

Group I “Bango” Security Report

Date: 2023-04-10

Table of Contents

Table of Contents	2
Disclaimer	3
Finding Severity Ratings	4
Scope	5
Scope Exclusions	5
Executive Summary	6
Attack Summary.....	6
Security Strengths	7
Azure CORS policy prevents external IP addresses from connecting to the database.....	7
Proper ORM implementation prevents XSS injection and SQL injection.	7
Security Weaknesses	8
No TLS	8
Weak Password Policy.....	Error! Bookmark not defined.
Attackers were not detected during testing. No action was taken to prevent the attack.	8
Weak cookie secret management.....	8
Audit Findings	9
Exposed / Reused Credentials– Database server (Critical)	9
Poor session security / using default secrets – Web server (Low)	12
Additional Reports and Scans (Informational)	12

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Scope

Assessment	Details
External Penetration Test	http://206.189.48.173:8080 ,
Internal code audit	https://github.com/GustavBrygger/DevOps

Scope Exclusions

Group i did not perform any Denial-of-Service attacks during testing and took additional considerations to preserve “Bangos” database and data integrity during the attack in order to not cause any data loss.

Additionally, all endpoints used by the teachers to insert data were excluded from testing as it would be difficult (Or impossible) to patch out potential vulnerabilities and any exploits would impact “Bangos” data integrity.

Executive Summary

Group i evaluated “Bango”s external security through an external network penetration test from March 28th, 2023 to April 9th, 2023. By leveraging a series of attacks, the attackers have found 1 critical level vulnerability that allowed full external access to the “Bangos” database. Additionally, the internal security audit has revealed vulnerabilities that could cause significant harm to both clients and “Bangos” operations. It is highly recommended that “Bango” address these vulnerabilities as soon as possible as the vulnerabilities are easily found through basic reconnaissance and exploitable without much effort.

Attack Summary

The following table describes what attacks we made and their results:

	Exploit	Result
1	XSS injection	All XSS injection attempts failed. The ORM chosen by “Bango” is sufficient to parse out malicious code.
2	SQL Injection	All SQL injection attempts were unsuccessful as ORM sufficiently parsed our malicious code.
3	Metasploit full attack suite.	Metasploit suite provided no successful attacks but revealed information that led to critical vulnerability.

4	OWASP zap attack suite	See “ZAP Scanning Report_1.pdf”
5	Changing Cookie data	Successful. The team managed to create identical cookies as Bango’s with new id’s.
6	Scanning the Github for possible exposed passwords.	Successful. Commit history contains passwords that provide access to the database.

Security Strengths

Azure CORS policy prevents external IP addresses from connecting to the database.

During the assessment, azure database link was identified inside “Bango”’s GitHub repository. However, due to the proper firewall set up, the attackers were unable to gain external access to the database.

Proper ORM implementation prevents XSS injection and SQL injection.

During the assessment no XSS scripts or SQL injection scripts were successful.

Security Weaknesses

No TLS

There is no TLS implemented. Exposing users to eavesdropping / message tampering between server and website.

Attackers were not detected during testing. No action was taken to prevent the attack.

During the assessment, “Bango” has not once notified the team that they detected our attempts to exploit the website. Additionally, no attempts to exploit the website from the same IP address were stopped.

Weak cookie secret management

“Bango” has left default cookie secret set up. Which allows the sessions to be replicated.

Audit Findings

Exposed / Reused Credentials– Database server (Critical)

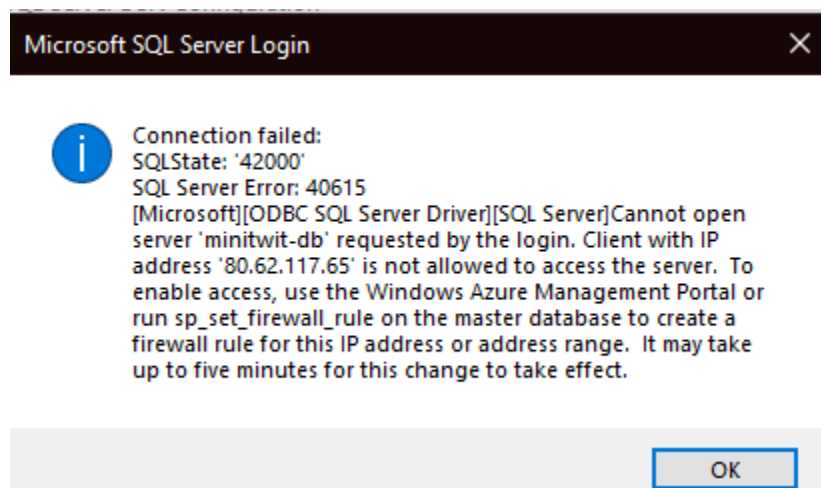
Description:	“Bango” has not changed their database super user credentials. Using an exposed port discovered during initial information gathering by Metasploit. The attacker managed to connect to the database and used inbuilt super user credentials to establish full root control over the product database.
Impact:	Critical
System:	http://206.189.48.173:5432
References:	https://github.com/organizationGB/DevOps/commit/fcd5a04ad83e8687d0d9cc82d1a4898e936b55e7

Exploitation Proof of Concept

```
13 14
14 15 var localConnectionString = fmt.Sprintf("host=%s user=%s password=%s port=%d dbname=%s", "localhost", "postgres", "postgres", 5432, "postgres")
15 16 - var azureConnectionString = fmt.Sprintf("host=%s user=%s password=%s port=%d dbname=%s",
16 17 + var azureConnectionString = fmt.Sprintf("server=%s;user_id=%s;password=%s;port=%d;database=%s;",
16 17 "minitwit-db.database.windows.net", "minitwit", "dbpassword1!", 1433, "minitwit-db")
17 18
18 19 func GetDbConnection() *gorm.DB {
19 20 connString := localConnectionString
20 21 isProduction := os.Getenv("IS_PRODUCTION")
21 22 if isProduction == "TRUE" {
22 23 connString = azureConnectionString
23 24 + dsn := "sqlserver://minitwit:dbpassword1!@minitwit-db.database.windows.net:1433?database=minitwit-db"
```

Firstly, the attacker has looked through the historical commit history and noted that “Bango” uses PostgreSQL and the credentials that were used in the development.

Once completed, the team attempted to log into the deployed azure server with said credentials, however the attempts were unsuccessful.

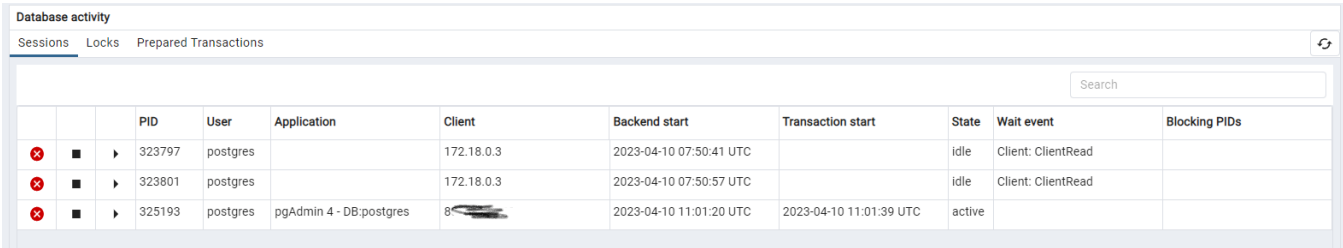


The attacker then returned to the information gathering stage and listed out all ports that “Bango” is using. During this stage, port 5432 was discovered to be open and does not restrict IP address log in attempts.

Another attempt to log into the database was made on <http://206.189.48.173:5432>.

The credentials that were used in commit history proved to be incorrect as “Bango” has removed such user. However, the team failed to change the superuser password which allowed the attacker to have full and root control over the product database.

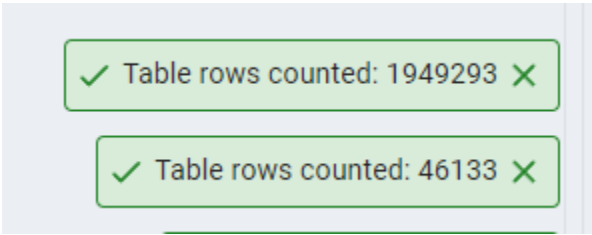
Additionally, while azure server was used for database actions. Having access to the website PostgreSQL server has provided access to the Azure server.



The screenshot shows the 'Database activity' window in a PostgreSQL management tool. It has tabs for 'Sessions', 'Locks', and 'Prepared Transactions'. The 'Sessions' tab is active, displaying a table of database sessions. The table has columns for PID, User, Application, Client, Backend start, Transaction start, State, Wait event, and Blocking PIDs. Three sessions are listed, all from IP 172.18.0.3. The first two are idle, and the third is active.

			PID	User	Application	Client	Backend start	Transaction start	State	Wait event	Blocking PIDs
✖	■	▶	323797	postgres		172.18.0.3	2023-04-10 07:50:41 UTC		idle	Client: ClientRead	
✖	■	▶	323801	postgres		172.18.0.3	2023-04-10 07:50:57 UTC		idle	Client: ClientRead	
✖	■	▶	325193	postgres	pgAdmin 4 - DB:postgres	8	2023-04-10 11:01:20 UTC	2023-04-10 11:01:39 UTC	active		

Additionally, to confirm that the attacker has full access over the production server. A simple row count was done on both “users” and “messages” table.



The screenshot shows two green boxes with checkmarks and 'X' icons, indicating successful row counts. The first box says 'Table rows counted: 1949293' and the second box says 'Table rows counted: 46133'.

✓ Table rows counted: 1949293 ✖
✓ Table rows counted: 46133 ✖

Once successfully confirmed. Any number of actions by the attacker can be taken such as creating new users with root access. Modifying / copying the current database and all credentials... ect

Remediation

Who:	IT Team
Vector:	Remote
Action:	Recommendation 1: Multiple attempts to log into the server were made by the attacker. A proper firewall / password policy should be implemented, and IP

	<p>addresses should be locked out / timeout after multiple failed attempts to log in.</p> <p>Recommendation 2: All passwords should be changed. Passwords should not be easy to guess and should be difficult to brute force (14 characters. Letters, Capital letters, numbers and symbols)</p> <p>Recommendation 3: Azure database link and all revealing database information should be stored in Github secrets.</p> <p>Recommendation 4: Proper logging and notifications for the development team should be implemented.</p>
--	---

Poor session security / using default secrets – Web server (Low)

Description:	"Bango" has not changed their database super user credentials. Using an exposed port discovered during initial information gathering by Metasploit. The attacker managed to connect to the database and used inbuilt super user credentials to establish full root control over the product database.
Impact:	Low. Can be used to imitate users.
System:	http://206.189.48.173:8080
References:	https://github.com/organizationGB/DevOps/blob/main/src/web/controller/session.go

Exploitation Proof of Concept

During the information gathering. The attacker noticed that the salt used for cookies is exposed and static.

```
func ConfigureSession(router *gin.Engine) {  
    store := cookie.NewStore([]byte("secret"))  
    store.Options(sessions.Options{MaxAge: 60 * 60 * 24})  
    router.Use(sessions.Sessions("mysession", store))  
}
```

This has allowed the attacker to use the same secret to create identical cookies that the website would issue. As a result, the attacker managed to create cookies with different ID's and insert messages as a different user.

Remediation

Who:	IT Team
Vector:	Internal
Action:	Recommendation 1: Cookie secret should be used from. env or Github secrets. Recommendation 2: Cookie secret should not be the default secret.

Additional Reports and Scans (Informational)

For additional info check out: "ZAP Scanning Report_1.pdf"

Last Page