Final Report

Group 7

Daniel Giulitto: Lead: Webmaster

Parth Mahida: Team Leader

Tony An: Lead Video Presentation

Mustaf Khire: Lead Report

BTN 710 NAA

Table of Contents

Introduction

The focus of this research project is to take a website that is non-malicious in nature, yet contains many vulnerabilities and show how that site can be easily exploited by malicious attackers. We have developed a basic, faulty system that we will then proceed to attack by targeting vulnerabilities. We will then be attempting to fix these vulnerabilities in order to increase user security in the future.

The site in question being utilized for this project is MjCatering, a site under development running on a local host. While not fully complete, the site does show the general structure and functionality of a basic food catering site and contains functionality such as creating and storing user accounts, adding and removing food items to a cart, and a checkout process where card details are processed for payment. User Accounts and information on the site are stored on a Mongo DB database where a basic CRUD API is utilized to interact with user accounts and data. This makes the stored data a prime example to utilize in a demonstration of what can go wrong when routes and data are not properly secured and is easily accessible to attackers.

This report will go over the process of deciding how to attack the site and the outcome of these attacks. The vulnerabilities of the site will first be examined in order to state what parts of the site are open to attack and why. The system setup that will be utilized in order to attack the site will discuss what hardware and systems the site will be running on when it will be attacked. We will also detail what hardware and systems the attack itself will be conducted on while the site is running, and a description of where in the network where the incident will occur. Then the exploits themselves will be discussed including how the exploit works, and an in depth look at how the attack was carried out. Finally possible improvements to the site will be suggested and a look at how the site can be improved for greater user security. Our findings will then be consolidated into a conclusion where we give an overall evaluation of our results. There are further forms on the website that could also be weak to SQL Injection attacks such as

Section 1 - Vulnerability

Due to the more basic nature of the developed site, the exploitability of the site is rather high. Most exploits that can happen to the site can be conducted with minimal equipment. The relatively minor detection software also shows that the site will not be able to detect threats very well, and the damage of these attacks can be high if things like financial information are targeted.

One of the immediately noticeable vulnerabilities on the site is the login page upon accessing the site. The login page is an input field that sends queries back to the database utilizing SQL requests. Because of this the login page can be weak to SQL injection as the user input is not sanitized before being sent into an SQL request. A malicious attacker would be able to input special SQL syntax within one of the fields, allowing them to tamper with the intended SQL statement that would be generated upon completion of the form. This could allow an attacker to pull more usernames and data from the database when they should not be able to. This is made even easier by the fact that usernames and passwords are not hashed within the Mongo database, making it even easier for an attacker to utilize a malicious SQL request to get user information from the database.

Because of the way that the site uses URL pathing in order to fetch this data from the API the site is also at risk to Insecure Direct Object References (IROD) attacks. For example, the site accesses users from the database with the base API route of “http://localhost:3000/api/”. If an attacker wished to exploit this to their advantage, they could attempt to get specific users by adding onto that path, or changing parameters. This makes the site weak to any attacks that would exploit the URL pathing to gain further data they are not authorized for out of the database.

Another one of the vulnerabilities present in the site has to do with the storage of a current user session. If a user is logged in to the site and they close a tab without logging out, they can return to the site with their session still active. While maintaining sessions for a set time is a common occurrence on sites, there are usually further security measures taken when checking out a transaction and financial information is involved. Unfortunately, on this site no further measures are taken, meaning that if a user were to close a tab and leave their computer in a public place, like a coffee shop for example, their information can be easily accessed by an attacker. They would simply need to leave their computer unattended for a moment and an attacker would easily be able to access their full account and financial information, which is a very large security flaw.

Because of the way this user session is managed, this also makes users susceptible to Cross Site Request Forgery. Due to the site having active sessions even after closing tabs, this can make users prone to phishing attacks through their emails. Malicious links could send users to change their password or make incorrect purchases as long as their session is still being maintained.

These different vulnerabilities make the site faulty and a large security risk for users. While the site is not malicious in nature, there are many security oversights that have not been considered by the developers, that if not fixed can result in massive damage by attacks to the site and its users.

Section 2 - System Setup

We do not have any hardware to demonstrate our attack. The main operating systems that we are going to use are laptop, desktop to demonstrate our attack. The protocol that we are going to use is our website to present the attack. Our website is basically a catering system which allows users to order food online. This website is built on Visual Studio code using React Framework, it runs locally on laptop/desktop. We built an API for this website using Node JS. The API does the CRUD operations. We tested all the CRUD operations on postman. The Database portion is stored in MongoDB. The main functionality of the database is to store the user credentials, it also stores the food items that users will be ordering. As for the payment section we also built an API it is called anxious. It would handle the payment transactions. It will validate all the transactions. The attacks will be committed from the same machine due to the local host nature of the site. Even though the site has been hosted locally that in live build attack would happen over the transmission from the user back to the hosting machine where the database is located.

Section 3 - Exploit

In the following section our team will go in depth on how the following exploits will be used to attack our web application. Our web application will implement the types of attacks that will be sent in the form of scripts that will be able to take advantage of our non-malicious web application with a faulty design. For our first attack we will implement an SQL Injection, which is essentially where an attacker would prioritize on first finding the user inputs that are vulnerable within the web application. This is very important for the attacker to find this because a web application that has an SQL Injection vulnerability, the attacker would have access to the user input directly in an SQL query in order to create input content. The attacker that inputs the content is injecting a content that is known to be a malicious payload and is the essential part of the attack. The following content that was sent by the attacker, malicious SQL commands are subsequently executed in the SQL database. The purpose behind SQL is that the language is designed to manage data stored into logical databases, and with that purpose one is able to access, modify, and delete data. Since many web applications use SQL databases to store all their data, SQL commands are able to run operating system commands such as on Windows and Linux systems. Thus, a successful injection can lead to very serious consequences.

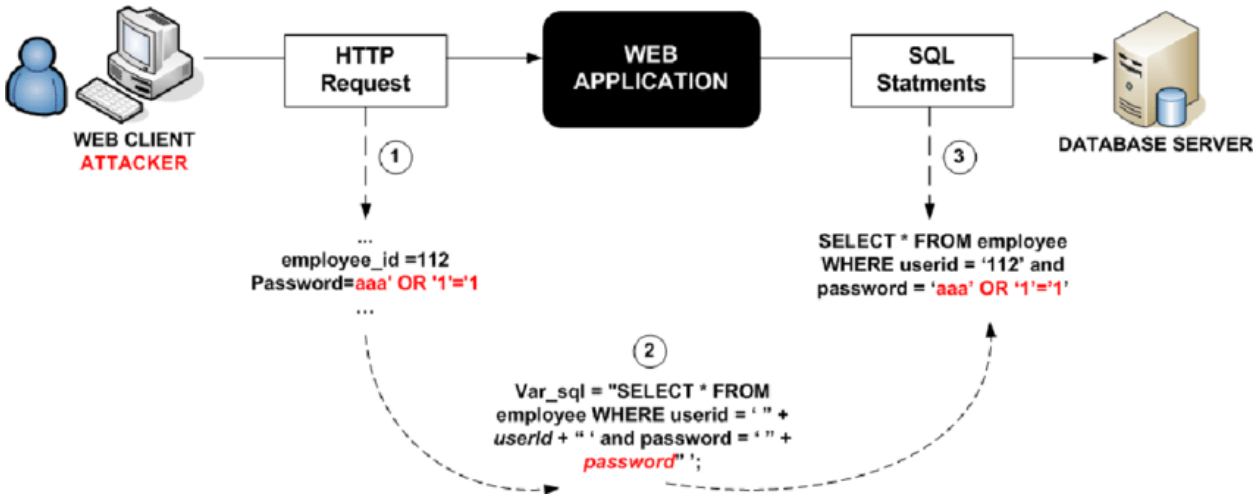

Figure 1 - SQL source code example

In addition to the attacks, we wish to implement, our next attack will focus on Insecure Direct Object References (IDOR) attack. The following attack essentially occurs when an attacker uses an identifier for direct access to an object in which that is internally implemented but only provides no additional authorization checks and access control. With that general notion of what an IDOR attack is, how we would implement such an attack is by obtaining all location in which user input was used to reference objects directly, for instance such location would be found where the user input is used to access the row of a database, the page of the website, a certain file and many others. The attacker would then modify the value of the parameter in which the reference object was used and check whether it is possible to retrieve said objects that belong to other users and even whether it can bypass authorization.

Source code:

Say for instance this is the URL used to retrieve a database record with its corresponding value in the parameter.

https://www.example.com/transaction.php?id=74656

The attacker can substitute the id value with a similar value in the place of the current value.

https://www.example.com/transaction.php?id=74657

The following id= 74657 can possibly be a real transaction that belongs to a user, but if the developer were to make an error in his authentication, the attacker will have access to the user's transaction which would lead to an IDOR attack.

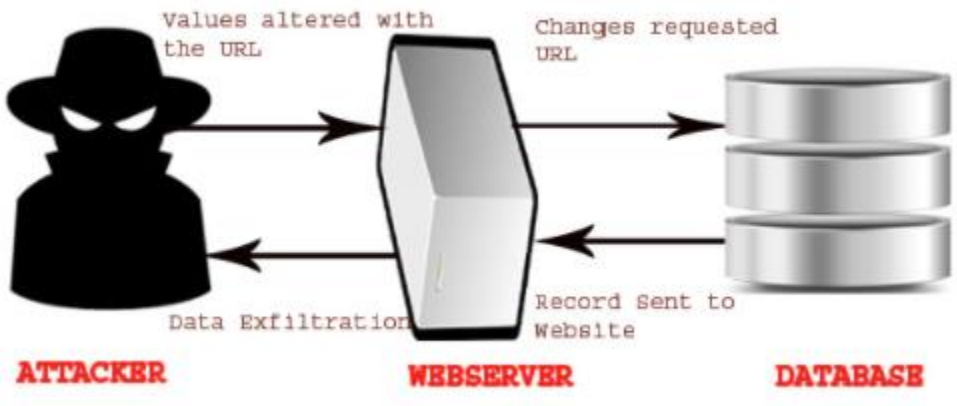

Figure 2 - IDOR Attack demonstration

The next following attack that we wish to implement on our web application is the weak Authentication and Session Management. In such vulnerabilities, the attacker would prioritize on detecting gaps in the user validation and verification. The attacker would then proceed to use automated tools to retrieve vital additional information and obtain a deeper control of the web application. The purpose that makes authentication and session management so important in web applications today is that it is a crucial element in security frameworks and attackers exploit these frameworks if the security implementation is not up to par, which would assist them gaining access.

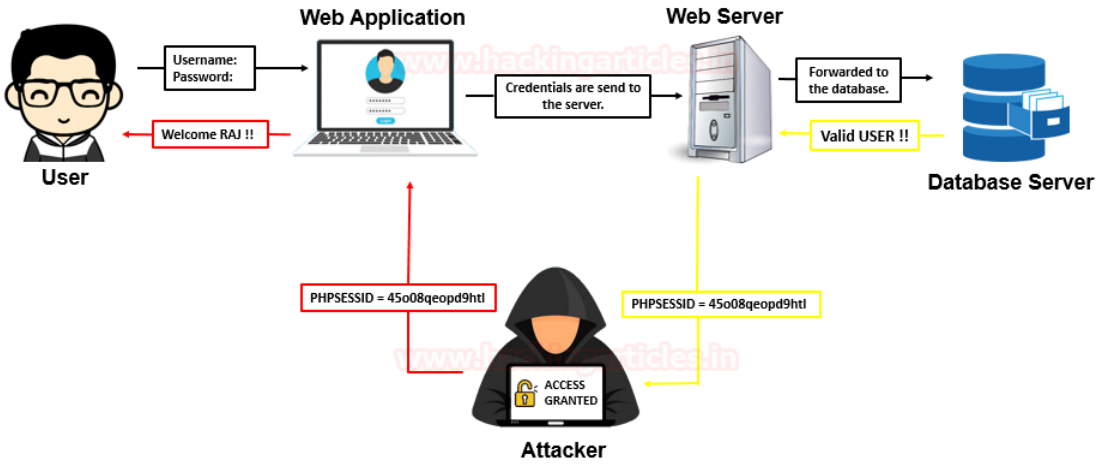

Figure 3 - Weak Authentication and Session Management

Lastly, the final attack that we wish to exploit onto our web application is called Cross Site Request Forgery (CSRF). This attack will essentially trick the user into making a request in which the user did not intend to make. This request in our case could be in the form of an un-authentic link through an email while the user logs in. That same link would be able to send data that the attacker desires to send back to the victim. Therefore, this will end up making the web application to not trust the victim's browser whenever the attacker wishes. For example, the attacker is able to change the user's email address or password, or even use victims' payment credentials to purchase things.

Source code:

Typical GET request for a $100 bank transfer

GET http://netbank.com/transfer.do?acct=PersonB&amount=$100 HTTP/1.1

Attacker can modify the script where the transfer will be made directly to their own account:

GET http://netbank.com/transfer.do?acct=AttackerA&amount=$100 HTTP/1.1

Attackers would then embed the request into a non-suspicious looking hyperlink and distribute it to a large number of bank customers for them to click the link to activate the transfer.

Read more!

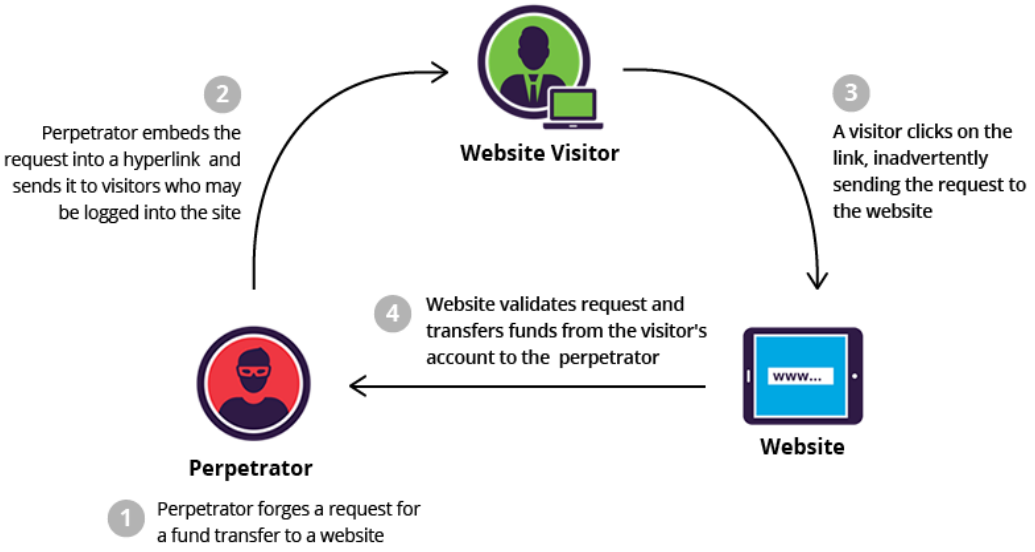

Figure 4 - CSRF diagram of attack

Signature of the attacks:

SQL Injection is unfortunately an attack that is difficult to detect, this is because the attack itself are vulnerabilities that don't leave any traces on any server that it attacks. As for IDOR, this attack can be detected by using a presented request via files being used for detection. These requests are either from previous and future requests by the application. One will be able to detect the requests that an authorized user should be able to see in JavaScript. As for weak Authentication and Session Management, it can be detected by exploiting by using dictionary attacks and automated tools with password lists. Lastly, CSRF is a known attack that never leaves any traces by the attacker since the fake request contains the same information as the same IP address which has the real request from the victim.

Section 4 - Security Policy and Controls

SQL Injection attacks are one of the largest and most common vulnerability attacks on the web.

This type of attack is quite common due to the web developer's laziness and lack of knowledge. To prevent this, we should use input validation and parameterized queries with prepared statements

which makes the parameters as the placeholder will be escaped by DJBC Driver. By adding these two lines of code in MySQL, you can prevent most SQL injections.

```
$stmt = $dbh->prepare("SELECT * FROM users WHERE USERNAME = ? AND PASSWORD = ?");
$stmt->execute(array($username, $password));
```

If it is not a fault from the developer's side, has no access to the software or security details, or it is simply too difficult to change at the time because there are thousands of codes written already, then one can choose to update software they are using. Also, they can secure the database itself more by giving only necessary privileges to the users to do the appropriate tasks. Lastly, using firewalls or vulnerability scanners like Acunetix to notice the weakness of the system.

The best way to prevent Insecure Direct Object Reference attacks is by implementing strict access control checks. This means the system needs to check who gets the access and who does not. It is better to avoid revealing direct object references. This results in using the data which is already there in the user's session on the server to find the resources instead of requesting the references from the actual URL. If exposing the reference to objects in the URL is inevitable, using an indirect reference map can be useful. How it works is that it substitutes the important direct reference in parameter or form with an unpredictable value like GUID to the user. Fortunately, most modern web frameworks like Django have no issues with IDOR. The access control has already been implemented to operate appropriately by default.

To manage sessions properly, we require user authentication credentials to be secured with encryption and hashing. Also, we should not be exposing session IDs in the URL, this can lead to attackers being able to use or access the credentials of currently logged-in users. Therefore, the session IDs should timeout completely after some time or when the users log out. This is obvious, but credentials must not be sent through insecure connections such as credentials, passwords, and session IDs.

Preventing broken authentication will be for both parties' jobs, users and us. We will encourage them to create relatively strong passwords by forcing them to include at least a capital letter, symbol, and number. At the same time, it is also their responsibility to create secure passwords. Making sure that when there is a login failure, the prompt must not specify which part of the login credential failed. Instead, it should say "Wrong password and/or username." This minimizes attackers from narrowing down the vulnerability. Plus, the accounts must be locked down once multiple login failures occur to prevent attackers from brute forcing into those accounts.

Cross Site Request Forgery attacks occur when attackers have knowledge of parameters and value combinations being used in forms. So, including an extra parameter with an unknown value which can be validated by the server will prevent this type of attack. Anti-CSRF token is a server side CSRF protection that is a known string to only the web browser of the user and application. The token is stored in the session variable, so it is hidden and sent when requested. When this value for the session variable and the form field match, the web app accepts the request. Or else, the request gets cancelled. Therefore, since the attacker has no idea which hidden field is needed for the request, the attacker cannot access the token to deploy a CSRF attack.

Lastly, users should have some sort of protections for their own systems such as anti-virus software. That software will automatically detect threats that are harmful for their systems and eliminate them. It is powerful since users or web developers often do not know if they are being attacked or not.

Conclusion

To conclude, MjCatering is our site that is currently under development running on a local host, which has the required functionality to test out attacks that we implemented. Throughout this deliverable, we have explicitly gone through the 4 types of attacks in which we chose would be best for our website. The following attack in which we exploited our site was SQL Injection, Insecure Direct Object References (IDOR) attack, Weak Authentication and Session Management and Cross Site Request Forgery (CSRF) attack. By executing these basic attack scripts, we demonstrate how a non-malicious catering site that is faulty in its security can easily be exploited to put users at risk.

Bibliography

Tajpour, Atefeh. “SQL Injection Attack Detection Using SVM - Researchgate.net.” *Research Gate*, https://www.researchgate.net/profile/Romil-Rawat/publication/258650506_SQL_injection_attack_detection_using_SVM/links/5f0810c892851c52d62697cd/SQL-injection-attack-detection-using-SVM.pdf.

Nidecki, Tomasz Andrzej. “What Are Insecure Direct Object References.” *Acunetix*, 23 Mar. 2020, https://www.acunetix.com/blog/web-security-zone/what-are-insecure-direct-object-references/.

-, Gurubaran S, et al. “A4-Insecure Direct Object References.” *GBHackers On Security*, 12 Nov. 2016, https://gbhackers.com/a4-insecure-direct-object-references/.

Chandel, Raj. “Comprehensive Guide on Broken Authentication & Session Management.” *Hacking Articles*, 10 Apr. 2021, https://www.hackingarticles.in/comprehensive-guide-on-broken-authentication-session-management/.

“What Is CSRF: Cross Site Request Forgery Example: Imperva.” *Learning Center*, 7 July 2020, https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

Morgenroth, Sven. “What Is the SQL Injection Vulnerability & How to Prevent It?” *Netsparker*, Netsparker, 5 May 2020, https://www.netsparker.com/blog/web-security/sql-injection-vulnerability/.

Sven Morgenroth ·, et al. “SQL Injection Vulnerabilities and How to Prevent Them - Dzone Security.” *Dzone.com*, 29 Jan. 2018, https://dzone.com/articles/what-is-the-sql-injection-vulnerability-amp-how-to.

Karande, Chetan. “Securing Node Applications.” *O'Reilly Online Learning*, O'Reilly Media, Inc., https://www.oreilly.com/library/view/securing-node-applications/9781491982426/ch04.html.

“What Is and How to Prevent Broken Authentication and Session Management: OWASP Top 10 (A2).” *Hdiv Security*, https://hdivsecurity.com/owasp-broken-authentication-and-session-management.

Banach, Zbigniew. “Cross-Site Request Forgery Attacks.” *Netsparker*, Netsparker, 27 Aug. 2021, https://www.netsparker.com/blog/web-security/csrf-cross-site-request-forgery/.