particularly effective in identifying and exploiting SQL Injection weaknesses in web applications. Its scanning capabilities make it an excellent choice for testing vulnerabilities like SQL Injection. Its the ideal choice for beginners at network security and furthermore PortSwigger and Youtube provide several helpful tutorials for testing therefore leading to a better understanding of the topic at hand.

\newpage

\section{Work Breakdown/Methodology}

1.Install Burp Suite and configure proxy settings.

\newline

2.Explore the OWASP web application to identify potential injection points.

\newline

3.Perform manual testing using Burp and show a step by step process on how to achieve that

\newline

4.Document findings and prepare a detailed report.

\newline

5.Determine the severity of the injection attack based on the findings.

\newline

6.Generate use cases that indicate the possibilities these attacks could bring about.

\section{Inclusions and Exclusions}

\textbf{Inclusions}

\newline

.Manual testing of user input fields.

\newline

.Step by step guide for a beginner to perform such tests

\newline

.Use of various SQL injection techniques

\newline

.Documentation of successful and unsuccessful attempts.

\newline

.Reporting on identified vulnerabilities.

\newline

.Generating use cases and ranking severity.

\newline

\newline

\textbf{Exclusions}

\newline

.Full-scale penetration testing beyond SQL injection.

\newline

.Vulnerabilities other than SQL Injections.

\newline

.Testing on websites other than OWASP web app.

\section{Functional Requirements(hw/sw)}

.Burp Suite installation and proper configuration.

\newline

.OWASP web application being functional for testing.

\newline

.Internet connectivity for resources,testing and documentation.

\newline

.Sufficient computing resources for efficient testing.

\newline

\newline

\textbf{Hardware Requirements}

.Computer with enough memory and processing ability for efficient testing

\newline

\newline

\textbf{Software Requirements}

.Burp Suite Community version , could switch to professional version if needed

\newline

.Web browser configured to use Burp as a proxy

\newline

.Access to OWASP web app for testing

\newpage

\section{Vulnerability Data Analysis Methodology}

.Categorize vulnerabilities based on severity(1-10 scale)

\newline

.Verify and validate each identified vulnerability.

\newline

.Provide images and results for every test run

\newline

.Provide use cases to identify what the vulnerability could lead to

\newline

\section{FrameWork}

\newline

\newline

\textbf{OAWSP top ten (SQL Injection):}

\newline

Role: Use OWASP top ten projects to check for SQL Injection and related CWE's

\newline

Incorporation: Integrate the recommended tests to conduct

\newline

\newline

\textbf{OWASP web security Testing Guide (WSTG):}

\newline

\newline

\textbf{Portswigger's guide for testing vulnerabilities via BURP Suite:}

\newline

Role: Use Portswigger's guide to effectively use BURP Suite for vulnerability testing.

\newline

Incorporation: Follow Portswigger's guide to align BURP Suite testing with best practices and techniques.

\newpage

\section{Methodology}

\newline

\newline

\textbf{Defining Objectives:}

\newline

Objective: Clearly define the goal of the project.

\newline

Activities: Understand the overall security objectives of the OWASP web app with regards to SQL Injection

\newline

Define specific goals for SQL injection testing.

\newline

\newline

\textbf{Scope Definition:}

\newline

Objective: Clearly define the scope of the testing.

\newline

Activities: Identify the specific functionalities and areas within the OWASP app to be tested for SQL injection vulnerabilities.

\newline

Set boundaries for the testing scope.

\newline

\newline

\textbf{Resource Allocation:}

\newline

Objective: Allocate necessary resources for testing.

\newline

Activities: Ensure the availability of tools for eg including Burp Suite, for the testing process.

\newline

\newline

\textbf{Test Planning:}

\newline

Objective: Develop a comprehensive plan for SQL injection testing.

\newline

Activities: Create a test plan outlining the testing approach and tools.

\newline

Define roles and responsibilities for testing team members.

\newline

\newline

\textbf{Execution:}

\newline

Objective: Execute the defined testing plan.

\newline

Activities: Perform information gathering on the OWASP app using Burp Suite.

\newline

Conduct threat modeling to identify potential SQL injection points.

\newline

Develop and execute SQL injection test cases.

\newline

\newline

\textbf{Analysis and Validation:}

\newline

Objective: Analyze test results and validate findings.

\newline

Activities: Analyze error messages and responses for indications of SQL injection vulnerabilities.

\newline

Validate identified vulnerabilities to ensure they are not false positives.

\newline

\newline

\textbf{Reporting:}

\newline

Objective: Document and communicate the testing results.

\newline

Activities: Generate a comprehensive report detailing identified SQL injection vulnerabilities, their severity and categorization.

\newline

Provide evidence and documentation to support findings.

\newpage

\section{Process}

\newline

\newline

\textbf{Information Gathering:}

\newline

Get familiar with the OWASP web app and understand its architecture, endpoints, and user inputs.

\newline

\newline

\textbf{Threat Modeling:}

\newline

Identify potential SQL injection points by analyzing user inputs, parameters, and data flow.

\newline

\newline

\textbf{Burp Suite Configuration:}

\newline

Configure Burp Suite on our system to intercept and analyze traffic.

\newline

\newline

\textbf{Test Case Development:}

\newline

Develop SQL injection test cases covering various techniques and contexts.

\newline

Create realistic use cases involving SQL queries to simulate user interactions.

\newline

\newline

\textbf{Injection Testing:}

\newline

Use Burp Suite's tools (Intruder, Repeater) for manual injection testing.

\newline

\newline

\textbf{Error Handling Analysis:}

\newline

Investigate error messages returned during injection attempts for clues about vulnerable points.

\newline

\newline

\textbf{Authentication Bypass Testing:}

\newline

Verify if SQL injection can lead to unauthorized access by bypassing authentication mechanisms.

\newline

\newline

\textbf{Data Extraction Testing:}

\newline

Check if it's possible to extract sensitive information from the database using SQL injection.

\newline

\newline

\textbf{Logging and Reporting:}

\newline

Log all testing activities, including successful injections, false positives, and issues encountered.

\newline

Generate a detailed report outlining identified vulnerabilities, their impact.

\newline

\newline

\textbf{Post-Assessment:}

\newline

Conduct a post-assessment to categorize the vulnerabilities based on severity , commonness and other variables.

\newpage

\section{Configuring Burp and OWASP Juice Shop}

\newline

\newline

\newline

First install juice shop via the command "git clone https://github.com/juice-shop/juice-shop.git --depth 1"

\newline

\newline

Then run the command "cd juice-shop"

\newline

\newline

To start Juice shop, for the first time run “npm-install” on command line, and then “npm-run”.

\begin{center}

\includegraphics[width=10cm]{command line prompt.png}

\end{center}

\newline

\newline

To access owasp juice shop, search for “localhost:3000” on your browser.

\begin{center}

\includegraphics[width=10cm]{owasp search.png}

\end{center}

\newline

\newline

\newpage

\textbf{There are two ways to intercept traffic via burp suite, the previous way involved burp configuration with your browser that included several steps i.e:}

\newline

\newline

-Assuming our browser is FireFox, add FoxyProxy extension and configure a burp suite proxy accordingly as such, this can be found in options -> proxies

\begin{center}

\includegraphics[width=10cm]{FoxyProxy integration.png}

\end{center}

\newline

\newline

\newline

-The hostname and port for this can be found directly through burp itself, this can be found in proxy settings in burp

\begin{center}

\includegraphics[width=10cm]{burp hostname.png}

\end{center}

\newline

\newline

-Using foxyproxy w can easily turn the proxy on and off when needed

\begin{center}

\includegraphics[width=6cm]{foxy toggle.png}

\end{center}

\newline

\newline

\newpage

-Alternatively, you can add the proxy manually into the browser however this doesn’t allow the freedom to turn the proxy on and off with a click as you have to edit the settings every time.

\begin{center}

\includegraphics[width=10cm]{burp manual config.png}

\end{center}

\newline

\newline

-You then also need to add the certificate for burp to allow the proxy’s validation, the certificate can be found by enabling the proxy and searching for http://burp that will lead you to this screen, download the certificate by clicking on CA certificate

\begin{center}

\includegraphics[width=10cm]{burp cert.png}

\end{center}

\newline

\newline

\begin{center}

\includegraphics[width=10cm]{certificate add.png}

\end{center}

\newline

\newline

The second way is much simpler as it doesn’t involve all of this integration, here you simply click open browser which opens google chromium which is already configured with burp, by turning interceptor on you can start intercepting queries on burp

\begin{center}

\includegraphics[width=10cm]{open interceptor browser.png}

\end{center}

\newline

\newline

\newpage

\section{Tests}

\newline

\newline

\newline

\newline

\Large\textbf{Login bypass SQL Injection}}

\newline

\newline

\large\textbf{The first thing we’ll be testing is login bypass using sql injections, here instead of username and password we inject sql payloads that allow us access without the username and password:}

\newline

\newline

-To start off, first open juice shop in your browser and click on login, from here turn interceptor on as shown above, this is assuming configuration has been done already

\newline

\newline

-The second step is to add random values in both username and password for testing, here I use the value “test” as username and password

\newline

\newline

\begin{center}

\includegraphics[width=8cm]{test login.png}

\end{center}

\newline

\newline

\newpage

-When you click login, go to burp where you will see requests in the proxy tab as, click forward until you see a request as such, as you can see below , username and password are shown

\begin{center}

\includegraphics[width=12cm]{test intercept.png}

\end{center}

\newline

\newline

-Right click anywhere on the screen while on this page and choose send to intruder as such

\begin{center}

\includegraphics[width=12cm]{burp intruder send.png}

\end{center}

\newline

\newline

\newpage

-From here go to intruder tab and click on the positions tab, this signifies the regions we want our payloads to be in, at first click “Auto”, this selects all possible payload locations, however we only wish to alter the username and password, so after clicking “Auto”, highlight the unneeded highlighted sections i.e. highlight the cookie and click “Clear”, this allows you to select only the payloads for username and password.

\begin{center}

\includegraphics[width=12cm]{positions.png}

\end{center}

\newline

\newline

-After doing this head on to the payloads tab, leave everything else unchanged, in the payloads section head down to the bottom to find the encoding section, this is often checked by default, we need this unchecked for our attacks as it encodes the payloads rendering them unusable.

\begin{center}

\includegraphics[width=12cm]{paylod encoding.png}

\end{center}

\newline

\newline

\newpage

-Next in payload settings, load the txt file containing the sample payloads for testing, these are provided by default in the paid version , you need to add these in manually for the community version, we used this sql payload file for our testing, https://github.com/xmendez/wfuzz/blob/master/wordlist/Injections/SQL.txt

\newline

\newline

-Click on load and select your payload file, or enter every payload individually by typing them in and clicking add

\begin{center}

\includegraphics[width=11cm]{paylist.png}

\end{center}

\newline

\newline

-After adding the payload click on start attack in the top right corner in the intruder tab

\begin{center}

\includegraphics[width=4cm]{start.png}

\end{center}

\newpage

-The attack can be paused by clicking on attack in the top left and clicking pause, otherwise let the attack continue

\newline

\newline

-What we need to look out for is a status value of 200, this indicates a successful attack

\begin{center}

\includegraphics[width=10cm]{stat 200.png}

\end{center}

\newline

\newline

-To get details for a particular payload double click it, a window pops up showing your request and the response to it.

\begin{center}

\includegraphics[width=10cm]{resonse wind.png}

\end{center}

\newline

\newline

\newpage

-As we can see below, the response for the payload "hi’ or 1=1 --"shows us an authentication token and the umail which stands for user email which we can use to login later on.

\begin{center}

\includegraphics[width=11cm]{respsp.png}

\end{center}

\newline

\newline

-We then use the payload itself to login i.e. in this case "hi’ or 1=1 --", the password can be anything as long as we use this in the username.

\newline

\newline

-Using this to login brings us to the screen where we were even greeted with an achievement, these are builtin for owasp juice shop which signify progress.

\begin{center}

\includegraphics[width=11cm]{2nd achievement.png}

\end{center}

\newline

\newline

\newpage

-However, for our own ease its better to get the actual username and password, for this we use the authentication token, we copy this into the decoder tab in burp, choose decode in to be base64 and paste your token here and choose text instead of hex in both sections on the right, upon clicking start decode we can see the username and password in the lower half

\begin{center}

\includegraphics[width=11cm]{decoder result.png}

\end{center}

\newline

\newline

-While the username is straightforward the password is hashed, to get the original password we can use https://crackstation.net/, copy and paste your hashed password here to see what the original password was in result which in this case is “admin123”

\begin{center}

\includegraphics[width=11cm]{crack1.png}

\end{center}

\newline

\newline

\newpage

-To confirm this, we can now use our discovered credentials to login

\begin{center}

\includegraphics[width=11cm]{testing recieved credentials.png}

\end{center}

\newline

\newline

-Upon logging in we are greeted with a new achievement as we logged in without an sql injection this time

\begin{center}

\includegraphics[width=11cm]{newach.png}

\end{center}

\newline

\newline

\newpage

-We can also see the results on burp which shows us the correct credentials intercepted as we entered them

\begin{center}

\includegraphics[width=11cm]{corr inter.png}

\end{center}

\newline

\newline

-The other status codes one can encounter are 401 and 500

\newline

\newline

-401 is a failed attempt as shown below

\begin{center}

\includegraphics[width=11cm]{401 status.png}

\end{center}

\newline

\newline

\newpage

-500 is an internal system error which does give a response and even other values like a password hash

\begin{center}

\includegraphics[width=11cm]{500 status.png}

\end{center}

\newline

\newline

-To investigate this further we used crackstation again to unhash this password to see if this attack was successful, the decryption gave us “test” as the result meaning this is also a failed attempt as we gained nothing of use.

\begin{center}

\includegraphics[width=11cm]{crack2.png}

\end{center}

\newline

\newline

\newpage

-A few payloads returned errors which looked like this

\begin{center}

\includegraphics[width=10cm]{erroror.png}

\end{center}

\newline

\newline

You may be wondering why we only investigated one successful attempt i.e. one payload only, the reason for that is that in all the successful attempts the resulting username was always admin@juice-sh.op and the password returned was the same as well, this doesn’t however render them useless as it shows multiple ways of constructing an sql injection attack, shown below is the payload for "‘ or 1=1 --" which gives similar results

\begin{center}

\includegraphics[width=10cm]{other.png}

\end{center}

\newline

\newline

\newpage

-We can also note our progress through logging

\begin{center}

\includegraphics[width=11cm]{log.png}

\end{center}

\newline

\newline

\newpage

\Large\textbf{SQL Injection Union Attack}}

\newline

\newline

\large\textbf{In a Union Attack, the attacker uses the UNION SQL operator to combine the results of the original query with those of another query, often to extract unauthorized information from the database. This is achieved by injecting additional SQL code into input fields, exploiting vulnerabilities in poorly sanitized user inputs.}

\newline

\newline

In this particular project, the target is to send user requests for product delivery to the locally hosted OWASP juice-shop server with malicious payload and check the responses to test success/failure.

\newline

\newline

Payload Link: https://github.com/payloadbox/sql-injection-payload-list/blob/master/Intruder/detect/Generic\_UnionSelect.txt

\newline

\begin{center}

\includegraphics[width=8cm]{nosh1.png}

\end{center}

\newline

\newline

\newpage

\raggedright

Using similar setup as the first attack, the second attack is launched. The payload is set to generate close to 3000 requests, each of which is set to produce a response which may or may not contain some entries from the server’s database.

\begin{center}

\includegraphics[width=11cm]{nosh2.png}

\end{center}

\newline

\newline

After launching the attack, a window pops up showing a list of the results produced by injection of each statement.

\begin{center}

\includegraphics[width=11cm]{nosh3.png}

\end{center}

\newline

\newline

\newpage

We sort this list by status code to easily navigate to the results with response codes 200/201 indicating success/resource created and ignoring the results with any other status codes.

\newline

\newline

Further we click each response to see what exactly was sent by the server.

\begin{center}

\includegraphics[width=12cm]{nosh4.png}

\end{center}

\newline

\newline

Upon inspection of one of the responses with status code 201, the above response was generated. This response contains only the confirmation of a new entry being created on the server’s database. In an ideal case for the attacker, the response may contain a list of username/password pairs which will give the attacker the ability to log into any user account.

\newline

\newline

Due to the limitations of burp suite community version, it takes a significant amount of time to generate all of the responses. Also, all of these responses need to be manually checked for any information useful to the attacker.

\newpage

\section{Use Cases}

\newline

\newline

\textbf{SQL login bypass}

\newline

\newline

\textbf{Severity rating: 10}

\newline

\newline

The reason this gets such a high severity rating is the risks such unauthorized access could bring to a website as this is what an attacker could use such a bypass for:

\newline

\newline

\textbf{Data Breach:}

\newline

-Steal sensitive user information, such as personal details, passwords, and financial data.

\newline

-Access confidential business data, intellectual property, or trade secrets.

\newline

\newline

\textbf{Financial Loss:}

\newline

-Manipulate or steal financial transactions.

\newline

-Redirect funds or alter payment details.

\newline

\newline

\textbf{Identity Theft:}

\newline

Impersonate users or administrators to carry out fraudulent activities.

\newline

Create fake accounts or profiles using stolen credentials.

\newline

\newline

\textbf{Website Defacement:}

\newline

Modify the website's appearance or content to spread false information or propaganda.

\newline

Display offensive or harmful content.

\newline

\newline

\textbf{Denial of Service (DoS) Attacks:}

\newline

Disrupt website functionality by overloading servers or services.

\newline

Render the website inaccessible to legitimate users.

\newline

\newline

\textbf{Distributed Denial of Service (DDoS) Attacks:}

\newline

Coordinate a large-scale attack using multiple compromised systems to overwhelm the website's infrastructure.

\newpage

\textbf{Ransomware:}

\newline

Encrypt website data and demand a ransom for its release.

\newline

Threaten to expose sensitive information unless payment is made.

\newline

\newline

\textbf{Malware Injection:}

\newline

Introduce malicious code into the website to infect visitors or compromise their devices.

\newline

Use the website as a platform to distribute malware to a wider audience.

\newline

\newline

\textbf{Sabotage or Destruction:}

\newline

Delete critical data or entire databases.

\newline

Intentionally damage the website's functionality or reputation.

\newline

\newline

\textbf{Backdoor Installation:}

\newline

Establish a secret entry point (backdoor) for persistent access.

\newline

Maintain control even after the initial breach has been detected and resolved.

\newline

\newline

\textbf{Legal Consequences:}

\newline

Engage in activities that could lead to legal action against the website owner or administrators.

\newline

Use the compromised website for illicit purposes, making the legitimate owners liable.

\newline

\newline

\newpage

\textbf{SQL Union Attack}

\newline

\newline

\textbf{Severity rating: 10}

\newline

\newline

The reason this gets such a high severity rating is the again the major risk of unauthorized access as well as other consequences as discussed below:

\newline

\newline

\textbf{Extracting Data:}

\newline

An attacker may use a union attack to combine results from different database tables, extracting sensitive information like usernames, passwords, or other confidential data.

\newline

\newline

\textbf{Identifying Database Structure:}

\newline

By manipulating the UNION statement, an attacker can gather information about the database structure, such as table names and column names, which helps in planning further attacks.

\newline

\newline

\textbf{Authentication Bypass:}

\newline

If a web application uses SQL queries for authentication, an attacker might attempt a union attack to bypass login mechanisms and gain unauthorized access.

\newline

\newline

\textbf{Data Tampering: }

\newline

Injection attacks can be used to modify or delete data in the database, impacting the integrity of the information stored.

\newline

\newline

\textbf{Error-based Attacks:}

\newline

Error-based Attacks: Union attacks can exploit error messages generated by the database system to reveal information about the structure of the query, helping the attacker refine their injection technique.

\newpage

\section{Conclusion}

In conclusion, the exploration of SQL injection vulnerabilities using Burp Suite has provided valuable insights into the potential risks associated with insecure databases. The lack of attention paid to such risks could evidently prove fatal to the success of many organizations.

\newline

\newline

Through the step-by-step guide and practical testing scenarios outlined in this report, we have demonstrated the effectiveness of Burp Suite in identifying and assessing SQL injection vulnerabilities while also allowing the understanding of both how to use burp suite to perform such tests as well as understand what these vulnerabilities could cause.

\newline

\newline

The generated use cases visualize the diverse range of malicious activities that can be carried out through successful SQL injection attacks. From unauthorized data access to manipulation of sensitive information and even potential remote code execution, the report describes the importance of mitigating SQL injection vulnerabilities promptly.

\newline

\newline

All in all this report serves as a reminder as to why SQL injections need to be taken seriously in the field of network security.

\include{chapters/chapter1}

\include{chapters/chapter2}

\include{chapters/chapter3}

\include{chapters/chapter4}

\include{chapters/chapter5}

\include{chapters/chapter6}

\bibliography{plain}{chapters/bibliography}

\makeacknowledgement

\end{document}