**COMPUTER SYSTEMS SECURITY**

|  |  |
| --- | --- |
| TITLE | PERSONAL ASSIGNMENT (PART 1) |
| STUDENT NAME | PEDRO FABIAN OWONO ONDO MANGUE |

**Table of Contents**

[1. Introduction 4](#_Toc190303070)

[2. Security Issues Identification and Analysis in E-Commerce 6](#_Toc190303071)

[Sensitive Data Exposure and Data Breaches 6](#_Toc190303072)

[Broken Authentication and Session Management 7](#_Toc190303073)

[Injection Attacks (SQL Injection) 8](#_Toc190303074)

[Cross-Site Request Forgery (CSRF) Attacks 9](#_Toc190303075)

[Brute Force and Credential Stuffing Attacks 10](#_Toc190303076)

[Security Misconfiguration and Insecure Defaults 11](#_Toc190303077)

[Insufficient Logging and Monitoring 12](#_Toc190303078)

[Weak Authentication and Authorization Mechanisms 13](#_Toc190303079)

[3. Requirements Identification 15](#_Toc190303080)

[Authentication and Authorization 15](#_Toc190303081)

[Data Encryption 16](#_Toc190303082)

[Secure Password Handling 16](#_Toc190303083)

[One-Time Password (OTP) Verification 17](#_Toc190303084)

[CAPTCHA for Bot Protection 18](#_Toc190303085)

[CSRF Protection 18](#_Toc190303086)

[Access Attempt Monitoring and Account Lockout 19](#_Toc190303087)

[4. Conclusion 20](#_Toc190303088)

**List of Figures**

[Figure 1: Web Application System 5](#_Toc190302689)

[Figure 2: Django request and response cycle 6](#_Toc190302690)

[Figure 3: Sensitive Data Exposure Diagram 7](#_Toc190302691)

[Figure 4: Broken Authentication Flow 8](#_Toc190302692)

[Figure 5: SQL Injection Attack 9](#_Toc190302693)

[Figure 6: CSRF Protection Workflow 10](#_Toc190302694)

[Figure 7: Brute Force Protection Mechanism 11](#_Toc190302695)

[Figure 8: Security Misconfiguration 12](#_Toc190302696)

[Figure 9: Logging and Monitoring System 13](#_Toc190302697)

[Figure 10: Weak Authentication and Account Takeover 14](#_Toc190302698)

[Figure 11: Authentication and Authorization 15](#_Toc190302699)

[Figure 12: Data Encryption 16](#_Toc190302700)

[Figure 13: Secure Password Handling 17](#_Toc190302701)

[Figure 14: OTP verification 17](#_Toc190302702)

[Figure 15: Captcha 18](#_Toc190302703)

[Figure 16: CSRF Protection 19](#_Toc190302704)

[Figure 17: Account Lockout 19](#_Toc190302705)

# 

# Introduction

E-Commerce Applications (where you will implement products, users, orders, payments to give easy shopping experience and that is a complex use case) These platforms are built on the multi-tier architecture and have the front end (UI), back end (business logic), middleware, and databases as independent layers. With hundreds of thousands of transactions per second, this architecture allows them to process a lot of transactions while maintaining speed or data accuracy.

Dynamic cloud services and microservice architectures have made e-commerce systems extremely scalable. That means they can independently scale pieces of their platform; say they want to scale up the server to handle payment processing without investor other features. They also include integrations with third-party services, such as payment gateways, CDNs, and CRM tools for performance and functionality.

Security is paramount because these platforms manage sensitive data, which can include user credentials and payment data. Poor protections are also leaving portals open to cyberattacks, such as data breaches and fraudulent transactions, which translate into financial loss and have the potential to erode customer trust.

Best practices to avoid these risks by e-commerce platforms. A single infinity data encryption transmits data somewhere, a multifactor authentication augments an account protection, a multiple-factor assessment called immediate fine rating access and role-based access control RBAC that restricts archive information to sensitive data acquisitions with sensitive permissions. Secure communication protocols (like HTTPS) prevent data interception. In addition, input validation prevents attacks such as SQL injection and cross-site scripting (XSS) and using reliable payment gateways gives you the assurance of following the Payment Card Industry Data Security Standards (PCI DSS).

We ensure user data security, maintain the system integrity, and establish user trust in safe and reliable shopping experience on the platform with these strategies applied.

A person sitting at a computer

Description automatically generated

Figure 1: Web Application System

**Django Framework for E-Commerce Development**

Django is a full-fledged open-source Python web framework that builds safe and maintainable web applications. Its Model-View-Template (MVT) architecture decouples business logic, data handling, and user interfaces, enabling developers to manage and scale applications more efficiently.

Django provides powerful security features for e-commerce platforms, including input validation, CSRF protection, and SQL injection prevention, all of which play a role in protecting sensitive customer data. Its popular Object-Relational Mapper (ORM) makes it easy to work with databases, helping to avoid common mistakes and giving the ability to use things like PostgreSQL and MySQL.

Django is great to scale, making it able to handle traffic spikes using caching, asynchronous task queues, and load balancing, all of which are highly suitable for busy shopping seasons. It also includes a built-in admin interface for easy management of products, orders, and customer data with limited custom development.

Tools from Django’s ecosystem like Django REST Framework (DRF) for API, and libraries like django-allauth for authentication with options for social login and multi-factor authentication. The DRY (Do not repeat yourself) principle of Django maintains the reusability of code and promotes a modular approach, which makes Django perfect to develop large data-intensive e-commerce applications that need regular updates.

Django allows you to build a compelling e-commerce application that not only meets your current needs but can also grow with you in the future thanks to its combination of security, scalability and flexibility.

A diagram of a web server

Description automatically generated

Figure 2: Django request and response cycle

# Security Issues Identification and Analysis in E-Commerce

Because these online stores handle sensitive customer and financial data, e-commerce websites are exposed to various threats. Each of the major vulnerabilities is explained in the following sections, and there are also examples of system limitations that show either a coding pitfall or describe inherent limitations of the security measure.

## Sensitive Data Exposure and Data Breaches

It is one of the TOP security risks for web applications that is also covered by OWASP. It happens when private or payment details such as credit card numbers, socially security numbers (SSNs), and address information are stored or transmitted without sufficient protection. This is often the result of poorly configured databases or weak encryption practices, which may leave sensitive data lingering in plaintext and open to attackers. Data leaks that affect user privacy are among the deadliest consequence of this vulnerability. The authors highlight that using key management best practices, including encryption of data at rest and in transit are fundamental to combat such attacks and to secure sensitive data (Kim et al., 2018).

A diagram of a computer network

Description automatically generated

Figure 3: Sensitive Data Exposure Diagram

We find that sensitive data can be leaked, as illustrated in Figure 3. This image illustrates an attacker exploiting a vulnerability, such as a misconfigured database. Bypass of weak security measures and privileged access to unprotected data. That emphasizes the necessity of the encryption and real-time observability of tools like data loss prevention (DLP) systems and intrusion detection systems (IDS). These enable businesses to identify and prevent security threats. Making it possible for attackers to exploit vulnerabilities they find in a system.

**System Limitation**

System limitations can prevent data from being adequately protected despite the security features in place. Real-time encryption could certainly reduce system performance dramatically particularly for high traffic seasons. This can create bottlenecks that slow operations down, leading to user dissatisfaction or even temporary outages of the system. Adding insult to injury, obsolete systems can be another major disadvantage. Larger-scale legacy systems or older systems may then on be poorly integrated with more recent protectors, since they were never supposed to incorporate encryption, in fact. As a result, businesses using this type of old infrastructure continues to be vulnerable even if they try to follow security best practices. Today, organizations are continually challenged to find a balance between performance, compatibility, and security in protecting sensitive data in the digital realm.

## Broken Authentication and Session Management

Broken authentication and session management is a major vulnerability and one of the top risks in OWASP security top ten. It occurs due to weak or misconfigured systems that handle user authentication, including password policies, session management, and access control. This allows the attackers to circumvent these layers of security, leading to the compromise of sensitive information and user accounts. Once gained access, they could access the data, steal information or escalate privileges. Alzomai & Almohammadi (2013) explain that this one of the most common vulnerabilities that an attacker can take advantage of in order to compromise sensitive information and the major attacks are credential stuffing, session hijacking, brute-force.

A computer screen shot of a computer

Description automatically generated

Figure 4: Broken Authentication Flow

The figure below (figure 4) demonstrates how another type of attack, a password reset attack, can exploit broken authentication. In this case, when a user makes a password reset request, the system generates a reset token and sends it to the user's email. If not properly secured, an attacker can read or get this token. In that way, the attacker can forge a request to the website with the compromised token and therefore can gain access to the user’s account without going through traditional authentication methods. This example highlights the need for proper token handling, secure communication channels, and strong session management to prevent this type of attack.

**Systems Limitation**

Authentication systems can be immutable in some way to facts or events, but attempting to tokenize the data may introduce certain limitations in the security mechanism. Examples include web servers (in a distributed, load-balanced fashion), network services, and cloud-based environments, where such servers are hosted in several machines. That is, session states may not be consistent across servers, leading to scenarios where expired sessions persist, or tokens are not invalidated appropriately. This allows attackers to hijack users' sessions and compromise their security. It is a high-risk area, which reveals that Systems can be fall short in protecting user data.

## Injection Attacks (SQL Injection)

Injection attacks are one of the biggest and most common security threats for web applications in the Wild today. SQL Injection (SQLi) is an evergreen example. OWASP identifies SQL injection (SQLI) as a high-severity vulnerability in web applications when inadequate sanitation of user input allows potential manipulation of queries sent to a database by an attacker. This occurs when untrusted input, such as form data or URL parameters, is directly integrated into SQL queries before validation or sanitisation. Attackers are abusing this vulnerability to run arbitrary SQL commands, and this may result in a data breach. SQLi was one of the most prevalent techniques being used to execute data breaches in industry sectors and can lead to various adverse outcomes such as data theft, data corruption, and total system compromise (Halfond, Viegas, Arch, 2006).

A diagram of a web application

Description automatically generated

Figure 5: SQL Injection Attack

The following figure illustrates the mechanics behind a standard SQL Injection attack (Figure 5). The attacker enters a malicious string like OR '1' = '1', which always returns true for the comparison on the query condition. With this vulnerability, the attacker can skip authentication checks and gain access to the underlying database. An example of this would be a vulnerable login form that executes an access-granting query without first checking if the user credentials are valid. This example demonstrates the importance of defensive measures like parameterized queries and input sanitization to prevent user inputs from being directly executed as SQL commands.

**System Limitation**

But even these protective measures have limitations in the system. Focusing on implementing parameterized queries and strict input validation lays a lot at the feet of developers, who often err in programming. In addition, many legacy applications were developed with static queries and have not adopted modern security procedures. Remediating these legacy systems to accommodate for best security practices is a time-consuming process, requiring substantial resources and possibly even longer periods of downtime. Consequently, such systems can be susceptible to injection for extended time periods, constituting a continuing security challenge. Finding the right balance between security and available, performant systems is an ongoing challenge for organizations that want to protect their applications against such kinds of exploits.

## Cross-Site Request Forgery (CSRF) Attacks

Cross-Site Request Forgery (CSRF) is an attack that exploits the trust a web application has in the user. In these types of attacks, hackers use a trick to get users — who are already logged in — to do things they don’t mean to do, like moving money, changing account settings or deleting important information. The problem is that the application has no reason to question the legitimacy of the request, as the user is authenticated. As discussed in (Jovanovic, Kruegel and Kirda, 2006), this is particularly the case when sensitive operations can be performed as a result of these forged requests, which makes CSRF such substantial threat to web security. Websites should use anti-CSRF tokens and also validate sessions to protect against such attacks.

A diagram of a diagram

Description automatically generated

Figure 6: CSRF Protection Workflow

An illustrative representation for a CSRF attack can be observed in Figure 6. First, the user logs into a secure site and receives a session token/cookie. An attacker, now aware that a user is authenticated, then crafts a malicious webpage or code and instructs the user to interact with it—whether through a phishing link or an embedded image. When the user clicks on this rogue content, the attacker’s code issues a forged request to the target site. Because the request contains the user's session token, the website processes it as if it were a valid request. Thus allowing the attacker to alter information, move funds or execute any other undesired transactions. Being Time No498: Yet another reminder why anti-CSRF tokens (unique identifier between a session and a request), a check of the origin of requests, and re-authentication for sensitive actions are so important.

**System Limitation**

Designing such functionalities inherently has some system limitations. Implementing anti-CSRF tokens throughout a web application, on every form and endpoint. Especially for dynamic applications that constantly add or update features and endpoints. Using a token to validate each request could also lead to synchronization problems (especially in distributed systems). Moreover, doing token verification on each request may increase the server latency, depreciating the user experience. Providing strong security without sacrificing system performance is an ongoing struggle that needs to be planned carefully and needs constant monitoring to prevent any security holes.

## Brute Force and Credential Stuffing Attacks

A straightforward approach that attackers frequently apply to breach user accounts is brute force and credential stuffing attacks. These attacks automate the process of attempting many username and password combinations until a correct one is identified. Brute Force Attack VS Credential Stuffing The difference between the two is that in case of a brute force attack, all combinations are tried; whereas with credential stuffing, you are using leaked username-password pairs obtained from earlier data leaks. Exploiting weak security controls, such as account lockout policies or the absence of multi-factor authentication (MFA), these attacks can circumvent authentication and lead to unauthorized access. These attacks are significant threats according to the OWASP, and Bonneau, Herley, and Van Oorschot (2012) demonstrate that countermeasures (such as rate-limiting, enforcing strong password policies, and implementing MFA) can greatly mitigate such threats.

**A diagram of a login password combination

Description automatically generated**

Figure 7: Brute Force Protection Mechanism

Figure 7 shows how these kind of attacks are performed. The attacker launches an automated script, grooming it to submit username and password credentials against the login page multiple times. If there are no protections in the application like rate limit or account lockout, the attacker will keep this up until they find a valid combination and gain access to system. This scenario shows why security mechanisms (such as requiring complex passwords and MFA (Multi-Factor Authentication) to add an additional layer of protection to accounts) and monitoring tools (looking for failed login attempts to flag account compromised and track suspicious activity) are essential no matter where authentication is performed.

**System Limitation**

We would have system restrictions with these countermeasures. For example, during high traffic periods or in cases where users forget their login information, rate limiting and account lockout policies may affect legitimate end users and disrupt service. Misidentifications can cause users to get locked out of accounts without cause, creating frustration and impact productivity. Legacy systems do not necessarily support modern security features, such as MFA, which can limit organizations' ability to protect against these attacks without investing in hardware and caching. Thus, it is a continuing challenge in the battle against brute force and credential stuffing attacks to balance security with user experience and the efficiency of the system.

## Security Misconfiguration and Insecure Defaults

Security misconfiguration is the most common and dangerous vulnerability in OWASP top security vulnerabilities in web applications. This happens when someone deploys servers, applications, or devices that are running with insecure settings or misconfigured defaults. Some common errors are leaving debug mode active, not deactivating unused services or failing to change default usernames and passwords. These vulnerabilities may leak sensitive system information, creating a way for an attacker to attack the system. As the underlying vulnerabilities that could lead to such attacks often arise from exposure to such code, regular security audits, proper configuration management, and a 'least privilege' implementation are prerequisites for reducing the risk of such attacks (Mutchler et al., 2015).

A diagram of a brick wall

Description automatically generated

Figure 8: Security Misconfiguration

Configuration is another topic that can lead to a breach, as shown in figure 8. The first step an attacker takes is to access the internal network, typically via open ports on misconfigured firewalls or as a result of unpatched/infected software. The attacker then scans the network for devices they can access with factory or weak logins. After they find a susceptible device, they seize control and obtain access to highly sensitive resources such as servers and databases. This goes to show how if you do not take the time to update default configs or regularly patch for security you can be severely exploited. Such a misuse would require robust access controls between the data layer and the application layer and regular patching from the user-side.

**System Limitation**

Even with the best effort, organizations live with more than significant limitations in systems to manage the security configurations. Misconfigurations are also more likely in large-scale enterprise systems that are regularly changed with frequent updates and configuration changes. Maintaining an audit trail of all these changes is labour intensive and susceptible to human error. And enforcing strict security policies across every system component consumes significant resources that many organizations struggle to provide. This is why; security misconfiguration is an evergreen challenges in enterprise environments.

## Insufficient Logging and Monitoring

OWASP has marked insufficient logging and monitoring as one of the main concerns for organizations when it comes to security vulnerabilities. That is when security-related incidents — like login attempts, network activity, and system changes — are not adequately recorded or examined. Security breaches may take a long time to be discovered without sufficient logging, making it possible that these incidents will continue for hours, days, or even weeks without detection, thereby delaying incident response efforts and causing greater damage. This flaw allows the attackers to launch prolonged attacks while avoiding detection. According to (Makanju, Brooks, and Zincir-Heywood, 2009), effective logging, real-time monitoring, and well-defined incident response protocols are essential for timely detection of and response to security threats.

A diagram of a computer network

Description automatically generated

Figure 9: Logging and Monitoring System

Good monitoring is about being proactive. This includes implementing systems for real-time threat detection, automated alerts, and comprehensive logging of all security-relevant events. This is illustrated in a botnet attack in Figure 9. In this case, a botmaster has a number of compromised devices (bots) that can perform tasks like distributing Trojans, executing web exploits, launching phishing campaigns and starting a Distributed Denial of Service (DDoS) attack. If there is no proper logging and monitoring in place, such attacks go unnoticed, and the attacker keeps exploiting the system vulnerabilities. This underlines the need for the IDS, DLP, and regular log review that allows to detect and prevent this kind of attack at an early stage.

**System Limitation**

However, implementing comprehensive logging and monitoring is not straightforward. Real-time monitoring use in high-volume applications represents a very expensive and resource intensive since it generates large volumes of data. Most organizations are limited in terms of how much log data they can store and generally have poor log retention policies in place which may hinder the detection of long term or multi-phase attacks. Many businesses struggle to strike a balance between the need for exhaustive logging and the cost associated with storing and analysing data. However, investing in strong monitoring systems is critical to safeguard ourselves from modern cyber-attacks, despite these limitations.

## Weak Authentication and Authorization Mechanisms

OWASP identifies weak authentication and authorization mechanisms in web applications as one of the top security issues. These vulnerabilities stem from weak password policies, lack of multi-factor authentication (MFA), and insufficiently enforced role-based access controls (RBAC). When these mechanisms are lacking, attackers can exploit user accounts with brute force, credential stuffing, or social engineering easily. After getting inside, attackers can elevate their privileges and obtain access to sensitive data and important system resources in addition to enabling further exploitation. Kuhn, Hu & Polk, 2010) Bots attack the API through password spraying, vulnerabilities in RBAC, and issues with private keys, and the implementation of a strong password policy, MFA, and a reliable RBAC can help mitigate these risks and ensure a stronger security posture for a system.

A diagram of a computer and a device

Description automatically generated

Figure 10: Weak Authentication and Account Takeover

Figure 10 shows how these vulnerabilities can be combined to allow for cross-account takeovers. In the absence of sufficient protections, malicious actors abuse these vulnerabilities by repeatedly trying to log in using various combinations of credentials. Integrating a risk-based authentication solution during the login process allows for the assessment of multiple factors, including device type, location, and user behaviour patterns, to help counter this. For instance, if a user tries to log in from an unfamiliar device in a different country, they may get additional verification (such as a one-time code sent to their phone). That layered approach can make it much more challenging for attackers to evade security.

**System Limitation**

While this brings advantages, there are challenges associated with strengthening authentication and authorization systems. The challenges of implementing robust RBAC and MFA often add significant development, integration and support costs. This measure has compatibility issues with legacy systems faced by many organizations integrating them. Also, user adoption can be another hurdle— MFA is resisted by some users, who find the extra step of logging into their account annoying and this, in turn, deflates the efficiency of these security augmentations.” Security vs Usability: Organizations need to find a happy medium between security and usability measures that are effective yet user-friendly so that they can be adopted widespread.

# Requirements Identification

## Authentication and Authorization

Authentication and authorization are one of the central or key security concerns in my e-commerce application. Such mechanisms help authenticate users to allow only valid users to interact with the system and its resources. Authentication establishes the user, usually via credentials such as a username and password and authorization define the level of access for each user according to his role (admin, merchant or customer). If these protections were absent, then nefarious actors could assume alternate identities, inquire about sources of access, or magnify their authority to carry out nefarious tasks, potentially compromising sensitive information.

To reduce these risks, role-based access control (RBAC) is introduced, which limits the access to system features according to user roles. This restricts the extent of what each user can view or manipulate in the system and limits the harm that can be caused by others accessing their information. Moreover, protected authentication procedures, including password hashing and strong session management, defend against attack vectors such as credential stuffing and brute-force attempts. MFA adds a critical second layer of protection that forces users to authenticate their identity in another way, such as an OTP.



Figure 11: Authentication and Authorization

In my e-commerce app, Django handles the user login and password verification in the backend with the built-in authentication system. This setup cuts down the whole work of authentication in segments and also ensures security and reliability without a lot of manual coding. In my case, I implemented a custom form named CustomLoginForm, which allows to access Django’s own authenticate() function to perform a secure check if the user’s password is correct. All of this happens in the background, isolating sensitive operations from possible attackers.

(For example, admin panel or account pages, etc.)– to ensure that critical parts of the site are protected, we’ve put access control directly in the backend. Django decorators (things like @login\_required) prevent a user from accessing certain views so that only logged in users can access those views/pages. For tighter security, I use @user\_passes\_test(), so I can filter out who can see what. That could mean checks to see what the user’s role is, for example, “only staff can see the order management system.”

To further secure the application, I’ve also implemented a two\_factor\_required decorator. It enforces MFA (multifactor authentication), meaning if users entered their password, they would need to complete another verification step. This renders attackers much less likely to be able to crack accounts, even if they manage to obtain login credentials in some way. All of these backend security features work together to provide strong protection against unauthorized access and prevent account compromises, ensuring the safety of my application and its users.

## Data Encryption

Data encryption is another major issue for my e-commerce platform to safeguard the sensitive data. Encryption takes human-readable data and turns it into a nonsensical format in which only trusted users with a secure key are able to decrypt. So, even if attackers penetrate the system, they cannot read or misuse sensitive data like personal data, passwords, or payment information. Sensitive data may be seen without sufficient encryption, leading to serious privacy and security breaches.

It is through encryption of data at rest and in transit that data breaches can be mitigated. For data in transit, communication secure protocols such as HTTPS protects the data exchanged from the client and server. For data in repose, strong algorithms like AES-256 encrypt data within the database — the enigmatic data will be indecipherable to any unauthorized user who obtains it. Skillful encryption key management is also essential to disallow unauthorized decryption.

A computer screen shot of several keys and a lock

AI-generated content may be incorrect.

Figure 12: Data Encryption

In my e-commerce application that by nature has sensitive data fields (the sensitive\_info field in the CustomUser model) and as such is encrypted at the backend for security. I utilize Django’s EncryptedCharField, which is based on AES-256 encryption, a strong industry standard for protecting data. This means when data is saved, it is automatically encrypted, and so even in the event of a database breach, the data will not be readable without the associated decryption key.

On the backend I have an EncryptionHelper class that handles the encryptions and decryption operations. All the grunt work is handled by this class, making sure that data is being correctly encrypted on storage and decrypted on retrieval. The actual encryption key is safely stored as part of the application settings in an ENCRYPTION\_KEY constant. This prevents the critical key from being accessible even by developers with access to the database. Key management is similarly important to ensure replayed messages aren't decrypted.

## Secure Password Handling

Insecure password storage is a common weakness of web applications. Plain text passwords are a major risk; if a system is hacked (after which it is too late), attackers can read this information without any effort. Instead, the password should be hashed and salted with strong algorithms in a way so that deriving the original password from it is not feasible. This is an important measure against unauthorized access to user accounts from credential leaks.

To counter such risks, the password is hashed prior to storing. Each time user’s login, the input they provide is hashed and compared to the stored hash, so plaintext passwords are never being exposed. They would implement strong password policies that require a minimum length and a combination of letters, numbers, and special characters, account lockout, and account lockout after multiple failed attempts, and secure session management among others.

A red padlock and a number

AI-generated content may be incorrect.

Figure 13: Secure Password Handling

For my application, I’m utilizing password security within my e-commerce application — so, it’s handled from the backend using the Django’s set\_password() function. This function not only stores passwords—it uses PBKDF2 hashing which is the most secure way of turning plain-text passwords into cryptographic hashes. This means, if someone cracks the database there is no way to read or reverse engineer the actual passwords. Instead, they will only see hashed data, a string of code that would take a huge amount of time and resources to crack.

Django provides us with a very important function to prevent logged in users to be logged out when updating the password, called as update\_session\_auth\_hash() function. Without this step, the user’s session may become invalid, and they would be required to log in again. Worse, not updating the session hash doesn't just render it useless, it leaves the door open for session hijacking since an attacker could exploit session inconsistencies to break into the session. It allows me to lower the risk of account hijacking by keeping the session hash current.

## One-Time Password (OTP) Verification

One-time password (OTP) validation is a common security feature implemented in many scenarios, including account recovery and multi-factor authentication. OTPs are transient codes, sent to a user through email or text, that must be entered for a secure action to take place. Since these codes have a short expiration time and can only be used once, they minimize risk of unauthorized access to a great degree, even in case if user’s password is stolen.

An OTP verification helps mitigate attacks by requiring the password of the user and an additional secure code. This two-layer approach makes sure that attackers cannot reach an account without the OTP. To enhance security, OTPs need to be randomly generated, encrypted, and time-limited, meaning hackers cannot try countless brute-force attempts or reuse codes.

A blue rectangular sign with text and a phone

AI-generated content may be incorrect.

Figure 14: OTP verification

My system uses a function called generate\_secure\_otp() that does this magic. These codes are saved inside the EmailOTP model and are sent to the users by email every time they want to execute secure actions, such as resetting passwords.

Validation happens whatever a user inserts their OTP and checks through whether it is substantial and has not lapsed. As a result, only users who have the permission to do so will take sensitive actions after this verification step. After the OTP is successfully verified, the system immediately invalidates OTP to ensure that it will not be reused. This defends against cases where an attacker attempts to reuse an old OTP or the one they intercept.

## CAPTCHA for Bot Protection

Computers, or bots, can be used for automated attacks like credential stuffing and brute-force attempts — repeatedly submitting forms. In order to combat this, my program utilizes CAPTCHA, which requires users to complete a task that is simple for humans and complex for bots. CAPTCHA reporter requires visual or cognitive recognition tasks that make it difficult for automated scripts to abuse public forms, for example.

To mitigate against automated attacks, the same form is used to block repeated submissions by a bot. This is technically complicated and requires significant resources for attackers seeking to automate the attacks to evade CAPTCHA challenges. This makes CAPTCHA a critical security measure for sensitive forms, such as login and registration pages.

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 15: Captcha

In my app I used the CaptchaDjango Django’s CaptchaField. ReCAPTCHA protects key forms, such as login and password reset forms, to prohibit automated abuse like bot attacks — this additional layer of security is added on top of these forms.

When such forms are interacted with an user is required to solve a CAPTCHA challenge. When the user fails to pass the challenge, their form submission silently fails. This prevents automated scripts and bots from flooding the system with login attempts or password reset requests and thus serves as an effective defense mechanism against brute-force and denial-of-service attacks. On pages of any sensitive sites (login and registration pages) CAPTCHA is an important protective measure.

## CSRF Protection

Cross Site Request Forgery (CSRF) is designed to exploit the trust that a site has for a user by causing an authenticated user to perform an action not intended by the user against a known trusted site. That could mean coercing a logged-in user to send a form to change their account preference on some villain site, for instance. CSRF protections operate on the insertion of unique tokens into each form submission:

The mitigation is to check that CSRF token is valid for every POST request. The token is unique to a particular session, preventing request forgery attacks. Most modern frameworks, like Django, already have CSRF protection enabled by default.

A diagram of a website

AI-generated content may be incorrect.

Figure 16: CSRF Protection

My e-commerce application uses Django’s csrf\_token template tag to prevent CSRF vulnerabilities. A user session CSRF token is generated and put behind every form associated with it in the system.

On submitting a form, the server verifies that the CSRF token in the request matches the one that it expects. If the token is not present or invalid, it simply blocks the request. This protects the target user from being duped to take actions that the attacker wants them to take such as changing account information or making purchases without the knowledge of the target. It's an essential backend security step for trust and safety of the app.

## Access Attempt Monitoring and Account Lockout

Lastly, account security is still threatened by brute-force attacks. Also, attackers can automate a number of logins attempts with different password combinations. To alleviate this, I made sure that my application implemented access monitoring (e.g. detecting failed logins or failed attempts) and account lockout to where if they tried logging in too many times, they would not be able to log in again until the account was unlocked.

This mitigation technique deters would-be attackers by making their efforts take longer and increasing the chances of their catching a watchful eye. Failed login attempts are logged, and suspicious activity can be monitored by the administrator for corrective action.

A diagram of a number of passwords

AI-generated content may be incorrect.

Figure 17: Account Lockout

My system uses Axes to track access attempts and improve security. To cover brute force attack, there is AccessAttempt model that stores failed attempts to login. Axes can lock the account or even temporarily lock login attempts by the offending IP address when a lot of failed attempts happen.

These access attempts can easily be managed by the admins from the admin panel; admins can check logs, unlock accounts, or whitelist trusted users as required. Not only does this help thwart unauthorized access, but it also means legitimate users can easily regain access with administrative assistance if they get into trouble.

.

# Conclusion

In conclusion, keeping sensitive information safe is a serious business for e-commerce platforms. If not well-managed, these threats like data losses, account creaking, and service outages can lead to serious damage. The report also shed light on common threats such as data exposure, broken authentication, injection attacks, and effective strategies to mitigate them.

Django's inbuilt security standards, secure password hashing, role-based access control - MFA, CSRF protection and access monitoring platforms can secure users and data. System Limitations Because of legacy infrastructure and performance overhead, systems have certain limits and constraints which can surface as issues in secure development practices; however, regular updates, audits and following best practices mitigate these risks.

In the end, security is a continuous process. Through a combination of strong application architecture and a proactive approach to ever evolving threats, enterprises can build trust in their platform, secure their platform and ensure their longevity in the digital storefronts.

# References

Alzomai, M. &. ((2013)). Authentication vulnerabilities in web applications: Attacks and defense mechanisms. *International Journal of Computer Science and Security*, 7(4), 155-167.

Bonneau, J. H. (2012). The quest to replace password: A framework for comparative evaluation of web authentication schemes. *IEE Symposium on Security and Privacy*, pp. 553-567.

Halfond, W. G. (2006). A classification of SQL-injection attacks and countermeasures. *IEEE International Symposium on Secure Software Engineering*, pp. 13-15.

Jovanovic, N. K. (2006). Preventing cross-site request forgery attacks. *IEE*, pp. 1-10.

Kim, D. C. ((2018)). A study on data encryption for secure database protection. *Journal of Information Security and Application*, 42, 40-50.

Kuhn, D. R. (2010). Introduction to role-based access control. National Institute of Standard and Technology (NIST). *Special Publication*, 800-95.

Makanju, A. B.-H. (2009). Log analysis for intrusion detection: A machine learning approach. *Journal of Information Assurance and Security*, 4(2), 141-150.

Mutchler, L. P. (2015). Security misconfiguration detection in web applications using machine learning. . *10th International Conference on Securiy and Criptography*, (pp. 423-432).