Beep security report

PONTHIEU Hugo, TCHILINGUIRIAN Théo.

What is Beep?

Beep is a discord-like web application that allows users to communicate in real-time through text, voice, and video.

Beep is a discord like application. It as a Node.js backend and a React frontend. All of the code is containerized using Docker. The images are built, then pushed inside a Docker registry.

Next, the application is deployed on a Kubernetes cluster. Each time the code is updated, a new image is built and pushed to the registry. In the same time an helm chart is updated with the new image tag. And an ArgoCD instance monitor this repo and redeploy the application when the tag changes.

For state management, Beep uses: - Redis - PostgreSQL - Minio

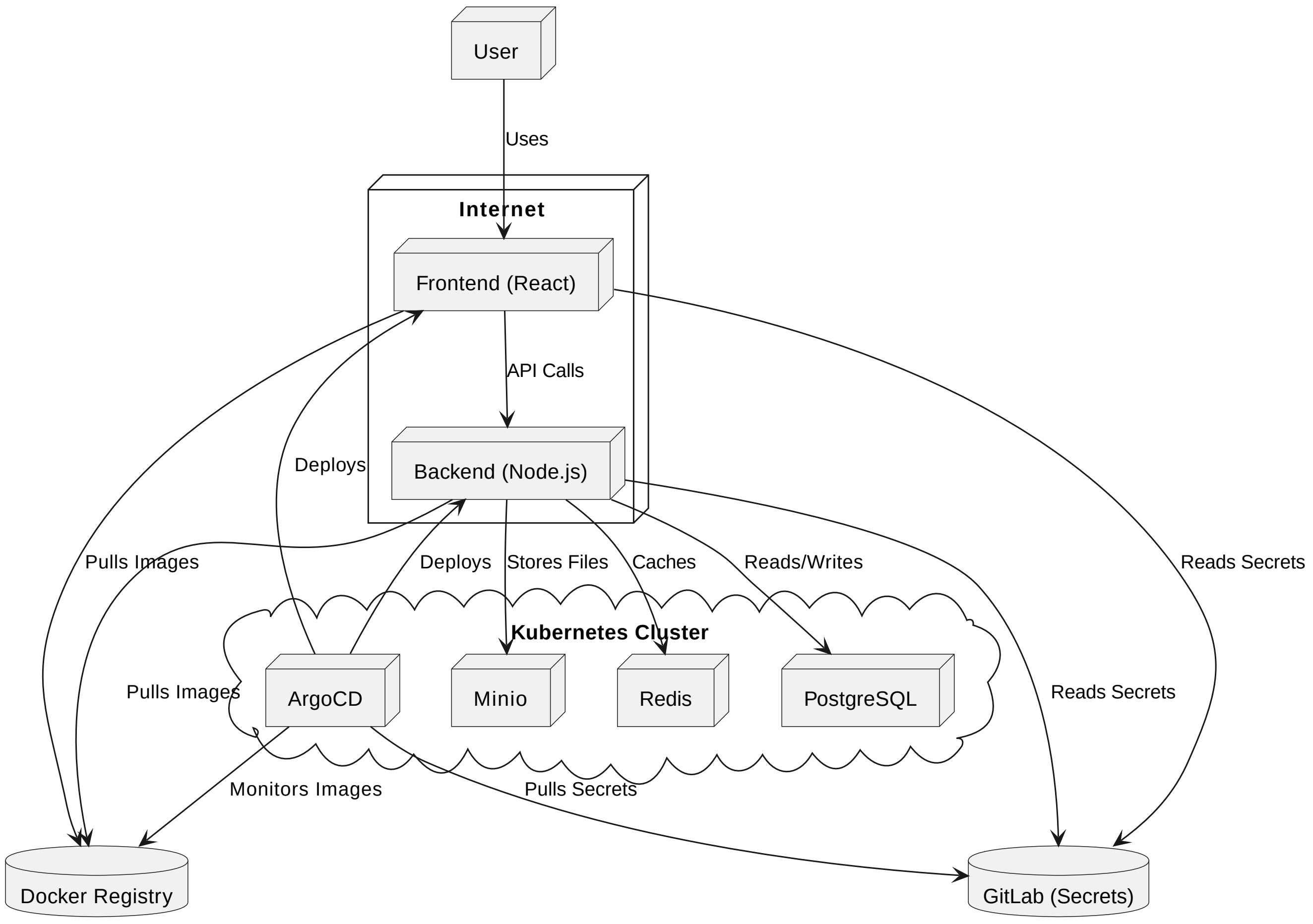
Those are deployed on the same Kubernetes cluster in the same network zone. This is deployed using IAC tools (Terraform). It deploys the services with open source helm charts. Then we use Helm terraform provider to deploy the helm charts with the right values.

The secrets are stored in gitlab and are encrypted with sealed secrets. The secrets are deploy in the same time that the services.

The frontend is accessible on Internet, as well as the backend, with no DMZ or firewalls in the way, except for Polytech's managed gateway. The backend consumes the Redis and PostgreSQL services. The frontend consumes the backend API.

Project deployment diagram

The following diagram illustrates the deployment architecture of Beep, including its components and their interactions:



Threats & security risk assessment

This report provides a comprehensive security risk assessment of the Beep platform, focusing on identifying potential vulnerabilities and threats to its components. The assessment includes an analysis of the backend API, frontend application, and infrastructure managed through Terraform.

Using automated tools, we have conducted Static Application Security Testing (SAST) and Software Composition Analysis (SCA) to identify vulnerabilities in the codebase and its dependencies, as well as container images. Additionally, we have performed threat modeling to assess potential risks and recommend mitigations.

Static Application Security Testing (SAST) is a method of analyzing source code to identify security vulnerabilities without executing the program. It helps developers find and fix issues early in the development process. Using Snyk to conduct SAST, we have found several vulnerabilities in the Beep platform’s codebase, including issues related to authentication, access control, and data security.

Software Composition Analysis (SCA) is a process of identifying and managing open source components and their dependencies in software projects. It helps to analyze dependencies, but also to ensure that the software is secure and compliant with licensing requirements. Using Trivy, we have identified vulnerabilities in the Beep platform’s container images, including outdated dependencies and known security issues, including published Common Vulnerabilities and Exposures (CVEs) and misconfigurations.

What we’ve done

Currently, Beep does not have security risk assessment mechanisms in place. To address this issue, we have conducted a security risk assessment of three repositories and container images of Beep:

- The backend API’s repository and container image,
- The frontend’s repository and container image,
- The Terraform IaC repository.

Each repository has a GitLab CI for building images (frontend and backend) or deploying infrastructure (Terraform IaC).

We have implemented Static Application Security Testing (SAST) and Software Composition Analysis (SCA) in the CI/CD pipelines of these repositories. This allows us to automatically scan for vulnerabilities in the code and its dependencies, as well as the build container images. This allows for security issues to be identified earlier, in the development process, before they reach production.

We have used IriusRisk, Snyk, and Trivy to perform the security risk assessment of the Beep platform.

Conducting this has allowed us to identify potential threat vectors and Common Vulnerabilities and Exposures (CVEs). The assessment was performed using various tools, including IriusRisk for threat modeling and risk assessment, Snyk for static code analysis, and trivy for container image scanning. Dependencies were also analyzed for known vulnerabilities.

Summary of findings

Threat modeling and security risk assessment with IriusRisk

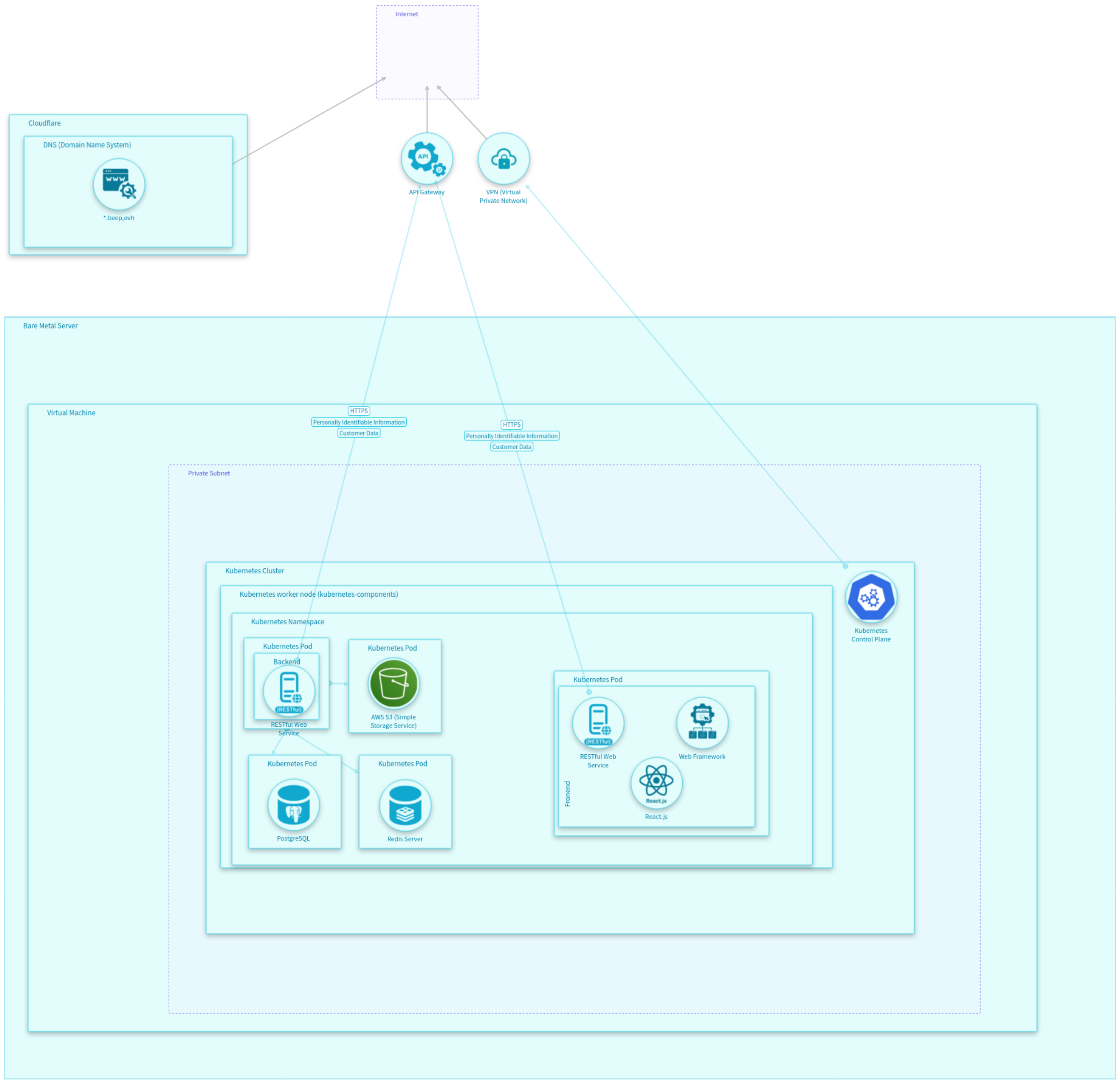
We used IriusRisk to build a model of our infrastructure. IriusRisk then generated a threat model based on the given architecture of the Beep platform, including potential threats, vulnerabilities, and mitigations for each component of the system.

Methodology

Our assessment followed a four-step approach: Asset Identification of critical systems and data; Threat Modeling using IriusRisk templates for web applications; Risk Evaluation based on likelihood and impact; and Countermeasure Planning for significant threats.

Key Findings

The threat modeling identified several high-priority concerns including unauthorized administrative access, DNS vulnerabilities (spoofing and unencrypted traffic), insufficient logging and monitoring, access control weaknesses, and infrastructure security gaps in Kubernetes deployment.



Critical Threats

Our assessment revealed several critical security threats to the Beep platform:

Authentication Issues: Unauthorized administrative access poses significant risk of configuration changes and data breaches. Inadequate RBAC implementation allows users to potentially access unauthorized resources.

Network Vulnerabilities: DNS spoofing could redirect users to malicious websites, while unencrypted DNS traffic exposes user browsing patterns. Insecure Kubernetes PKI key file permissions could enable unauthorized system access.

Operational Weaknesses: Insufficient logging prevents detection of malicious activities. Lack of rate limiting exposes services to denial-of-service attacks, particularly affecting Redis Server performance.

Data Security Concerns: Insecure file permissions on PostgreSQL databases and inadequate protection of secrets management increase risk of data exposure.

Potential Mitigations

Short-term Actions: Our immediate recommendations focus on critical security controls. Implementing multi-factor authentication for administrative interfaces will significantly reduce unauthorized access risk. Applying proper file permissions (600) to Kubernetes PKI keys is essential to prevent unauthorized system access. Enabling comprehensive logging across all critical services will help detect suspicious activities. Encrypting DNS traffic using DoH or DoT protocols will protect against DNS-based attacks and eavesdropping.

Medium-term Strategy: For sustained security improvement, we recommend deploying Role-Based Access Control throughout the application stack to enforce the principle of least privilege. Implementing rate limiting and resource throttling will protect services from denial-of-service attacks. Establishing secure secrets management workflows will safeguard sensitive credentials. Setting up real-time monitoring and alerting systems will enable rapid response to security incidents.

Long-term Approach: To build a security-focused culture, we suggest developing comprehensive security training for all developers to ensure secure coding practices. Establishing automated security testing in the CI/CD pipeline will catch vulnerabilities early in development. Creating detailed incident response procedures for various threat scenarios will improve recovery time. Implementing regular security assessments and penetration testing will continuously identify and address new vulnerabilities.

The implementation of these mitigations, prioritizing the highest-risk items first, will substantially improve Beep's security posture and reduce vulnerability to attacks.

You can find more details about the threat model in the `countermeasures/` directory.

SAST on codebase with Snyk

We have updated Beep's GitLab CI test stages to include SAST scans using Snyk. This allows us to automatically scan the codebase for vulnerabilities.

here is the updated GitLab CI configuration for SAST with Snyk:

```
code-test:
  stage: test
  image: snyk/snyk:docker
  variables:
    SNYK_TOKEN: $SNYK_TOKEN_HUGO
  tags:
    - beep-runner
  script:
    - cd $CI_PROJECT_DIR
    - ls
    - echo "Running Snyk tests..."
    - echo SNYK_TOKEN
    - snyk code test --json-file-output=./code-test.json --severity-threshold=medium
  artifacts:
    paths:
      - code-test.json
    when: always
  rules:
    - if: $CI_COMMIT_TAG
      when: never
    - if: $CI_COMMIT_REF_NAME == $CI_DEFAULT_BRANCH
    - if: $CI_PIPELINE_SOURCE == 'merge_request_event'

deps-scan:
  stage: test
  image: snyk/snyk:docker
  variables:
    SNYK_TOKEN: $SNYK_TOKEN_HUGO
  tags:
    - beep-runner
  script:
    - cd $CI_PROJECT_DIR
    - echo "Running Snyk tests..."
    - snyk test --json-file-output=./deps-scan.json
  artifacts:
    paths:
      - deps-scan.json
    when: always
  rules:
    - if: $CI_COMMIT_TAG
      when: never
    - if: $CI_COMMIT_REF_NAME == $CI_DEFAULT_BRANCH
    - if: $CI_PIPELINE_SOURCE == 'merge_request_event'
```

Here is a table summarizing the 10 most important vulnerabilities that were found in the codebase:

#	Issue Type	Severity	Score	File	Description
1	Hardcoded Secret (JWT sign)	error	807	apps/users/services/user_service.ts:101	Hardcoded cipher key used in `jsonwebtoken.sign`.
2	Hardcoded Secret (JWT verify)	error	807	apps/users/services/user_service.ts:113	Hardcoded cipher key used in `jsonwebtoken.verify`.

#	Issue Type	Severity	Score	File	Description
3	ServerLeak	warning	594	apps/authentication/exceptions/authentication_exception_handler.ts:6	Error object leaked via `send()`.
4	ServerLeak	warning	594	apps/authentication/exceptions/current_password_mismatch_exception.ts:6	Error object leaked via `send()`.
5	ServerLeak	warning	594	apps/channels/exceptions/channel_cant_be_children_exception.ts:6	Error object leaked via `send()`.
6	ServerLeak	warning	594	apps/channels/exceptions/channel_not_found_exception.ts:6	Error object leaked via `send()`.
7	ServerLeak	warning	594	apps/channels/exceptions/wrong_channel_type.ts:6	Error object leaked via `send()`.
8	ServerLeak	warning	594	apps/invitations/exceptions/unusable_invitation_exception.ts:6	Error object leaked via `send()`.

#	Issue Type	Severity	Score	File	Description
9	ServerLeak	warning	594	apps/invitations/exceptions/wrong_invitation_format_exception.ts:6	Error object leaked via `send()`.
10	ServerLeak	warning	594	apps/members/exceptions/member_not_in_server_exception.ts:6	Error object leaked via `send()`.

And the below table exposes the vulnerability that was found in the dependencies:

ID	Title	Severity	CVSS Score	Package (Version)	Exploit Status
SNYK-JS-LODASHSET-1320032	Prototype Pollution	High	7.3	lodash.set (4.3.2)	Proof of Concept

NOTE This vulnerability affects the `lodash.set` package and does not have a fixed version. It allows attackers to exploit prototype pollution, potentially leading to denial of service, remote code execution, or unauthorized privilege escalation.

You can find more details about the threat model in the `sast/` directory.

SCA on container images with Trivy

We have updated Beep's GitLab CI test stages to include SCA scans using Trivy. This allows us to automatically scan container images for vulnerabilities during the build process.

Here is the updated GitLab CI configuration for SCA with Trivy:

```
container-scan:
  stage: test
  image:
    name: docker.io/aquasec/trivy:latest
    entrypoint: [""]
  variables:
    # No need for pre-pulling the image as it comes from the build stage
    TRIVY_NO_PROGRESS: "true"
    TRIVY_CACHE_DIR: .trivycache/
    TRIVY_SEVERITY: HIGH,CRITICAL
  tags:
    - beep-runner
  script:
    - trivy image --format json --output trivy-image-report.json
$CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
  artifacts:
    paths:
      - trivy-image-report.json
    when: always
  rules:
    - if: $CI_COMMIT_TAG
      when: never
    - if: $CI_COMMIT_REF_NAME == $CI_DEFAULT_BRANCH
    - if: $CI_PIPELINE_SOURCE == 'merge_request_event'
```

Here is a table summarizing the 10 most important vulnerabilities that were found in the container image:

CVE ID	Title	Severity	CVSS Score	Package (Version)	Description
CVE-2024-6119	Possible denial of service in X.509 name checks	HIGH	7.5	libssl3, libcrypto3 (3.1.4-r2)	Applications performing certificate name checks may attempt to read an invalid memory address resulting in abnormal termination of the application process.
CVE-2023-6129	POLY1305 MAC implementation corrupts vector registers on PowerPC	MEDIUM	6.5	libssl3, libcrypto3 (3.1.4-r2)	The POLY1305 MAC implementation contains a bug that might corrupt the internal state of applications running on PowerPC CPU based platforms.

CVE ID	Title	Severity	CVSS Score	Package (Version)	Description
CVE-2023-6237	Excessive time spent checking invalid RSA public keys	MEDIUM	5.9	libssl3, libcrypto3 (3.1.4-r2)	Applications that use EVP_PKEY_public_check() may experience long delays with untrusted keys, leading to Denial of Service.
CVE-2024-0727	Denial of service via null dereference	MEDIUM	5.5	libssl3, libcrypto3 (3.1.4-r2)	Processing maliciously formatted PKCS12 files may lead to crashes via null pointer dereference.
CVE-2024-4741	Use After Free with SSL_free_buffers	MEDIUM	5.6	libssl3, libcrypto3 (3.1.4-r2)	Calling SSL_free_buffers may cause memory to be accessed after being freed, leading to crashes or arbitrary code execution.
CVE-2024-5535	SSL_select_next_proto buffer overread	MEDIUM	5.9	libssl3, libcrypto3 (3.1.4-r2)	Buffer overread could result in up to 255 bytes of private memory data being sent to peers.
CVE-2023-42363	Use-after-free in awk	MEDIUM	5.5	busybox, busybox-binsh (1.36.1-r15)	A use-after-free vulnerability in BusyBox's xasprintf function could lead to denial of service.
CVE-2023-42364	Use-after-free	MEDIUM	5.5	busybox, busybox-binsh (1.36.1-r15)	A use-after-free vulnerability in BusyBox allows attackers to cause denial of service via crafted awk patterns.

CVE ID	Title	Severity	CVSS Score	Package (Version)	Description
CVE-2023-42365	Use-after-free	MEDIUM	5.5	busybox, busybox-binsh (1.36.1-r15)	A use-after-free vulnerability in BusyBox via crafted awk patterns in the copyvar function.
CVE-2023-42366	Heap-buffer-overflow	MEDIUM	5.5	busybox, busybox-binsh (1.36.1-r15)	A heap-buffer-overflow vulnerability in BusyBox's next_token function at awk.c:1159.

You can find more details about the container scan results in the `trivy/` directory.

À rendre à cchassagnard@slb.com Amaury Viala: aviala@slb.com