e

CD SECURITY

AUDIT REPORT

Drew Security Bot

October 2024

Introduction

A time-boxed security review of the **Drew Security Bot** was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Drew Security Bot**

The Drew Security Bot is a Discord's bot that supports server's owner maintain the channels' permissions. The bot requires team members to enter a separate 2FA code to post an announcement and use the `@everyone` permission, with a temporary time limit set by the admin. This security feature helps prevent scenarios where a compromised team or moderator account could post malicious links in the announcement channel and spam everyone without additional authentication.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - a275855f687ca1788027882e20c45c8ecee1ddf0**

Scope

The following smart contracts were in scope of the audit:

- `Everything under ./src folder`

The following number of issues were found, categorized by their severity:

- Critical & High: 3 issues
- Medium: 7 issues
- Low: 4 issues

Findings Summary

ID	Title	Severity
[C-01]	Some permissions are never revoked	Critical
[H-01]	Lack of `whitelistedAt` reset can lead to problems	High
[H-02]	The admin user cannot call `unauthenticate` command	High
[M-01]	The application uses cryptographically insecure random numbers	Medium
[M-02]	Application lacks whitelist cancel command	Medium
[M-03]	The `reset-2fa command` doesn't terminate current session	Medium
[M-04]	Automatic session termination can be delayed	Medium
[M-05]	Permissions remain granted after channel removal	Medium
[M-06]	Inconsistent application of session check	Medium
[M-07]	User's TOTP secret is exposed to the third party service	Medium
[L-01]	User's username is exposed to the third party service	Low
[L-02]	The authentication timeout lacks upper band input validation	Low
[L-03]	Inaccurate information can be presented by show-list command	Low
[L-04]	The application availability depends on the third party API	Low
[I-01]	Users metadata are stored under third party control	Informational
[I-02]	Undocumented behavior of some commands	Informational
[I-03]	Lack of opt-out and permanent data wipe functionality	Informational
[I-04]	Bot does not log lack of permissions	Informational
[I-05]	The `authenticatedRoleId` column is not used	Informational
[I-06]	Vulnerable dependencies in use	Informational
[I-07]	DoS is possible via database records flood	Informational

Detailed Findings

[C-01] Some permissions are never revoked

Severity

Impact: High

Likelihood: High

Description

The bot allows server's owner grant users access to Discord's channels with two different commands: `set-channel` and `set-user-channel`.

- The `set-channel` allows all registered users access channels from this list. This configuration is saved in the `WhitelistedChannel` table.
- The `set-user-channel` allows selected registered users access selected channels from this list. This configuration is saved in the `UserWhitelistedChannel` table.

Whenever registered user authorise him/herself in the bot application, the `updateAuthentication` function is executed to grant permissions to user.

File: `index.ts`

```
...
            const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
                where: {
                    serverId: interaction.guildId!
                }
            });

            const userWhitelistedChannels = await
Postgres.getRepository(UserWhitelistedChannel).find({
                where: {
                    userId: interaction.user.id,
                    serverId: interaction.guildId!
                }
            });
...
            updateAuthentication(interaction.member as GuildMember,
                userWhitelistedChannels.length
                    ? userWhitelistedChannels.map((channel) =>
channel.channelId)
                    : whitelistedChannels.map((channel) =>
channel.channelId)
            , true);
```

Within the `updateAuthentication` function the user is granted following permissions:

- ViewChannel

- MentionEveryone
- EmbedLinks
- AttachFiles
- ReadMessageHistory
- SendMessages

File: util.ts

```
export const updateAuthentication = async (
  member: GuildMember,
  channelIds: string[],
  enabled: boolean
) => {
  const promises: Promise<void | NonThreadGuildBasedChannel>[] = [];

  channelIds.forEach((channelId) => {
    promises.push(
      member.guild.channels
        .fetch(channelId)
        .then((channel) => {
          if (enabled) {
            return (channel as TextChannel).permissionOverwrites
              .create(member, {
                ViewChannel: true,
                MentionEveryone: true,
                EmbedLinks: true,
                AttachFiles: true,
                ReadMessageHistory: true,
                SendMessages: true,
              })
              .finally(() => {
                console.log(
                  `Permissions granted for ${member.user.tag} in ${
                    channel!.name
                  }`
                );
              })
              .catch((err) => {
                console.error(err);
              });
          } else {
            return (channel as
TextChannel).permissionOverwrites.delete(member);
          }
        })
        .catch((err) => {
          console.error(err);
        })
    );
  });
```

```
  return Promise.allSettled(promises);
};
```

These permissions are meant to be granted timely, and after session timeout they are supposed to be revoked. There are two possibilities to revoke privileges. Firstly, by means of the `unauthenticate` command.

File `unauthenticate.ts`

```
...
    const member = await interaction.guild!.members.fetch(user.id);

    const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
        where: {
            serverId: interaction.guildId!
        }
    });
    updateAuthentication(member, whitelistedChannels.map(c =>
c.channelId), false);

    return interaction.reply(successEmbed(`User ${user.toString()} has
been unauthenticated!`));
}
```

Secondly, by the cron's task `check-session`.

File: `check-session.ts`

```
...
                const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
                    where: {
                        serverId: session.serverId
                    }
                });

                const member = await
client.guilds.fetch(session.serverId).then((guild) =>
guild.members.fetch(session.userId));
                updateAuthentication(member,
whitelistedChannels.map((channel) => channel.channelId), false);
...
```

However, in both cases the permissions revoke is being done only for channels configured via `set-channel` command and not for `set-user-channel`. Thus, every user authorised to get permissions for selected channels only is granted permissions permanently.

Recommendations

It is recommended to revoke permissions to all configured channels for particular user upon session timeout.

[H-01] Lack of `whitelistedAt` reset can lead to problems

Severity

Impact: High

Likelihood: Medium

Description

Before new user can register in the bot application, he/she must be whitelisted by the server's owner. This action can be done by the `whitelist` command. Upon user's whitelisting, the `whitelistedAt` property is updated with the current date and time.

File: `whitelist.ts`

```
...
    const user = (interaction.options as
CommandInteractionOptionResolver).getUser('user', true);

    await Postgres.getRepository(UserAuthenticationWhitelist).insert({
        userId: user.id,
        serverId: interaction.guildId!,
        whitelistedBy: interaction.user.id,
        whitelistedAt: new Date()
    });

    return interaction.reply(successEmbed(`User ${user.toString()} added
to authentication whitelist for the next 7 days!`));

}
```

Then, user has 7 days to register in the bot application. Whenever the user completes the registration process the `authenticationEnabled` property is toggled on to allow user further use of one-time-passwords (OTP). However, the `whitelistedAt` property remains unchanged.

```
File: `index.ts`
...
            userAuthentication.authenticationEnabled = true;
            await
Postgres.getRepository(UserAuthentication).save(userAuthentication);
```

```
            interaction.reply(successEmbed('Your 2FA authentication has
been enabled! You can now use the /post command (or /admin one).'));

            sendLogs(client, serverSettings!, `**${interaction.user.tag}**
- enabled 2FA authentication.`);
...
```

This weakness leads to the situation that in the event of emergency, even if the user registration is revoked by the `reset-2fa` command, he/she still can authenticate again for full 7 days since whitelisted.

Recommendations

It is recommended to reset the `whitelistedAt` property upon successful registration.

[H-02] The admin user cannot call `unauthenticate` command

Severity

Impact: High

Likelihood: Medium

Description

The bot allows the server's owner to configure additional list of admins that can manage the bot configuration. Whenever user is added to admins list, it is granted access to the following commands:

- reset-2fa
- set-auth-timeout
- set-boost-channel
- set-boost-role
- set-channel
- set-logging-channel
- set-user-channel
- show-channels
- show-list
- show-user-channels
- whitelist

However, it has no access to the `unauthenticate` command, which timeouts user's session. Having in mind that admin has access to the `reset-2fa` command which can be considered superior over the `unauthenticate` command, it appears that it lacks access to this essential functionality.

File: `unauthenticate.ts`

```
```

```
...
export const run: SlashCommandRunFunction = async (interaction) => {

    const isOwner = (interaction.member as GuildMember).guild.ownerId ===
interaction.user.id;
    if (!isOwner) return void interaction.reply(errorEmbed('You must be
the server owner to use this command'));

    const user = (interaction.options as
CommandInteractionOptionResolver).getUser('user', true);

    await Postgres.getRepository(UserSession).update({
        userId: user.id,
        serverId: interaction.guildId!
    }, {
        isEnded: true
    });

    const member = await interaction.guild!.members.fetch(user.id);

    const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
        where: {
            serverId: interaction.guildId!
        }
    });
    updateAuthentication(member, whitelistedChannels.map(c =>
c.channelId), false);

    return interaction.reply(successEmbed(`User ${user.toString()} has
been unauthenticated!`));
}
```

Recommendations

It is recommended to change the implementation so the admin is allowed to call the `unauthenticate` command.

[M-01] The application uses cryptographically insecure random numbers

Severity

Impact: High

Likelihood: Low

Description

The main functionality of the bot application is to authorise whitelisted users to give them timely a set of privileges allowing posting messages on the particular Discord's channel. The authorisation is done by generating and verifying one-time-passwords (OTP). The OTP is generated by means of random hash generated per user by the application. However, within this process the `Math.random()` function is being used. The `Math.random()` function is not considered cryptographically secure. E.g. Mozilla states that:

> Note: Math.random() does not provide cryptographically secure random numbers. Do not use them for anything related to security. Use the Web Crypto API instead, and more precisely the Crypto.getRandomValues() method.

Thus, this weakness makes users' secrets to have possibly predictable values.

File: `authenticate.ts`

```
...
    if (!authenticationWhitelist) {
        return void
interaction.reply(errorEmbed(messages.NOT_WHITELISTED));
    }

    const validBase32Character = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567';
    const generatedHash = Array(16).fill(0).map(() =>
validBase32Character.charAt(Math.floor(Math.random() *
validBase32Character.length))).join('');

    const new2FA = new UserAuthentication();
    new2FA.userId = interaction.user.id;
    new2FA.serverId = interaction.guildId!;
    new2FA.generatedHash = generatedHash;
    new2FA.authenticationEnabled = false;
    await Postgres.getRepository(UserAuthentication).save(new2FA);
...
```

Recommendations

To generate secrets it is recommended to use cryptographically secure functions, such as Crypto API method: `crypto.getRandomValues()`.

[M-02] Application lacks whitelist cancel command

Severity

Impact: Medium

Likelihood: Medium

Description

Before new user can register in the bot application, he/she must be whitelisted by the server's owner. This action can be done by the `whitelist` command. Upon user's whitelisting, the `whitelistedAt` property is updated with the current date and time.

File: `whitelist.ts`

```
...
    const user = (interaction.options as
CommandInteractionOptionResolver).getUser('user', true);

    await Postgres.getRepository(UserAuthenticationWhitelist).insert({
        userId: user.id,
        serverId: interaction.guildId!,
        whitelistedBy: interaction.user.id,
        whitelistedAt: new Date()
    });

    return interaction.reply(successEmbed(`User ${user.toString()} added
to authentication whitelist for the next 7 days!`));

}
```

Then, user has 7 days to register in the bot application. However, there is not function to revert the whitelisting. Thus, in the event of the emergency, there is no possibility to disallow authentication by the user, until 7 full days are passed.

```
File: `authenticate.ts`
...
    const authenticationWhitelist = await
Postgres.getRepository(UserAuthenticationWhitelist).findOne({
        where: {
            userId: interaction.user.id,
            serverId: interaction.guildId!,
            whitelistedAt: MoreThan(new Date(Date.now() - 1000 * 60 * 60 *
24 * 7))
        }
    });
...
```

Recommendations

It is recommended to implement function that immediately removes particular user from the whitelist. The function must have authorisation implemented, that verifies whether the sever's owner or admin is executing it.

[M-03] The `reset-2fa command` doesn't terminate current session

Severity

Impact: Medium

Likelihood: Medium

Description

The server's owner can reset 2FA configuration of any user by means of `reset-2fa` command. After calling it, user will be not capable to generate one-time-passwords (OTP) anymore and new registration will be required. Technically, it switches the `authenticationEnabled` parameter off. However, this command does not terminate current session, which is flagged in the `isEnded` property. Whenever user has active session, this user will be capable to post until session timeout or the `unauthenticate` command is called.

File: `reset-2fa.ts`

```
...
    const user = (interaction.options as
CommandInteractionOptionResolver).getUser('user', true);

    await Postgres.getRepository(UserAuthentication).update({
        userId: user.id,
        serverId: interaction.guildId!
    }, {
        authenticationEnabled: false
    });

    return interaction.reply(successEmbed(`2FA has been reset for
${user.toString()}`));

}
```

Recommendations

It is recommended to reset user's session upon resetting his/her 2FA configuration.

[M-04] Automatic session termination can be delayed

Severity

Impact: Low

Likelihood: Medium

Description

The bot implements cron task to manage users' sessions. The cron task is scheduled to be called every second. Within the task the runtime attempts to execute heavily time-consuming operation. Firstly, it fetches all users in the database. Then it tries to determine whether the session is stale for every registered server and update it. Finally, it attempts to post logs on the logging channel. It was observed that single run can last around 40-45 seconds. Thus, the final time of session termination can be longer than defined by the sever's owner.

File: `check-session.ts`

```
...
export const crons = [
    '0 * * * * *'
];

export const run = async () => {

    const activeUserSessions = await
Postgres.getRepository(UserSession).find({
        where: {
            isEnded: false
        }
    });

    if (activeUserSessions) {


        activeUserSessions.forEach(async (session) => {

            const serverSettings = await
Postgres.getRepository(ServerSettings).findOne({
                where: {
                    serverId: session.serverId
                }
            });
            const user = await client.users.fetch(session.userId);

            const timeout = serverSettings?.authTimeout || 300000;
            if (session.sessionStartedAt.getTime() + timeout < Date.now())
{
                session.isEnded = true;
                await Postgres.getRepository(UserSession).save(session);

                const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
                    where: {
                        serverId: session.serverId
                    }
                });

                const member = await
client.guilds.fetch(session.serverId).then((guild) =>
guild.members.fetch(session.userId));
```

```
                updateAuthentication(member,
whitelistedChannels.map((channel) => channel.channelId), false);

                sendLogs(client, serverSettings!, `**${user.tag}** -
permissions removed after ${Math.round(timeout / (60 * 1000))} mins.`);
            } else {
                console.log(`Session for ${user.tag} is still active
(${session.sessionStartedAt.getTime() + timeout - Date.now()} ms
remaining)`);
            }
        });

    }

};
```

Recommendations

It is recommended to consider implementing additional check for session invalidation upon calling any command available for the user.

[M-05] Permissions remain granted after channel removal

Severity

Impact: High

Likelihood: Low

Description

The bot allows server's owner grant users access to Discord's channels with two different commands: `set-channel` and `set-user-channel`.

- The `set-channel` allows all registered users access channels from this list. This configuration is saved in the `WhitelistedChannel` table.
- The `set-user-channel` allows selected registered users access selected channels from this list. This configuration is saved in the `UserWhitelistedChannel` table.

Whenever registered user authorise him/herself in the bot application, the `updateAuthentication` function is executed to grant permissions to user.

File: `index.ts`

```
...
            const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
```

```
                where: {
                    serverId: interaction.guildId!
                }
            });

            const userWhitelistedChannels = await
Postgres.getRepository(UserWhitelistedChannel).find({
                where: {
                    userId: interaction.user.id,
                    serverId: interaction.guildId!
                }
            });
...
            updateAuthentication(interaction.member as GuildMember,
                userWhitelistedChannels.length
                    ? userWhitelistedChannels.map((channel) =>
channel.channelId)
                    : whitelistedChannels.map((channel) =>
channel.channelId)
            , true);
```

Within the `updateAuthentication` function the user is granted following permissions:

- ViewChannel
- MentionEveryone
- EmbedLinks
- AttachFiles
- ReadMessageHistory
- SendMessages

File: `util.ts`

```
export const updateAuthentication = async (
  member: GuildMember,
  channelIds: string[],
  enabled: boolean
) => {
  const promises: Promise<void | NonThreadGuildBasedChannel>[] = [];

  channelIds.forEach((channelId) => {
    promises.push(
      member.guild.channels
        .fetch(channelId)
        .then((channel) => {
          if (enabled) {
            return (channel as TextChannel).permissionOverwrites
              .create(member, {
                ViewChannel: true,
                MentionEveryone: true,
                EmbedLinks: true,
                AttachFiles: true,
```

```
                ReadMessageHistory: true,
                SendMessages: true,
              })
              .finally(() => {
                console.log(
                  `Permissions granted for ${member.user.tag} in ${
                    channel!.name
                  }`
                );
              })
              .catch((err) => {
                console.error(err);
              });
          } else {
            return (channel as
TextChannel).permissionOverwrites.delete(member);
          }
        })
        .catch((err) => {
          console.error(err);
        })
    );
  });

  return Promise.allSettled(promises);
};
```

These permissions are meant to be granted timely, and after session timeout they are supposed to be revoked. There are two possibilities to revoke privileges. Firstly, by means of the `unauthenticate` command.

File: `unauthenticate.ts`

```
...
    const member = await interaction.guild!.members.fetch(user.id);

    const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
        where: {
            serverId: interaction.guildId!
        }
    });
    updateAuthentication(member, whitelistedChannels.map(c =>
c.channelId), false);

    return interaction.reply(successEmbed(`User ${user.toString()} has
been unauthenticated!`));
}
```

Secondly, by the cron's task `check-session`.

File: `check-session.ts`

```
...
                const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
                    where: {
                        serverId: session.serverId
                    }
                });

                const member = await
client.guilds.fetch(session.serverId).then((guild) =>
guild.members.fetch(session.userId));
                updateAuthentication(member,
whitelistedChannels.map((channel) => channel.channelId), false);
...
```

However, the bot revokes permissions only for channels stored in the database. If the server's owner remove the channel by means of the `set-channel` function, when user has active session, the user will remain with granted permissions permanently.

Recommendations

It is recommended to store information about granted permissions per channels in separate table which is independent from the channels configured in `WhitelistedChannel` and `UserWhitelistedChannel` channels. Upon session timeout the corresponding record must be removed then and permissions must be revoked.

[M-06] Inconsistent application of session check

Severity

Impact: Medium

Likelihood: Medium

Description

The server's owner has two command available to control the users session and 2FA authorisation: `reset-2fa` and `unauthenticate`. To execute `reset-2fa` owner or admin must have valid session in prior.

File: `reset-2fa.ts`

```
export const run: SlashCommandRunFunction = async (interaction) => {
  const owner =
    (interaction.member as GuildMember).guild.ownerId ===
interaction.user.id;
  const admin = await isAdmin(interaction.user.id, interaction.guildId!);
```

```
  if (!owner && !admin)
    return void interaction.reply(
      errorEmbed("You must be the server owner or an admin to use this
command")
    );

  if (!owner && admin) {
    const userSession = await
Postgres.getRepository(UserSession).findOne({
      where: {
        userId: interaction.user.id,
        serverId: interaction.guildId!,
        isEnded: false,
      },
    });

    if (!userSession) {
      return requestAuth(interaction);
    }
  }

  const user = (
    interaction.options as CommandInteractionOptionResolver
  ).getUser("user", true);

  await Postgres.getRepository(UserAuthentication).update(
    {
      userId: user.id,
      serverId: interaction.guildId!,
    },
    {
      authenticationEnabled: false,
    }
  );

  return interaction.reply(
    successEmbed(`2FA has been reset for ${user.toString()}`)
  );
};
```

However, to call the `unauthenticate` command owner does not have to have valid session.

File: `unauthenticate.ts`

```
    const isOwner = (interaction.member as GuildMember).guild.ownerId ===
interaction.user.id;
    if (!isOwner) return void interaction.reply(errorEmbed('You must be
the server owner to use this command'));

    const user = (interaction.options as
CommandInteractionOptionResolver).getUser('user', true);
```

```
    await Postgres.getRepository(UserSession).update({
        userId: user.id,
        serverId: interaction.guildId!
    }, {
        isEnded: true
    });

    const member = await interaction.guild!.members.fetch(user.id);

    const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
        where: {
            serverId: interaction.guildId!
        }
    });
    updateAuthentication(member, whitelistedChannels.map(c =>
c.channelId), false);

    return interaction.reply(successEmbed(`User ${user.toString()} has
been unauthenticated!`));
}
```

This inconsistency in the session check application between these two essential commands raise security concerns. Either the session should be checked or skipped in both cases.

Recommendations

It is recommended to review the implementation and consider whether session check is applied correctly for aforementioned functions. Skipping of session check can speed up commands execution in the event of the emergency. Checking the session status even of owner user can be beneficial, assuming owner account can be compromised as well.

[M-07] User's secret is exposed to the third party service

Severity

Impact: Medium

Likelihood: Medium

Description

The bot application requires from a new user to register new record in the selected 2FA mobile application (such as Google Authentication) to be capable of generating one-time-passwords (OTP). To achieve this, the bot generates QR code for user to scan it with mobile phone. The QR code image is generated in the third party service: https://api.qrserver.com/v1/create-qr-code/. As input parameter, the `generatedHash` is forwarded among the others. This property holds a per-user generated secret, required for time-based

one-time (TOTP) passcode generation and verification. This property is meant only to be known by the client and server as it does use symmetric cryptography. Thus, every secret generated can be considered as compromised.

Additionally it is uncertain who is the owner of the third party service and whether the server-side logs are stored with requestor's data.

File: `authenticate.ts`

```
...
    const otp = generateOTP(generatedHash, interaction.user.username);

    const qrCode = `https://api.qrserver.com/v1/create-qr-code/?
size=150x150&data=${otp.toString()}`;
...
```

File: `totp.ts`

```
import { totp, generateKey, getKeyUri, importKey } from "otp-io";
import { hmac } from "otp-io/crypto";

export function generateOTP(secret: string, username: string) {
  return getKeyUri({
    type: "totp",
    secret: importKey(secret),
    name: username,
    issuer: "Drew Security",
  });
}

export const getCurrentOTP = (secret: string) =>
  totp(hmac, { secret: importKey(secret) });
```

Recommendations

It is recommended to either remove the usage of third party service from the implementation and generate QR code images server-side.

[L-01] User's username is exposed to the third party service

Severity

Impact: Medium

Likelihood: Low

Description

The bot application requires from a new user to register new record in the selected 2FA mobile application (such as Google Authentication) to be capable of generating one-time-passwords (OTP). To achieve this, the bot generates QR code for user to scan it with mobile phone. The QR code image is generated in the third party service: https://api.qrserver.com/v1/create-qr-code/. As input parameter, the `interaction.user.username` is forwarded among the others. The username can be considered sensitive due to several reasons:

- It may contain personally identifiable information (PII), such as first name and last name.
- It may be used to de-anonymise user while attempting to identify similar usernames in other social media services using various OSINT techniques.
- It may be a consider as protected under the General Data Protection Regulation (GDPR) regulation and disclosure can be considered as violation.

Additionally it is uncertain who is the owner of the third party service and whether the server-side logs are stored with requestor's data.

File: `authenticate.ts`

```
...
    const otp = generateOTP(generatedHash, interaction.user.username);

    const qrCode = `https://api.qrserver.com/v1/create-qr-code/?
size=150x150&data=${otp.toString()}`;
...
```

Recommendations

It is recommended to either remove the usage of third party service from the implementation and generate QR code images server-side or get rid of `interaction.user.username` usage and use `interaction.user.id` instead.

[L-02] The authentication timeout lacks upper band input validation

Severity

Impact: Medium

Likelihood: Low

Description

After successful registration, a user can authenticate in the bot application by means of one-time-passwords (OTP). By default, the session lasts 5 minutes. The server's owner can update this property, setting at least one minute or more. However, there is no upper band check implemented for this command.

Thus, the server's owner can set any arbitrary value, even extremely high, which may cause the session practically never invalidate.

File: `set-auth-timeout.ts`

```
...
    const timeout = (interaction.options as
CommandInteractionOptionResolver).getInteger('timeout', true);

    if (timeout < 1) return void interaction.reply(errorEmbed('The timeout
must be at least 1 minute'));

    const serverSettings = await
getOrCreateServerSettings(interaction.guildId!);

    serverSettings.authTimeout = timeout * 60 * 1000;
    await Postgres.getRepository(ServerSettings).save(serverSettings);

    return interaction.reply(successEmbed(`The authentication timeout has
been set to ${timeout} minutes`));

}
```

Recommendations

It is recommended to implement reasonable upper band check that prevents the owner setting irrationally long session timeout.

[L-03] Inaccurate information can be presented by show-list command

Severity

Impact: Low

Likelihood: Low

Description

After successful registration, a user can authenticate in the bot application by means of one-time-passwords (OTP). By default, the session lasts 5 minutes. The server's owner can update this property, setting at least one minute or more. Also, the owner can display the list of authenticated users by means of the `show-list` command along with the remaining time of their session. However, this command displays accurate information only when the session timeout is set to default value: 5 minutes. This weakness exists because the algorithm uses fixed value of `1000 * 60 * 5` milliseconds as reference to the time that passed since session was started. Eventually, in the event of the non-default timeout value, it will display overestimated or senseless value.

File: show-list.ts

```
...
    const generateEntry = (entry: UserSession) => {
        const remainingTime = Math.floor(1000 * 60 * 5 - (Date.now() -
entry.sessionStartedAt.getTime()));
        return `<@${entry.userId}> - ${humanizeDuration(remainingTime, {
round: true })} left\n`;
    }

    const embeds = await generateEmbeds({
        entries: authenticatedUsers,
        generateEmbed,
        generateEntry
    });
...
```

Recommendations

It is recommended to fix the implementation so it uses the reference value saved in the bot's database instead of hard-coded value.

[L-04] The application availability depends on the third party API

Severity

Impact: Medium

Likelihood: Low

Description

The bot application requires from a new user to register new record in the selected 2FA mobile application (such as Google Authentication) to be capable of generating one-time-passwords (OTP). To achieve this, the bot generates QR code for user to scan it with mobile phone. The QR code image is generated in the third party service: https://api.qrserver.com/v1/create-qr-code/. This dependency is under third party control and the server's condition and stability is unknown. Any disturbance in the service availability will impact the bot application, as it will be not possible to fetch QR codes and register new users.

File: authenticate.ts

```
...
    const otp = generateOTP(generatedHash, interaction.user.username);

    const qrCode = `https://api.qrserver.com/v1/create-qr-code/?
```

```
size=150x150&data=${otp.toString()}`;
...
```

Recommendations

It is recommended to either remove the usage of third party service from the implementation and generate QR code images server-side.

[I-01] Users metadata are stored under third party control

The bot makes use of the Postgres database to store data about channels, users and sessions. The database is not accessible from the bot's UI and it is under third party control. It is unknown whether the stored data are being used for any analytical processing. To provide the service, the application saves following metadata about Discord's servers, channels and users in the database:

- serverId
- userId
- channelId

These properties can be considered as pseudo-anonymised. The finding was reported to drag the Client's team attention.

Recommendations

It is recommended to consider deployment of the private instance of the bot application with private database instance.

[I-02] Undocumented behavior of some commands

The bot allows server's owner grant users access to Discord's channels with two different commands: `set-channel` and `set-user-channel`.

- The `set-channel` allows all registered users access channels from this list. This configuration is saved in the `WhitelistedChannel` table.
- The `set-user-channel` allows selected registered users access selected channels from this list. This configuration is saved in the `UserWhitelistedChannel` table.

However, these settings are mutually exclusive, where `set-user-channel` is superior over `set-channel`. E.g. whenever user has record in the `UserWhitelistedChannel` table, he/she will not be granted permissions to channels saved in the `WhitelistedChannel`. On the other hand, the bot's user may expect that registered user will be granted permissions to channels from both settings.

File: `index.ts`

```
...
            const whitelistedChannels = await
Postgres.getRepository(WhitelistedChannel).find({
                where: {
                    serverId: interaction.guildId!
                }
            });

            const userWhitelistedChannels = await
Postgres.getRepository(UserWhitelistedChannel).find({
                where: {
                    userId: interaction.user.id,
                    serverId: interaction.guildId!
                }
            });
...
            updateAuthentication(interaction.member as GuildMember,
                userWhitelistedChannels.length
                    ? userWhitelistedChannels.map((channel) =>
channel.channelId)
                    : whitelistedChannels.map((channel) =>
channel.channelId)
            , true);
```

Recommendations

It is recommended to document the behavior accurately, so bot's users are not confused. Alternatively, it is recommended to change the implementation so registered user is granted permissions for channels defined in both tables.

[I-03] Lack of opt-out and permanent data wipe functionality

The bot makes use of the Postgres database to store data about channels, users and sessions. To provide the service, the application saves following metadata about Discord's servers, channels and users in the database:

- serverId
- userId
- channelId

Additionally, some relationships information are saved in database, such as who whitelisted particular user. However, there is no opt-out functionality implemented in the bot. Thus, all data is saved there permanently, until database's owner perform some manual maintenance.

Recommendations

It is recommended to implement an opt-out functionality, which wipes all data related to the particular Discord's server.

[I-04] Bot does not log lack of permissions

Within the `updateAuthentication` function the bot attempts to grant the user following permissions:

- ViewChannel
- MentionEveryone
- EmbedLinks
- AttachFiles
- ReadMessageHistory
- SendMessages

To fulfil this function, the bot must be granted `Manage Permissions` privilege for particular channel. If it is not done in advance, the `updateAuthentication` function will throw exception silently, without any notification.

File: `util.ts`

```
export const updateAuthentication = async (
  member: GuildMember,
  channelIds: string[],
  enabled: boolean
) => {
  const promises: Promise<void | NonThreadGuildBasedChannel>[] = [];

  channelIds.forEach((channelId) => {
    promises.push(
      member.guild.channels
        .fetch(channelId)
        .then((channel) => {
          if (enabled) {
            return (channel as TextChannel).permissionOverwrites
              .create(member, {
                ViewChannel: true,
                MentionEveryone: true,
                EmbedLinks: true,
                AttachFiles: true,
                ReadMessageHistory: true,
                SendMessages: true,
              })
              .finally(() => {
                console.log(
                  `Permissions granted for ${member.user.tag} in ${
                    channel!.name
                  }`
                );
              })
              .catch((err) => {
```

```
                console.error(err);
              });
          } else {
            return (channel as
TextChannel).permissionOverwrites.delete(member);
          }
        })
        .catch((err) => {
          console.error(err);
        })
    );
  });

  return Promise.allSettled(promises);
};
```

Recommendations

It is recommended to log information about unsuccessful permissions granting or revoking on the `logging-channel`, whenever the bot has insufficient permissions.

[I-05] The authenticatedRoleId column is not used

The `ServerSettings` entity has defined `authenticatedRoleId` property. However, this property is not used in the application.

File: `database.ts`

```
@Entity()
export class ServerSettings extends BaseEntity {
    @PrimaryGeneratedColumn()
    id!: number;
...
    Column({
        nullable: true
    })
    authenticatedRoleId!: string;
...
```

Recommendations

It is recommended to remove unused properties of the database's entities.

[I-06] Vulnerable dependencies in use

The solution uses dependencies which contains known vulnerabilities. The npm's audit results states that there are 27 vulnerabilities (6 low, 6 moderate, 11 high, 4 critical) in total.

```
npm audit
# npm audit report

@babel/traverse  <7.23.2
Severity: critical
Babel vulnerable to arbitrary code execution when compiling specifically
crafted malicious code - https://github.com/advisories/GHSA-67hx-6x53-jw92
fix available via `npm audit fix`
node_modules/@babel/traverse

@fastify/multipart  7.0.0 - 7.4.0
Severity: high
Denial of service due to unlimited number of parts -
https://github.com/advisories/GHSA-hpp2-2cr5-pf6g
fix available via `npm audit fix`
node_modules/@fastify/multipart

@fastify/session  <10.9.0
Severity: high
@fastify/session reuses destroyed session cookie -
https://github.com/advisories/GHSA-pj27-2xvp-4qxg
fix available via `npm audit fix --force`
Will install @adminjs/fastify@4.1.3, which is a breaking change
node_modules/@fastify/session
  @adminjs/fastify  >=3.0.0
  Depends on vulnerable versions of @fastify/cookie
  Depends on vulnerable versions of @fastify/session
  node_modules/@adminjs/fastify

axios  0.8.1 - 0.27.2
Severity: moderate
Axios Cross-Site Request Forgery Vulnerability -
https://github.com/advisories/GHSA-wf5p-g6vw-rhxx
fix available via `npm audit fix --force`
Will install adminjs@7.8.13, which is a breaking change
node_modules/axios
  adminjs  <=7.0.0-beta-v7.13
  Depends on vulnerable versions of axios
  Depends on vulnerable versions of flat
  node_modules/adminjs

class-validator  <0.14.0
Severity: critical
SQL Injection and Cross-site Scripting in class-validator -
https://github.com/advisories/GHSA-fj58-h2fr-3pp2
fix available via `npm audit fix --force`
Will install class-validator@0.14.1, which is a breaking change
node_modules/class-validator

cookie  <0.7.0
cookie accepts cookie name, path, and domain with out of bounds characters
- https://github.com/advisories/GHSA-pxg6-pf52-xh8x
```

```
fix available via `npm audit fix --force`
Will install @adminjs/fastify@4.1.3, which is a breaking change
node_modules/@sentry/node/node_modules/cookie
node_modules/cookie
  @fastify/cookie  7.1.0 - 9.1.0
  Depends on vulnerable versions of cookie
  node_modules/@fastify/cookie
  @sentry/node  4.0.0-beta.0 - 7.74.2-alpha.1
  Depends on vulnerable versions of cookie
  node_modules/@sentry/node
  light-my-request  3.7.0 - 5.13.0 || 6.0.0-pre.fv5.1 - 6.0.0
  Depends on vulnerable versions of cookie
  node_modules/light-my-request

fastify  4.0.0-alpha.1 - 4.25.2
Severity: high
Fastify: Incorrect Content-Type parsing can lead to CSRF attack  -
https://github.com/advisories/GHSA-3fjj-p79j-c9hh
Depends on vulnerable versions of find-my-way
fix available via `npm audit fix`
node_modules/fastify

find-my-way  5.5.0 - 8.2.1
Severity: high
find-my-way has a ReDoS vulnerability in multiparametric routes -
https://github.com/advisories/GHSA-rrr8-f88r-h8q6
fix available via `npm audit fix`
node_modules/find-my-way

flat  <5.0.1
Severity: critical
flat vulnerable to Prototype Pollution -
https://github.com/advisories/GHSA-2j2x-2gpw-g8fm
fix available via `npm audit fix --force`
Will install adminjs@7.8.13, which is a breaking change
node_modules/flat

follow-redirects  <=1.15.5
Severity: moderate
Follow Redirects improperly handles URLs in the url.parse() function -
https://github.com/advisories/GHSA-jchw-25xp-jwwc
follow-redirects' Proxy-Authorization header kept across hosts -
https://github.com/advisories/GHSA-cxjh-pqwp-8mfp
fix available via `npm audit fix`
node_modules/follow-redirects

json5  2.0.0 - 2.2.1
Severity: high
Prototype Pollution in JSON5 via Parse Method -
https://github.com/advisories/GHSA-9c47-m6qq-7p4h
fix available via `npm audit fix`
node_modules/json5

luxon  1.0.0 - 1.28.0
```

```
Severity: high
Luxon Inefficient Regular Expression Complexity vulnerability -
https://github.com/advisories/GHSA-3xq5-wjfh-ppjc
fix available via `npm audit fix`
node_modules/luxon

rollup  <2.79.2
Severity: high
DOM Clobbering Gadget found in rollup bundled scripts that leads to XSS -
https://github.com/advisories/GHSA-gcx4-mw62-g8wm
fix available via `npm audit fix`
node_modules/rollup

semver  <=5.7.1 || 6.0.0 - 6.3.0 || 7.0.0 - 7.5.1
Severity: high
semver vulnerable to Regular Expression Denial of Service -
https://github.com/advisories/GHSA-c2qf-rxjj-qqgw
semver vulnerable to Regular Expression Denial of Service -
https://github.com/advisories/GHSA-c2qf-rxjj-qqgw
semver vulnerable to Regular Expression Denial of Service -
https://github.com/advisories/GHSA-c2qf-rxjj-qqgw
fix available via `npm audit fix`
node_modules/fastify/node_modules/semver
node_modules/make-dir/node_modules/semver
node_modules/semver

send  <0.19.0
Severity: moderate
send vulnerable to template injection that can lead to XSS -
https://github.com/advisories/GHSA-m6fv-jmcg-4jfg
fix available via `npm audit fix`
node_modules/send
  @fastify/static  <=6.6.0
  Depends on vulnerable versions of send
  node_modules/@fastify/static

undici  <=5.28.3
Undici's Proxy-Authorization header not cleared on cross-origin redirect
for dispatch, request, stream, pipeline -
https://github.com/advisories/GHSA-m4v8-wqvr-p9f7
Undici's fetch with integrity option is too lax when algorithm is
specified but hash value is in incorrect -
https://github.com/advisories/GHSA-9qxr-qj54-h672
Undici proxy-authorization header not cleared on cross-origin redirect in
fetch - https://github.com/advisories/GHSA-3787-6prv-h9w3
fix available via `npm audit fix`
node_modules/undici
  @discordjs/rest  2.0.1-dev.1690848847-1af7e5a0b.0 - 2.3.0-
dev.1714824213-96169add6
  Depends on vulnerable versions of undici
  node_modules/@discordjs/rest
  discord.js  14.0.0-dev.1640779371.9cdc448 - 14.0.0-dev.1657757514-
fe34f48 || 14.12.2-dev.1690891477-7295a3a94.0 - 14.15.0-dev.1714824206-
96169add6
```

```
    Depends on vulnerable versions of undici
    Depends on vulnerable versions of ws
    node_modules/discord.js

  ws  8.0.0 - 8.17.0
  Severity: high
  ws affected by a DoS when handling a request with many HTTP headers -
  https://github.com/advisories/GHSA-3h5v-q93c-6h6q
  fix available via `npm audit fix`
  node_modules/@discordjs/ws/node_modules/ws
  node_modules/ws

  xml2js  <0.5.0
  Severity: moderate
  xml2js is vulnerable to prototype pollution -
  https://github.com/advisories/GHSA-776f-qx25-q3cc
  fix available via `npm audit fix`
  node_modules/xml2js
    typeorm  0.1.0-alpha.1 - 0.3.14-dev.daf1b47 || >=0.3.21-dev.28a8383
    Depends on vulnerable versions of xml2js
    node_modules/typeorm

  27 vulnerabilities (6 low, 6 moderate, 11 high, 4 critical)
```

Recommendations

It is recommended to keep dependencies up to date and use alternative software which is free from known vulnerabilities.

[I-07] DoS is possible via database records flood

The bot makes use of the Postgres database to store data about channels, users and sessions. Additionally, some relationships information are saved in database, such as who whitelisted particular user. However, there is no rate-limiting functionality implemented. Thus, any Discord's user can create as many servers as possible, add bot to it, and then configure multiple admins, configure channels and whitelist users to flood the database with records. Then, at the certain point of time the database's storage can get full and it will stop processing subsequent `insert` requests.

Recommendations

It is recommended to implement rate-limiting per Discord's server to prevent over-utilisation of the database's storage.