Fallible
hello@fallible.co

Code-Audit-Report of Ente's *museum* GitHub Repository 19-04-2023

Introduction

The Ente team contacted us for the audit of their backend server hosted at https://github.com/ente-io/museum. The access was granted to a senior researcher at our firm using a secure end-to-end encrypted communication tool. The repository at the commit hash `5140fbe72d0be64c3efedf556d54723d03d0c10c` had approximately 21k lines of code.

The audit was conducted during the period of 15th February to 15th April with manual code audits as the main focus but the tools at disposal were not limited and the researchers were free to use static and dynamic analysis tools as well. To make the audit more sensitive, a live account was created and used as well to verify certain possibilities that may arise due to the vulnerability in the code. Severe issues, if suspected, were quickly discussed over a secure channel with the Ente team to conclude if it needed more attention or not. The entire review was conducted in 46 hours.

The report below details the findings in three main segments Security Vulnerabilities (**SV**), Security Code Smells (**SCS**) and Security Recommendations (**SR**).

Scope

- **Backend code audits**
    - **SV:** Identifying existing security vulnerabilities in the backend server code hosted at https://github.com/ente-io/museum
    - **SCS:** Identifying security code smells in the same repository since proactive programming helps reduce cyber security incidents.
    - **SR:** Security recommendations for maintaining an open source project specific to this project.

Security Vulnerabilities

SV-ENTE-001 - Using `math/random` instead of `crypto/random` for generating OTP and referral code. (Medium)

Note: *In the latest commit, we can see the math/random has already been replaced with crypto/random by the ente team by identifying the issue themselves.*

Using `math/random` instead of `crypto/random` to generate one-time passwords (OTP) and referral codes is a bad practice because `math/random` generates pseudo-random numbers that are not cryptographically secure. This means that the numbers generated are not truly random and can be predicted or manipulated, making them vulnerable to attacks.

An attacker can easily guess or brute-force the OTP or referral code if `math/random` is used. In contrast, `crypto/random` generates cryptographically secure random bytes that are unpredictable and uniformly distributed, making them much harder to guess or manipulate.

Someone can identify the use of `math/random` in a GitHub repository by reviewing the code and looking for instances where random numbers are generated. The code can then be audited to determine whether `math/random` or `crypto/random` is used. Alternatively, one can use automated code analysis tools or security scanners to detect potential security vulnerabilities in the code.

SV-ENTE-002 - Use of `COPY . .` command in Dockerfile (Low)

When you use the "`COPY . .`" command in a Dockerfile, you are copying the entire context of the build directory into the container image. This includes all files and directories in the directory where the Dockerfile is located. This can potentially include sensitive information, such as configuration files, credentials, and other secrets.

So, it is recommended to either copy the exact files and directories that are needed or explicitly ignore sensitive files via `.dockerignore` file.

SV-ENTE-003 - Use of `USER` command was missing in Dockerfile (Low)

The **`USER`** instruction in a Dockerfile allows you to specify the user context under which the commands in the Dockerfile and the container itself will run. Specifying a user context is important from a security perspective, because it helps to reduce the potential impact of any security vulnerabilities or exploits in the container.

Here are some of the reasons why using a **`USER`** instruction is important for security in Docker:

1. Privilege escalation: By default, Docker containers run as root, which gives them access to all resources on the host system. If a container is compromised, an attacker could potentially use this access to escalate their privileges and gain control of the host system. By specifying a non-root user with limited permissions, you can reduce the impact of any such attacks.
2. File system permissions: When files are created inside a container, their ownership and permissions are determined by the user context under which the container is running. If the container is running as root, any files that are created will be owned by root and may have overly permissive permissions. By specifying a non-root user with appropriate permissions, you can ensure that files are created with the correct ownership and permissions.
3. Compliance requirements: Some security compliance frameworks, such as CIS Docker Benchmark, require that containers run as non-root users to meet security best practices.

To use a **`USER`** instruction in your Dockerfile, you should first create a user and set appropriate permissions for any files and directories that the container will access. You can then use the **`USER`** instruction to switch to this non-root user in your Dockerfile.
For example, here is an example Dockerfile that uses a non-root user:

```
FROM ubuntu

# create a non-root user with limited permissions
RUN useradd --create-home myuser
WORKDIR /home/myuser

# set appropriate permissions on any files and directories
RUN chown -R myuser:myuser /home/myuser

# switch to the non-root user context


USER myuser

# run any commands under the non-root user context
CMD [ "echo", "Hello, world!" ]
```

By using a non-root user in your Dockerfile, you can help to reduce the impact of security vulnerabilities and exploits in your container, and meet security compliance requirements.

See

- MITRE, CWE-284 - **Improper Access Control**
- **nginxinc/nginx-unprivileged: Example of a non-root container by default**
- **Microsoft docs, When to use ContainerAdmin and ContainerUser user accounts**

Security Code Smells

SCS-ENTE-001 - Access-Control-Allow-Origin (ACAO) set to * (Info)

Path: /museum/cmd/museum/main.go

Access-Control-Allow-Origin (ACAO) is a header that is used in HTTP responses to indicate which origins are allowed to access the resources of a web page. The "*" value for the ACAO header means that any origin is allowed to access the resource, which can potentially cause security issues.

The main security risk associated with setting ACAO to "*" is that it can allow cross-site scripting (XSS) attacks. XSS attacks occur when an attacker injects malicious code into a web page, which is then executed by unsuspecting users who access the page. By setting ACAO to "*", an attacker can potentially execute code on any website that accesses the resource, regardless of whether it is intended to do so or not.

To mitigate this risk, it is recommended that you set the ACAO header to the specific origin(s) that are allowed to access the resource. This can be done by specifying the origin(s) in the header value, like this:

```
c.Writer.Header().Set("Access-Control-Allow-Origin",
"https://example.com")
```

Although no other cookies than CloudFlare load balancer cookies exist at the moment, it is recommended to avoid using Access Control Allow Origin with * to prevent any potential security risks.

SCS-ENTE-002 - Sensitive Tokens are passed in query params, which can be logged. (Info)

File: auth.go

Example:
```
func GetToken(c *gin.Context) string {
   token := c.GetHeader("X-Auth-Token")
   if token == "" {
       token = c.Query("token")
   }
   return token
}
```

One of the main risks of passing sensitive tokens in query parameters is that they can be easily logged by various components along the request/response chain. This includes web servers, proxies, and even client-side tools such as web browsers. If the token is logged, it could potentially be exposed to unauthorized third parties, putting the security of the system and the data it protects at risk.

SCS-ENTE-003 - Character Injection attack possibility. (Info)

Example:
```
   1. filePath := c.tempStorage + "/" + fileName
```

Or

```
   2. fmt.Sprintf("%s:%s:%s", app, token, *jwtClaimScope)
      fmt.Sprintf("%s:%s", app, token)
```

Since `app` and `token` can be manipulated by the user, there is a real chance of the user passing a string like “aa:123” which contains a colon as part of the `token` there by injecting the third value from their end. However we found that this will not occur in this particular case as the token in these two statements differ significantly, one is string while the other being a json string, so such an injection is not possible here. But we suggest not to encourage certain string manipulation techniques (plain concatenation or key creation with colon as a separator where users can control the values) as in future this might pose a real problem.

SCS-ENTE-04 - Email template not specifying possibility of a 3rd party trigger (Info)

There is a possibility that any 3rd party can trigger any sign in OTP emails to anyone if they know the email address. And since this is unavoidable due the nature of the activity taking place right before authorization. Usually the prudent practice is to simply write in the email (sent for OTP) that they can ignore the email if not triggered by them. This will avoid the usual panic that these sorts of emails cause.

Security Recommendations

SR-ENTE-001 - Clear note on how passwords are handled. (Info)

Some fake passwords are hardcoded but it was informed that production.yaml is used for real passwords and are not publicly uploaded to *GitHub*. It is used directly in the server machine through a secure channel. GitHub users might think that those fake passwords are real ones and that is what they need to change leading to a bad security practice. There was no document documenting this practice of fetching password from a secure endpoint, so it is recommended to make a clear note of how passwords and sensitive api keys are handled in **README** or any other supporting doc.

SR-ENTE-002 - Delete the commit history before making the repo public. (Info)

Sensitive third party Api key was found in the commit history, so it is recommended to go with a fresh repo and paste the contents in that repo. Otherwise GitHub has this document which can be followed to delete the history https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository. It is also recommended to set up a scanner which scans for sensitive data in the repository on every push.

Notes

Some test passwords for databases and Minio were found in the repository. It is requested to verify that they are not for any live databases.

Private vulnerability reporting, together with the rest of GitHub's security capabilities like Dependabot, code scanning, and secret scanning, which is free for public repositories, will also be very helpful for the security of the repository going forward.

Conclusion

It is good to see that the Ente team has taken steps to address the issue of using `math/random` instead of `crypto/random` to generate one-time passwords and referral codes.

It's also worth noting that the use of "COPY . ." in a Dockerfile and the lack of a USER command were identified as potential security issues. While these are lower priority issues, it's still important to address them to maintain the overall security of the system.

Finally, the use of Access-Control-Allow-Origin (ACAO) set to "*" and the passing of sensitive tokens in query params were identified as potential security risks. It's important to mitigate these risks by setting ACAO to specific origins and avoiding passing sensitive tokens in query params.

The audit did not reveal any critical, high, or medium-level (unaddressed) security issues. This is a positive outcome as it means that the application has been developed with security in mind, and any potential vulnerabilities have been addressed before deployment. However, it's still important to remain vigilant and continue to conduct regular security scans to ensure that the application remains secure over time. With the absence of critical, high, or medium-level issues, the team can now focus on addressing any lower-level security issues and continuing to enhance the overall security posture of the application.