MTC
WEB
SERVICES

DevSecOps на час

Андрей Моисеев
DevSecOps
Cloud Native Security

WHOAMI

Андрей Моисеев

DevSecOps
Cloud Native Security

Чем занимается DevSecOps?

0%

100%

Чем занимается DevSecOps?

01 Не даёт катить фичи

Чем занимается DevSecOps?

01 Не даёт катить фичи

02 Создаёт проблемы

Чем занимается DevSecOps?

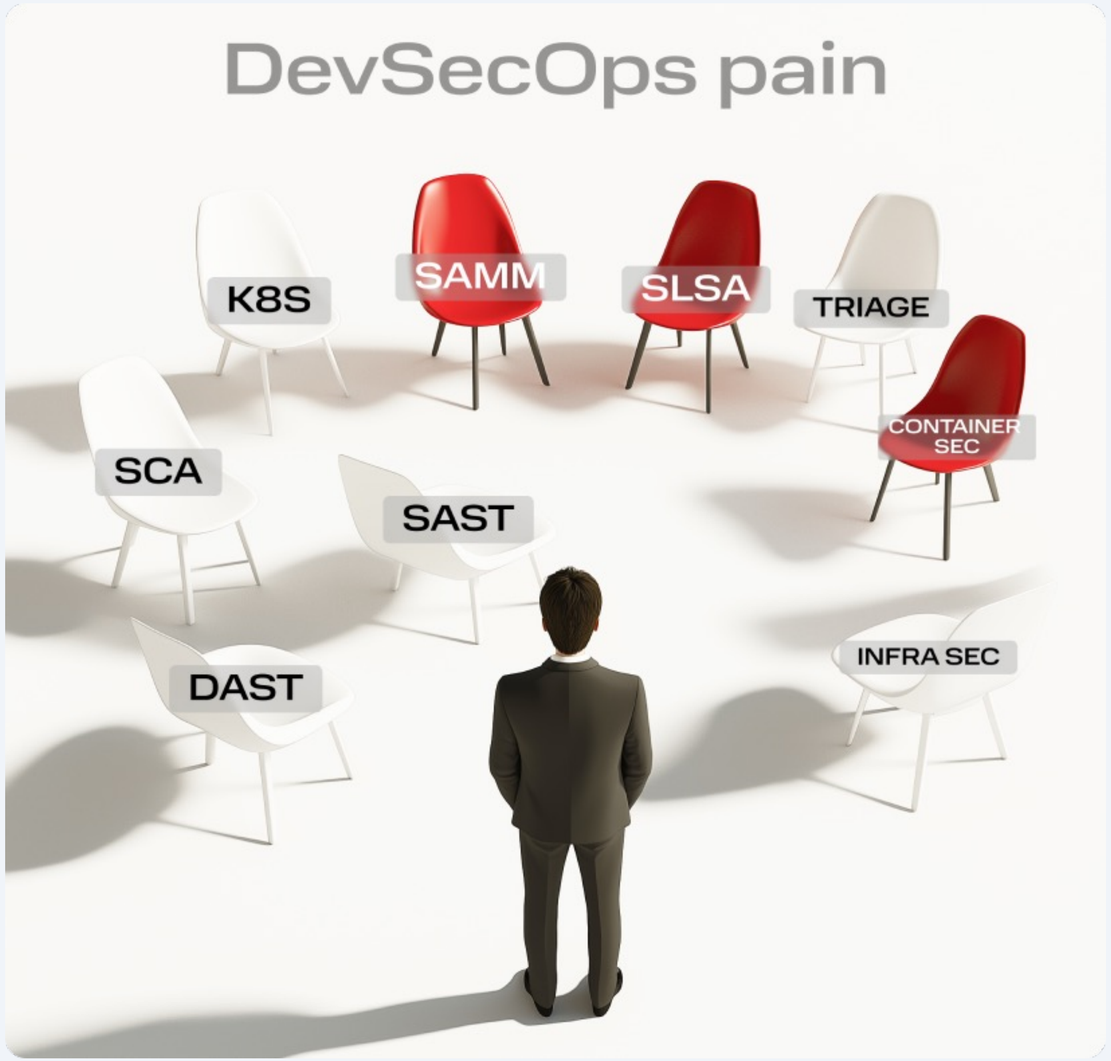
01 Не даёт катить фичи

02 Создаёт проблемы

03 Развивает безопасность

Проблема one-man-team

01 Иллюзия выбора



Проблема one-man-team

01 Иллюзия выбора

02 И один в поле воин

Проблема one-man-team

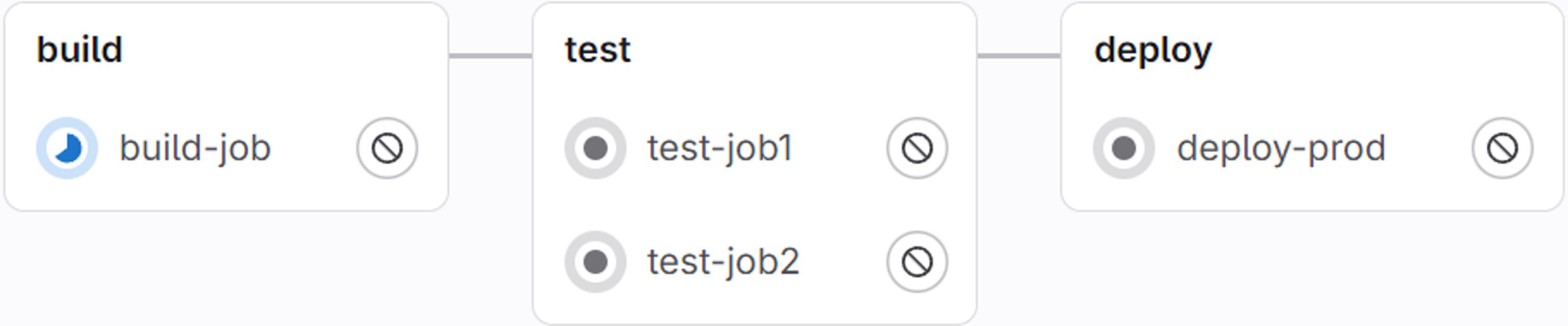
01 Иллюзия выбора

02 И один в поле воин

03 Нужно все и сразу

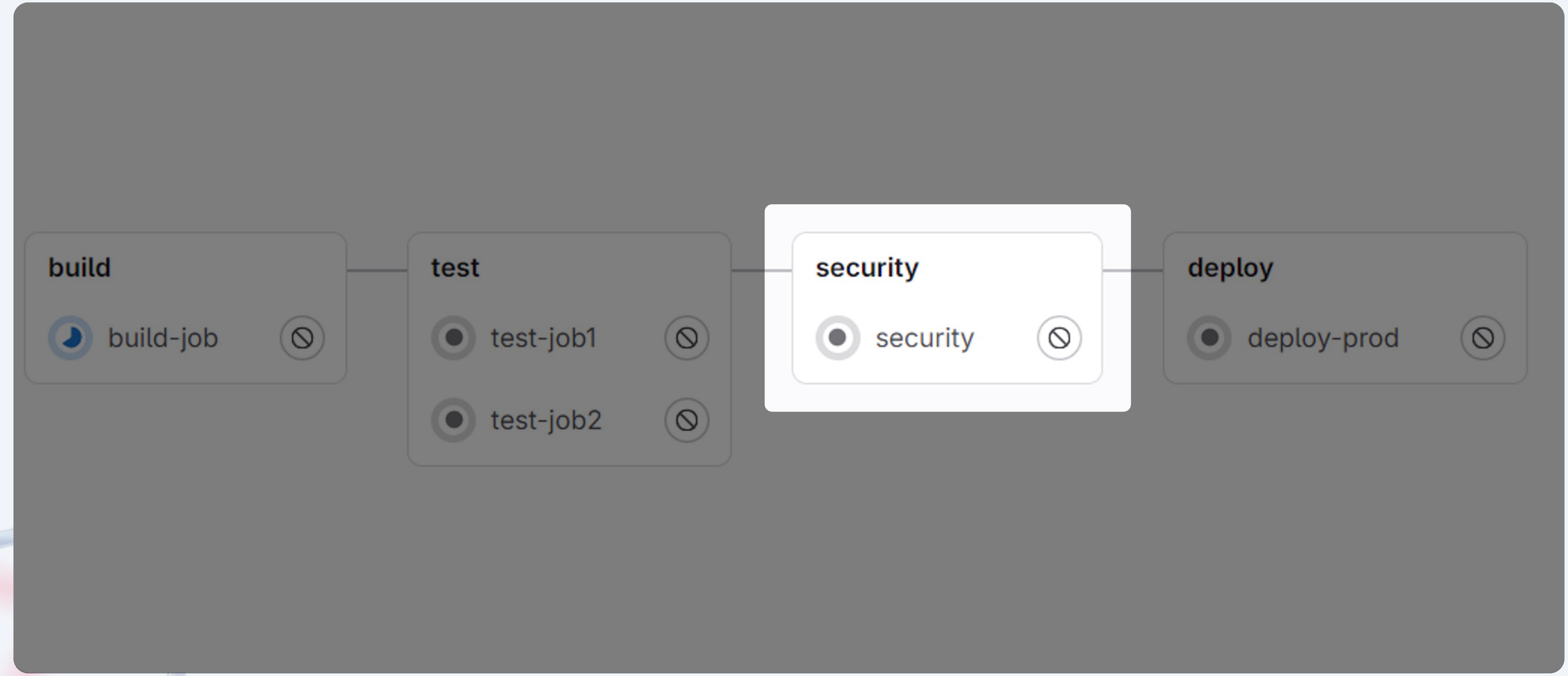
Начинаем!

Базовый CI/CD



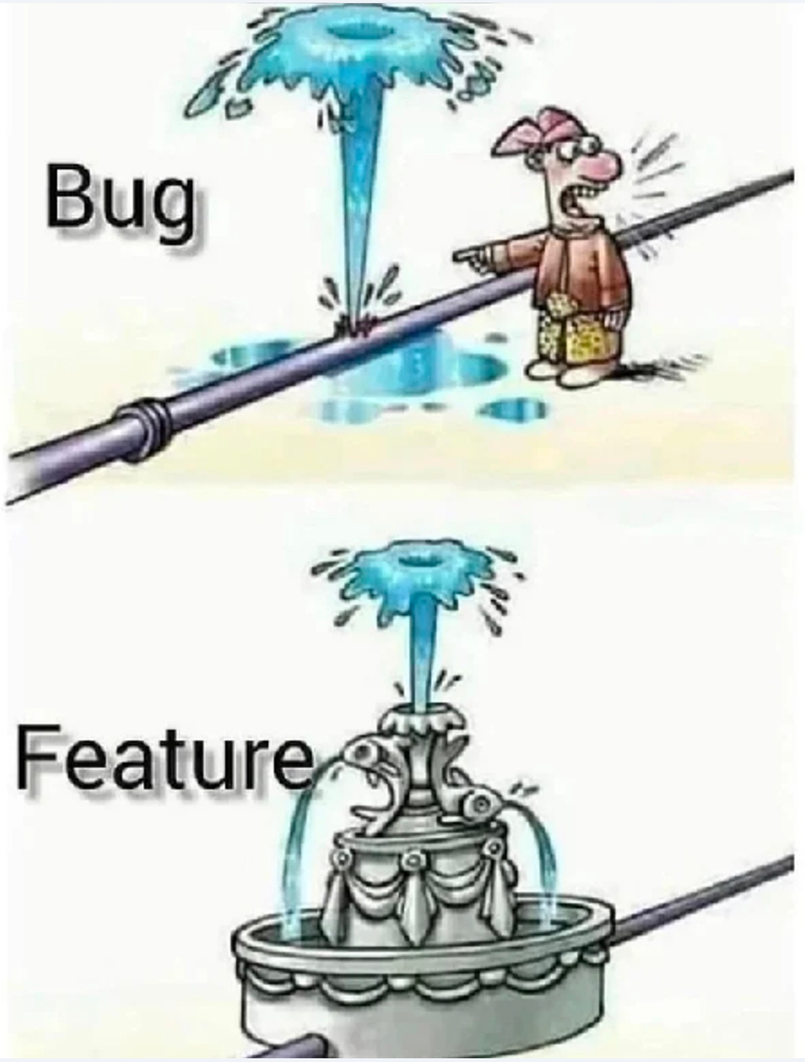
Базовый CI/CD

Нет security stage!



Зачем security

01 Собственный код может содержать уязвимости



Зачем security

01 Собственный код может содержать уязвимости

02 Заимствованный код, тоже может содержать уязвимости

Зачем security

01 Собственный код может содержать уязвимости

02 Заимствованный код, тоже может содержать уязвимости

03 Окружение тоже может содержать уязвимости

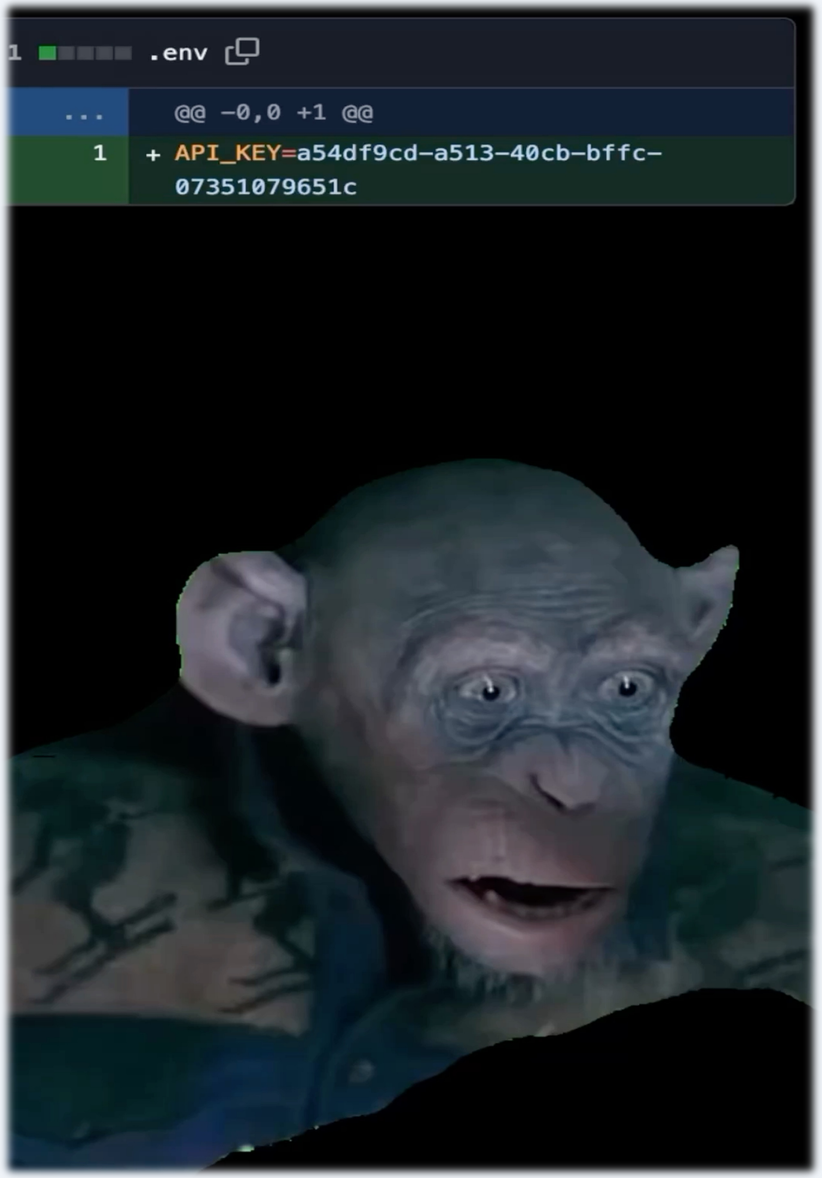
Зачем security

01 Собственный код может содержать уязвимости

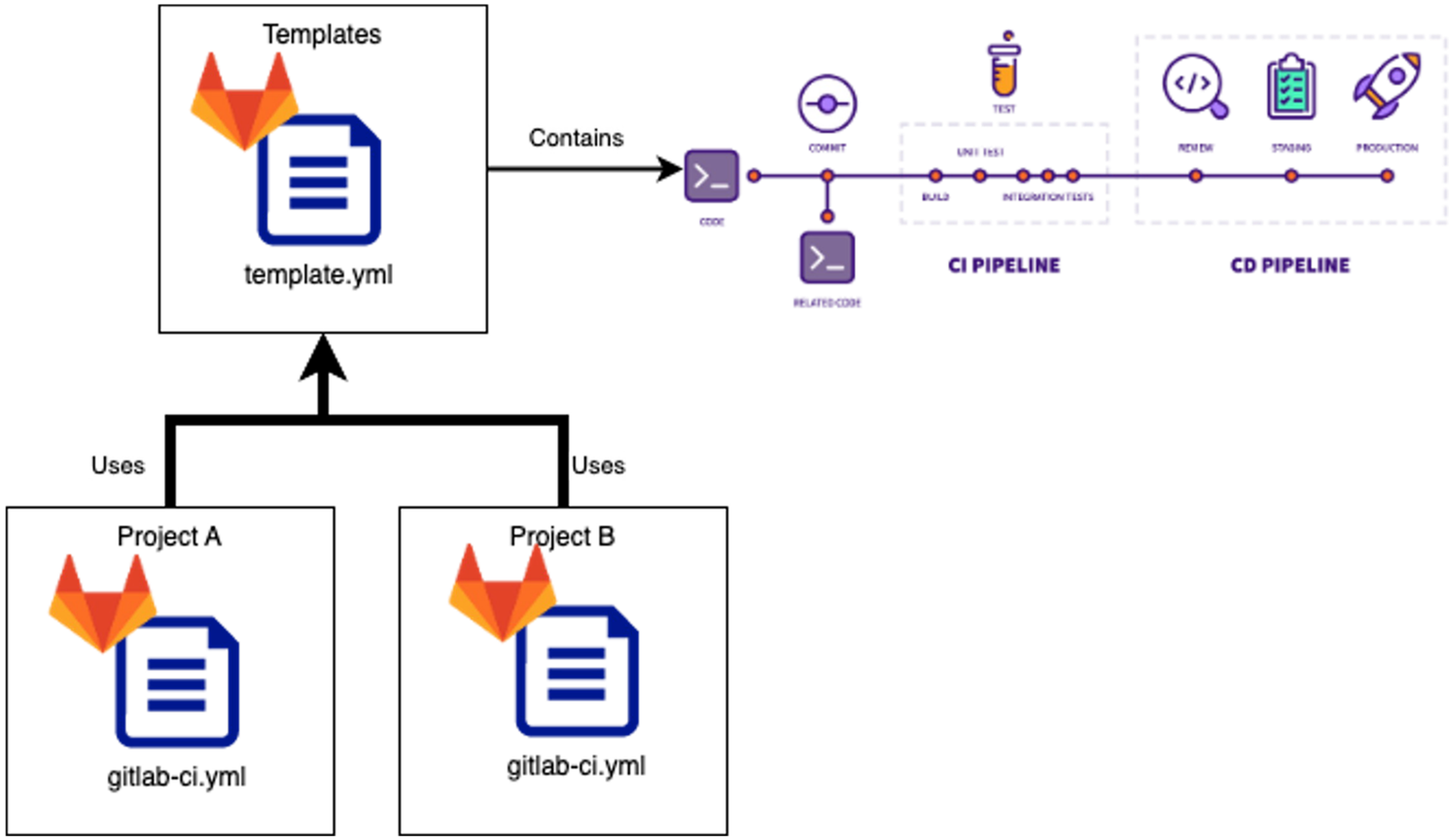
02 Заимствованный код, тоже может содержать уязвимости

03 Окружение тоже может содержать уязвимости

04 **Человеческий фактор**



Gitlab CI Security templates



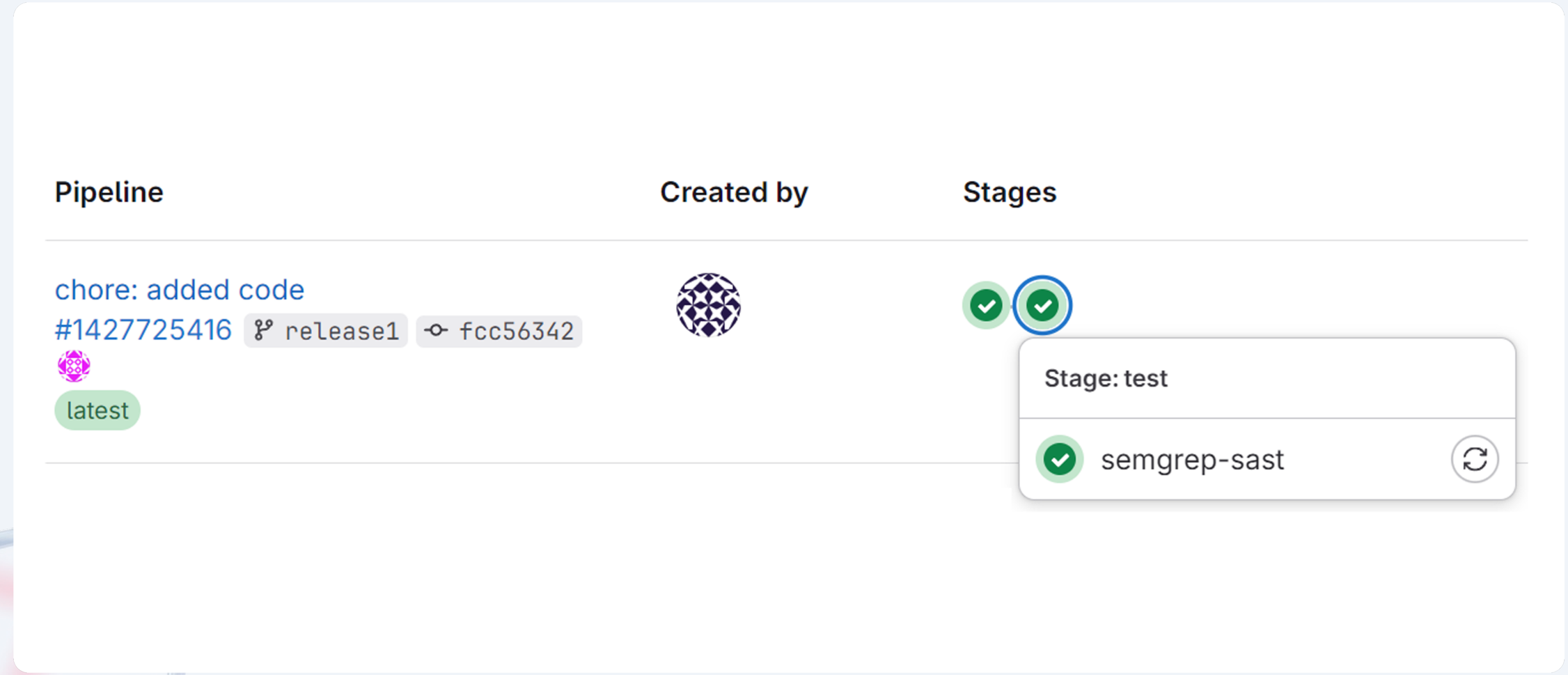
Gitlab CI Security templates

SAST

```yaml
stages:
  - build
  - test #needed for template
  - security
  - deploy

include:
  - template: Jobs/SAST.gitlab-ci.yml
```

Gitlab CI Security templates

SAST



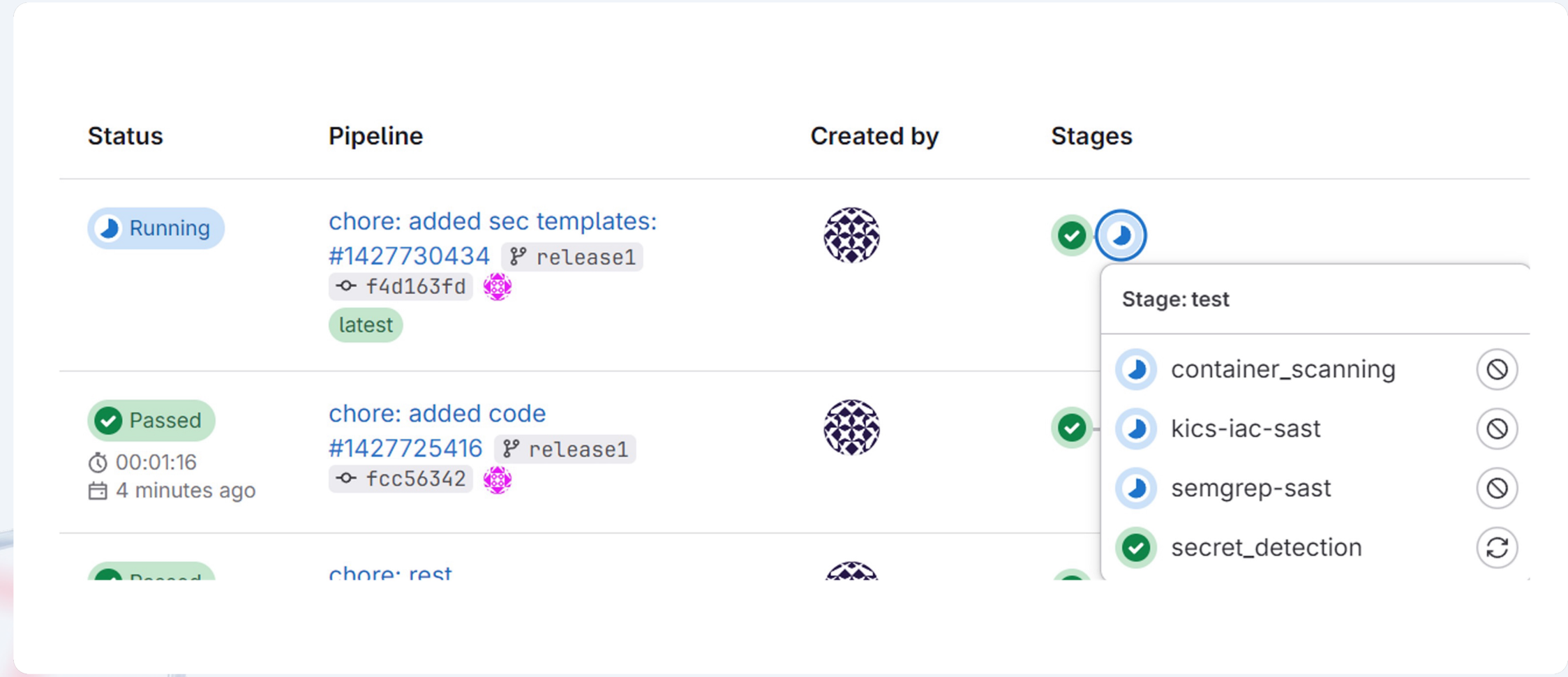
Gitlab CI Security templates

Остальные шаблоны

```yaml
stages:
  - build
  - test #needed for template
  - security
  - deploy

include:
  - template: Jobs/SAST.gitlab-ci.yml
  - template: Jobs/Container-Scanning.gitlab-ci.yml
  - template: Jobs/Secret-Detection.gitlab-ci.yml
  - template: Jobs/SAST-IaC.latest.gitlab-ci.yml
  - template: Jobs/Dependency-Scanning.gitlab-ci.yml
```

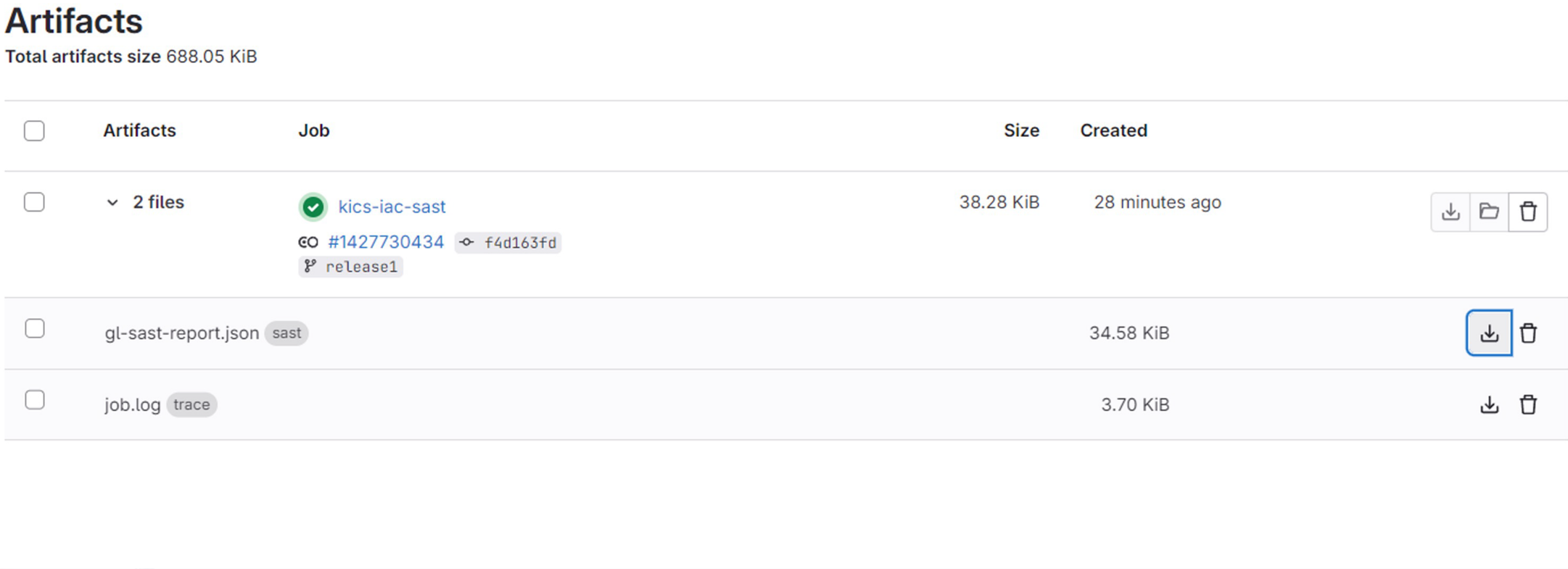
Gitlab CI Security templates

Остальные шаблоны



Gitlab CI Security templates

Результаты



Gitlab CI Security templates

Результаты

{} gl-sast-report (10).json ●

C: > Users > Work > Downloads > {} gl-sast-report (10).json > {} scan > {} scanner


```
{
    "version": "15.1.4",
    "vulnerabilities": [
        {
            "id": "1123c58da1b9278254607db97bc4ece1b65614e38851126eca9c1ce652a1cc4a",
            "category": "sast",
            "name": "Missing User Instruction",
            "description": "A user should be specified in the dockerfile, otherwise the image will run as root",
            "cve": "kics_id:fd54f200-402c-4333-a5a4-36ef6709af2f:1:0",
            "severity": "Critical",
            "scanner": {
                "id": "kics",
                "name": "kics"
            },
            "location": {
```

Gitlab CI Security templates. Profit

01 Подключается в одну строку

Gitlab CI Security templates. Profit

01 Подключается в одну строку

02 Не требует тонкой настройки

Gitlab CI Security templates. Profit

01 Подключается в одну строку

02 Не требует тонкой настройки

03 Есть для разных сканеров

Gitlab CI Security templates. Profit

- Подключается в одну строку
- Не требует тонкой настройки
- Есть для разных сканеров
- **НО ВСЕГДА ЕСТЬ НЮАНС**

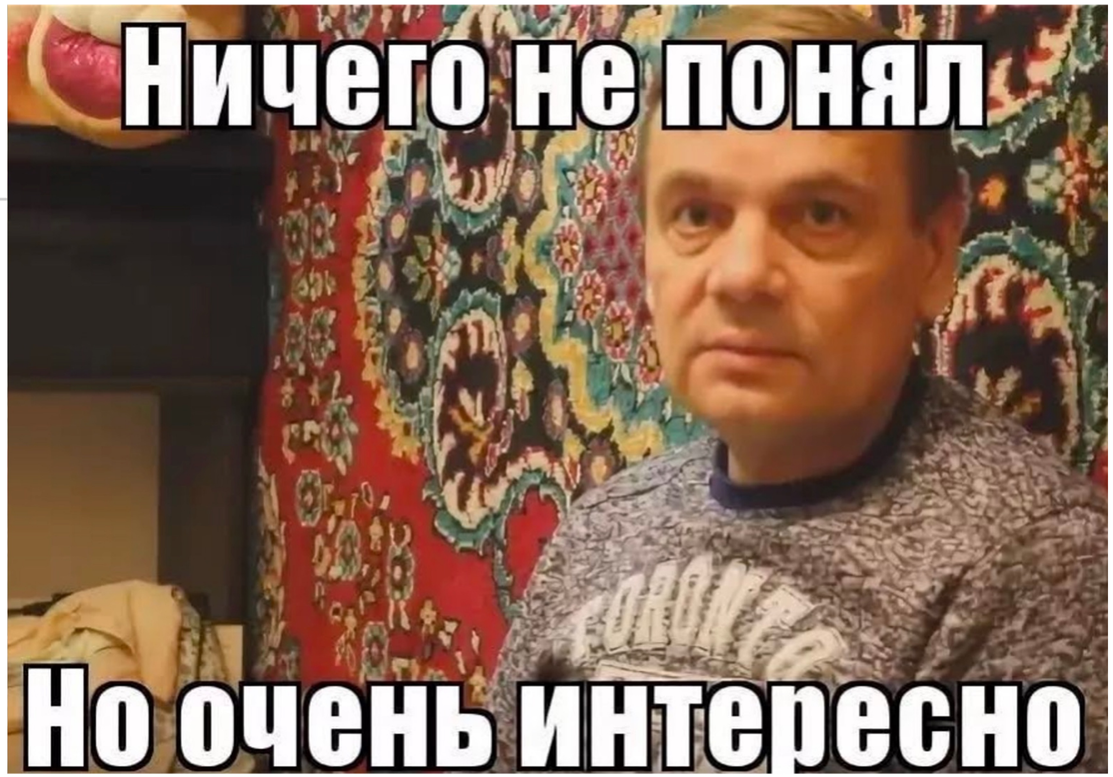
Проблемы шаблонов

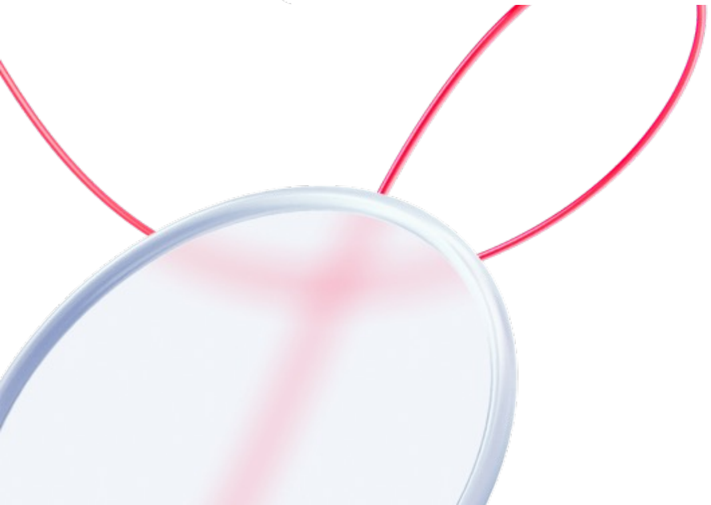
Может не работать из коробки

```
$ gtcs scan
[INFO] [2024-08-26 17:11:05 +0000] [container-scanning]  >  Remediation is disabled; /builds/off.moiseev.andrey/devoops/Dockerfile cannot be found. Have you set `GIT_STRATEGY` and
`CS_DOCKERFILE_PATH`?
See https://docs.gitlab.com/ee/user/application_security/container_scanning/#solutions-for-vulnerabilities-auto-remediation
[INFO] [2024-08-26 17:11:06 +0000] [container-scanning]  >  Scanning container from registry registry.gitlab.com/off.moiseev.andrey/devoops/release1:f4d163fd7638f6b12b792e3f852eae2d762ec842 for vulnerabilities with severity level UNKNOWN or higher, with gcs 7.3.5 and Trivy Version: 0.52.1, advisories updated at 2024-08-26T12:12:59+00:00
[ERROR] [2024-08-26 17:11:07 +0000] [container-scanning]  >  Scanner has not created a file with results (tmp.json)
```

Проблемы шаблонов

- Может не работать из коробки
- Все-таки придется разобраться в чужом коде





Gitlab CI Security templates. Выводы

01 Неуправляемый флоу сканеров

Gitlab CI Security templates. Выводы

01 Неуправляемый флоу сканеров

02 Зачастую сложная реализация

Gitlab CI Security templates. Выводы

01 Неуправляемый флоу сканеров

02 Зачастую сложная реализация

03 Нужен кастом под себя

Gitlab CI Security templates. Выводы

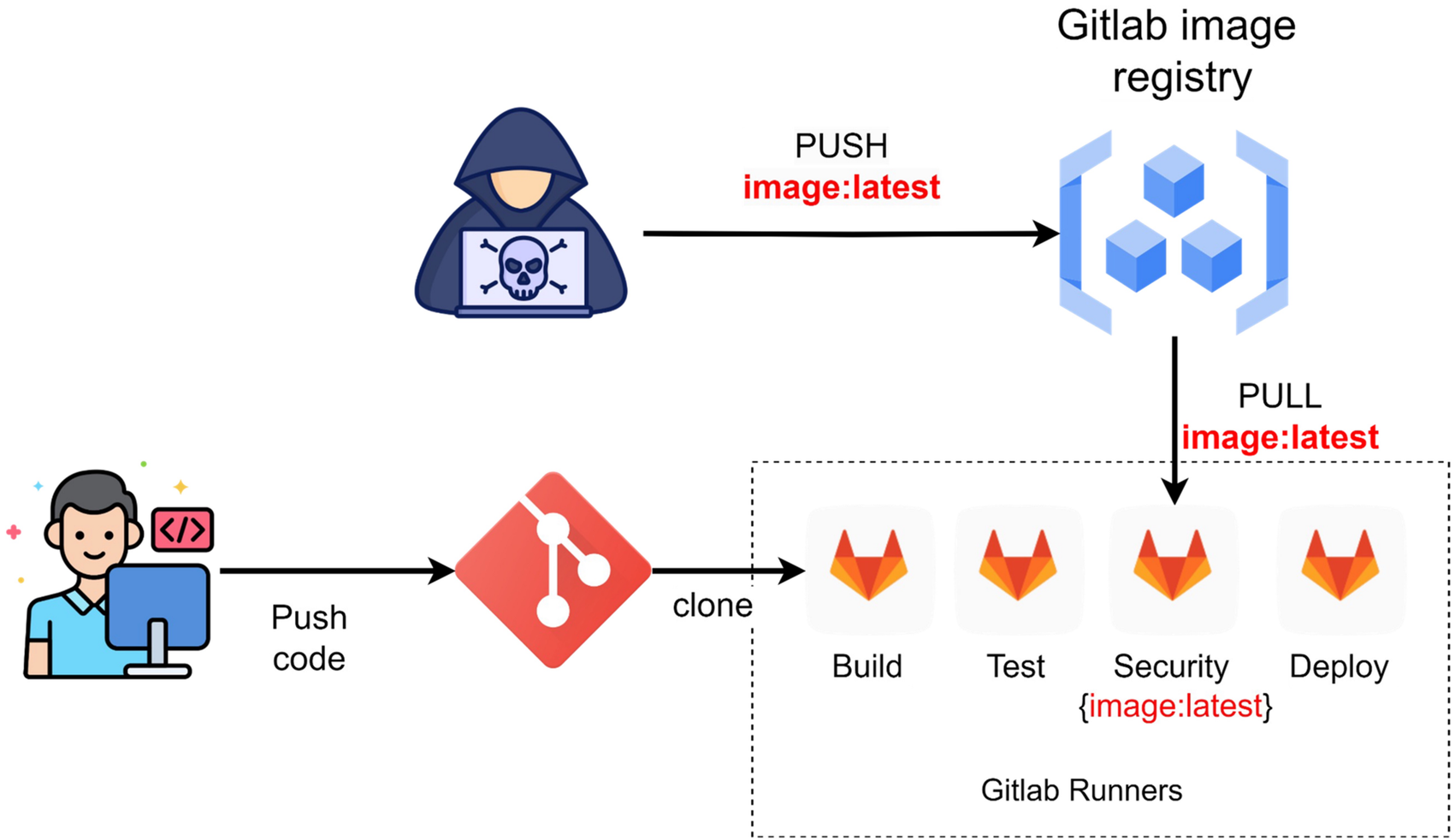
01 Неуправляемый флоу сканеров

02 Зачастую сложная реализация

03 Нужен кастом под себя

04 **Прямая зависимость от gitlab registry**

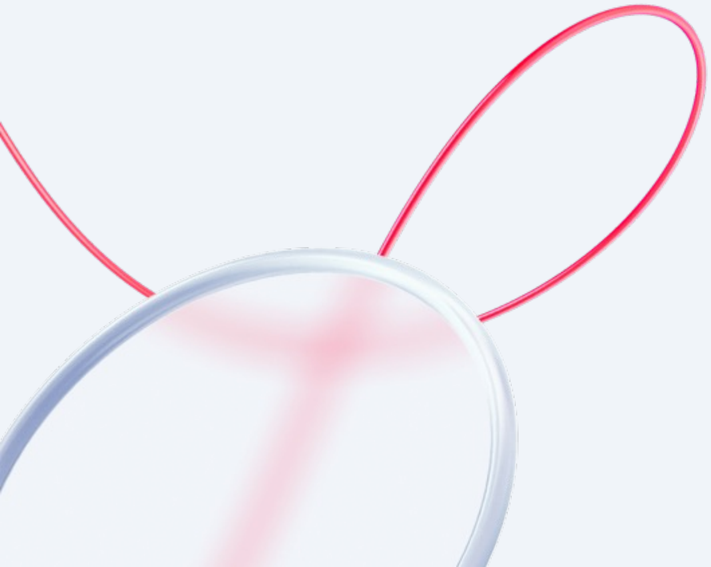
Dependency confusion



Dependency confusion

```
.ds-analyzer:
  extends: dependency_scanning
  allow_failure: true
  variables:
    # DS_ANALYZER_IMAGE is an undocumented variable used internally to allow QA to
    # override the analyzer image with a custom value. This may be subject to change or
    # breakage across GitLab releases.
    DS_ANALYZER_IMAGE: "$SECURE_ANALYZERS_PREFIX/$DS_ANALYZER_NAME:$DS_MAJOR_VERSION"
    # DS_ANALYZER_NAME is an undocumented variable used in job definitions
    # to inject the analyzer name in the image name.
    DS_ANALYZER_NAME: ""
  image:
    name: "$DS_ANALYZER_IMAGE$DS_IMAGE_SUFFIX"
```

Что делаем?



Создаём свой шаблон

Наш шаблон

andrey moiseev / **Security-template**

main | security-template / .gitlab / templates / .secuirty-template.yml

chore: init
Andrey Moiseev authored just now

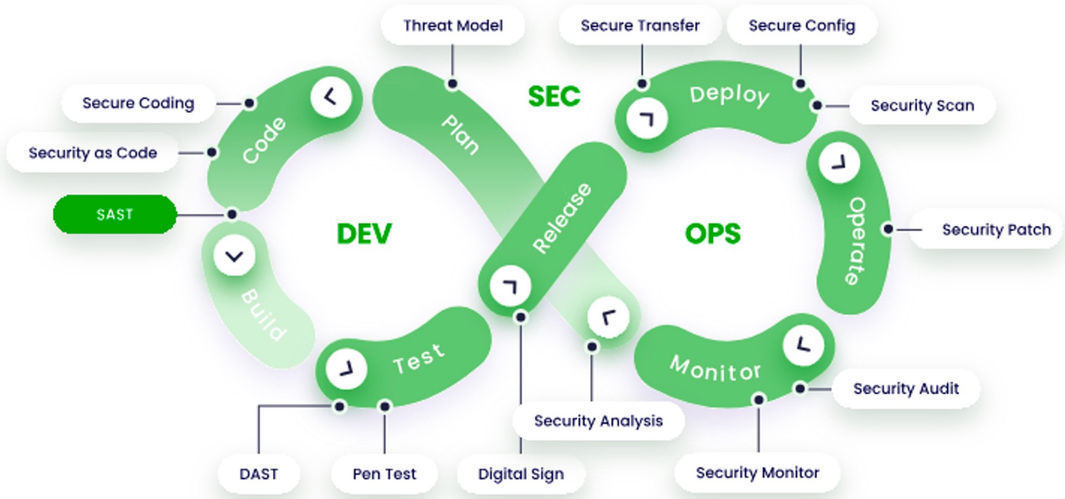
.secuirty-template.yml 317 B

```yaml
# Security jobs
variables:


sast:
  allow_failure: true
  stage: security

secret_detection:
  allow_failure: true
  stage: security
```

0% 50% 100%

Что нам нужно

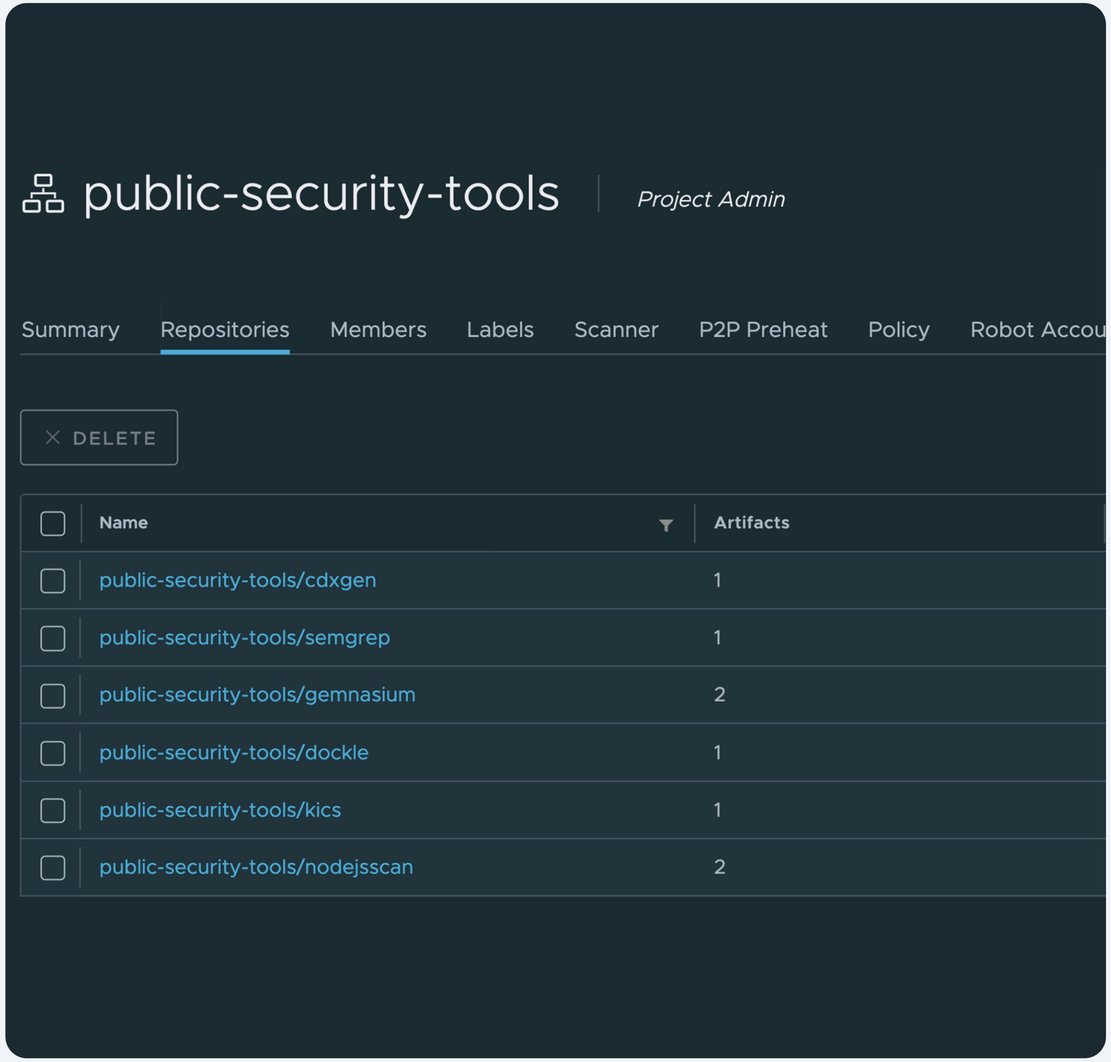
01 Перечень security практик, которые внедряем



Что нам нужно

01 Перечень security практик, которые внедряем

02 Собственные образы для сканеров



Что нам нужно

01 Перечень security практик, которые внедряем

02 Собственные образы для сканеров

03 Внутреннее registry, чтобы их хранить

GitLab
Container Registry

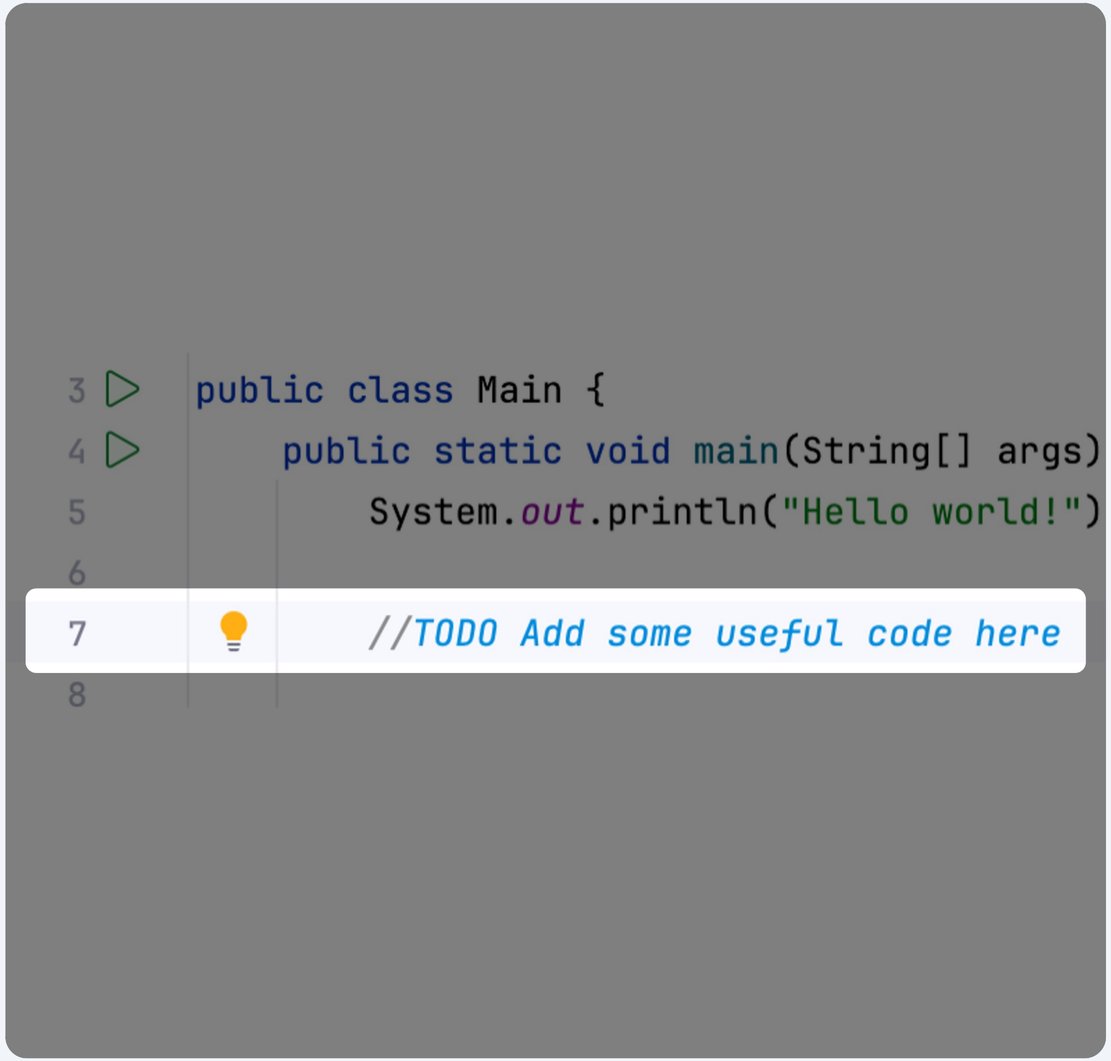
Что нам нужно

01 Перечень security практик, которые внедряем

02 Собственные образы для сканеров

03 Внутреннее registry, чтобы их хранить

04 Реализация security job



Перечень практик

01 Secret management — GitLeaks

OWASP Top 10:2021

A01:2021 – Broken Access Control

Перечень практик

01 Secret management — GitLeaks

02 SCA — Gemnasium

OWASP Top 10:2021

A06:2021 – Vulnerable and Outdated Components

Перечень практик

01 Secret management — GitLeaks

02 SCA — Gemnasium

03 SAST — nodejsscan

04 Additional SAST — Semgrep

05 IAC SAST — Kics

OWASP Top 10:2021

A01:2021 – Broken Access Control

OWASP Top 10:2021

A02:2021 – Cryptographic Failures

OWASP Top 10:2021

A05:2021 – Security Misconfiguration

Secret management

Xakep
https://xakep.ru

Автомобили Kia можно было взломать удаленно, зная

3 дня назад — «Эти атаки могли выполняться удаленно на любом оборудованном

Kia

Nokia

Secret management

Xakep
https://xakep.ru

Автомобили Kia можно было взломать удаленно, зная

3 дня назад — «Эти атаки могли выполняться удаленно на любом оборудованном

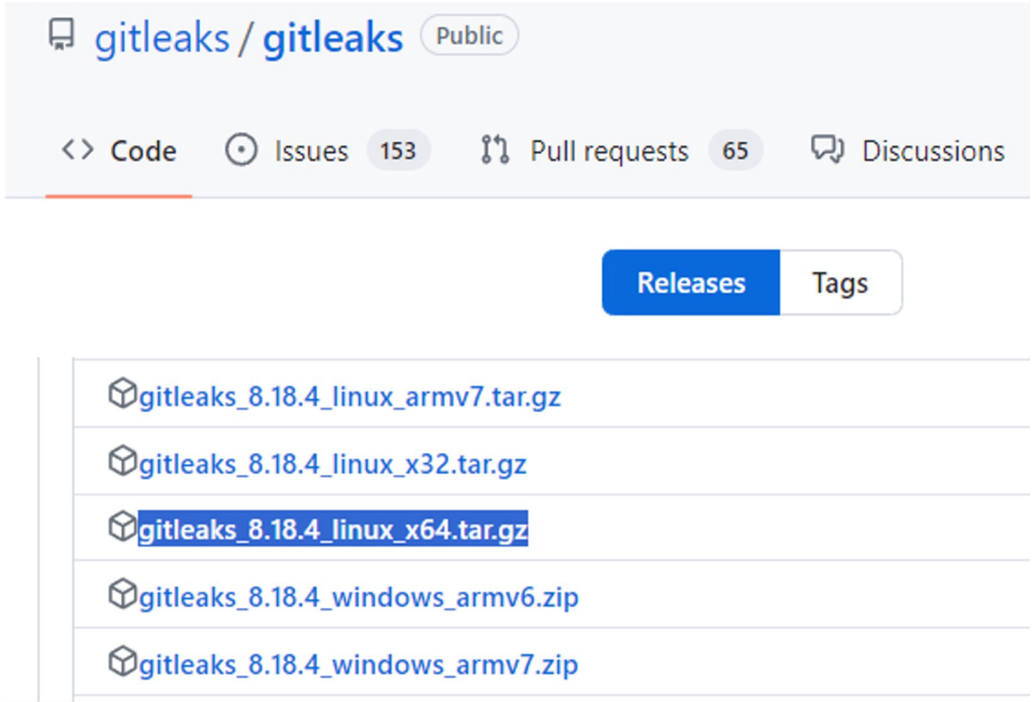
```
POST /apps/services/owners/apigwServlet.html HTTP/2
Host: owners.kia.com
Httpmethod: GET
Apiurl: /door/unlock
Servicetype: postLoginCustomer
Cookie: JSESSIONID=SESSION_TOKEN;
```

Kia

Nokia

Secret management

01 Берем сорсы
сканера с github.com



Secret management

01 Берем сорсы
сканера с github.com

02 Пишем свой докерфайл

Secret management

```
1  FROM alpine:3.18.2
2
3  ARG GITLEAKS_VERSION=8.18.1
4  ARG GITLEAKS_SHA256="3e157a26081e296d4cb94ef0d87441c9afc5f392cb02957656dd5cfeb7aaf6c9"
```

Secret management

```
FROM alpine:3.18.2

ARG GITLEAKS_VERSION=8.18.1
ARG GITLEAKS_SHA256="3e157a26081e296d4cb94ef0d87441c9afc5f392cb02957656dd5cfeb7aaf6c9"

RUN adduser -D gitleaks && \
apk update && apk upgrade --no-cache && \
apk add --no-cache git curl &&\
wget https://github.com/gitleaks/gitleaks/releases/download/v${GITLEAKS_VERSION}/gitleaks_${GITLEAKS_VERSION}_
sha256sum /tmp/gitleaks.tar.gz > sha256sum.txt && \
```

Secret management

```
FROM alpine:3.18.2

ARG GITLEAKS_VERSION=8.18.1
ARG GITLEAKS_SHA256="3e157a26081e296d4cb94ef0d87441c9afc5f392cb02957656dd5cfeb7aaf6c9"

RUN adduser -D gitleaks && \
apk update && apk upgrade --no-cache && \
apk add --no-cache git curl &&\
wget https://github.com/gitleaks/gitleaks/releases/download/v${GITLEAKS_VERSION}/gitleaks_${GITLEAKS_VERSION}_
sha256sum /tmp/gitleaks.tar.gz > sha256sum.txt && \
cat sha256sum.txt && \
echo ${GITLEAKS_SHA256} && \
echo "${GITLEAKS_SHA256} /tmp/gitleaks.tar.gz" | sha256sum -c && \
tar xf /tmp/gitleaks.tar.gz -C /tmp gitleaks && \
rm -f /tmp/gitleaks.tar.gz && \
mv /tmp/gitleaks /usr/local/bin/gitleaks
```

Secret management

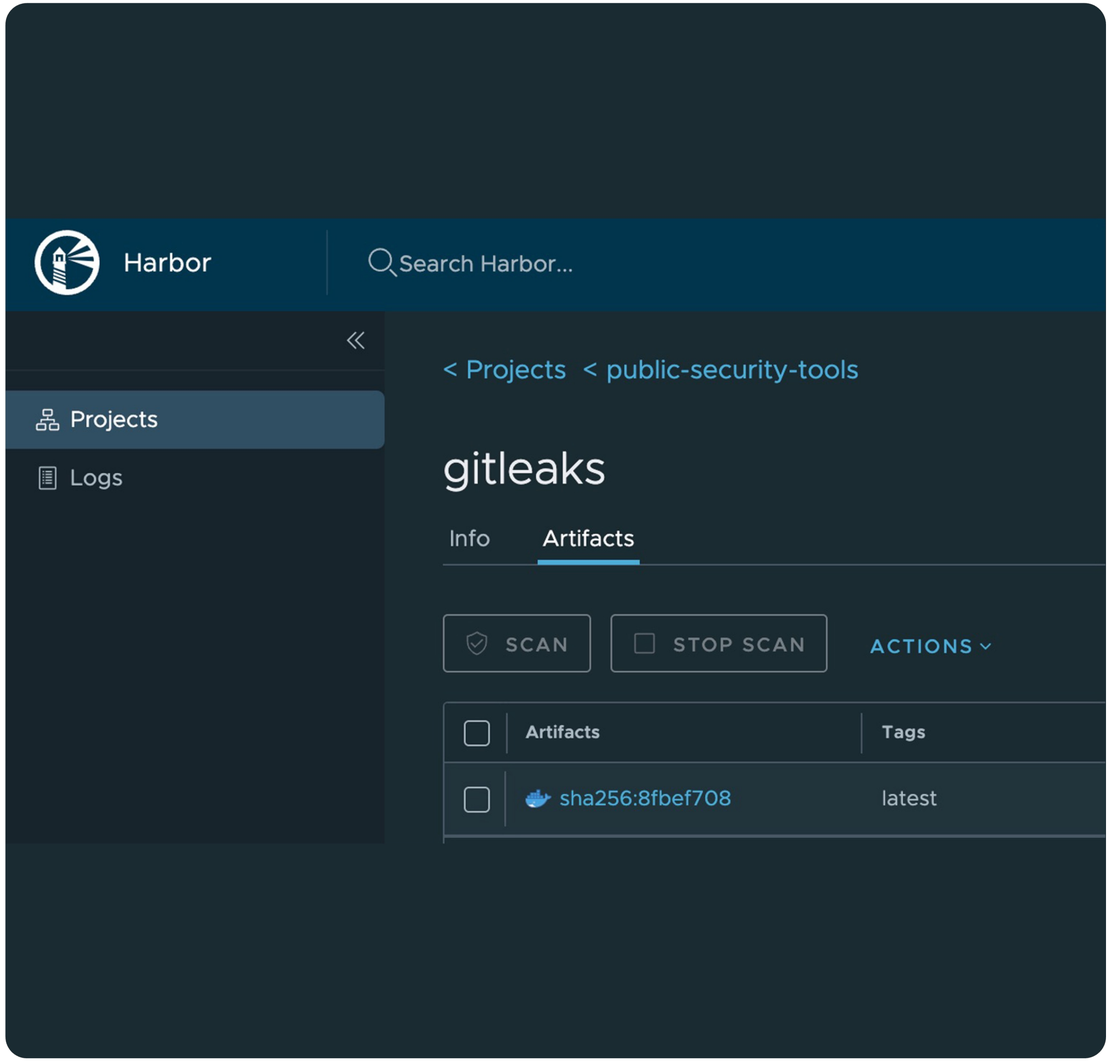
```
FROM alpine:3.18.2

ARG GITLEAKS_VERSION=8.18.1
ARG GITLEAKS_SHA256="3e157a26081e296d4cb94ef0d87441c9afc5f392cb02957656dd5cfeb7aaf6c9"

RUN adduser -D gitleaks && \
apk update && apk upgrade --no-cache && \
apk add --no-cache git curl &&\
wget https://github.com/gitleaks/gitleaks/releases/download/v${GITLEAKS_VERSION}/gitleaks_${GITLEAKS_VERSION}_
sha256sum /tmp/gitleaks.tar.gz > sha256sum.txt && \
cat sha256sum.txt && \
echo ${GITLEAKS_SHA256} && \
echo "${GITLEAKS_SHA256} /tmp/gitleaks.tar.gz" | sha256sum -c && \
tar xf /tmp/gitleaks.tar.gz -C /tmp gitleaks && \
rm -f /tmp/gitleaks.tar.gz && \
mv /tmp/gitleaks /usr/local/bin/gitleaks

USER gitleaks
WORKDIR /home/gitleaks

RUN git config --global --add safe.directory '*'

ENTRYPOINT [""]
```

Secret management

01 Берем сорсы
сканера с github.com

02 Пишем свой докерфайл

03 Пушим во внутренний режистри

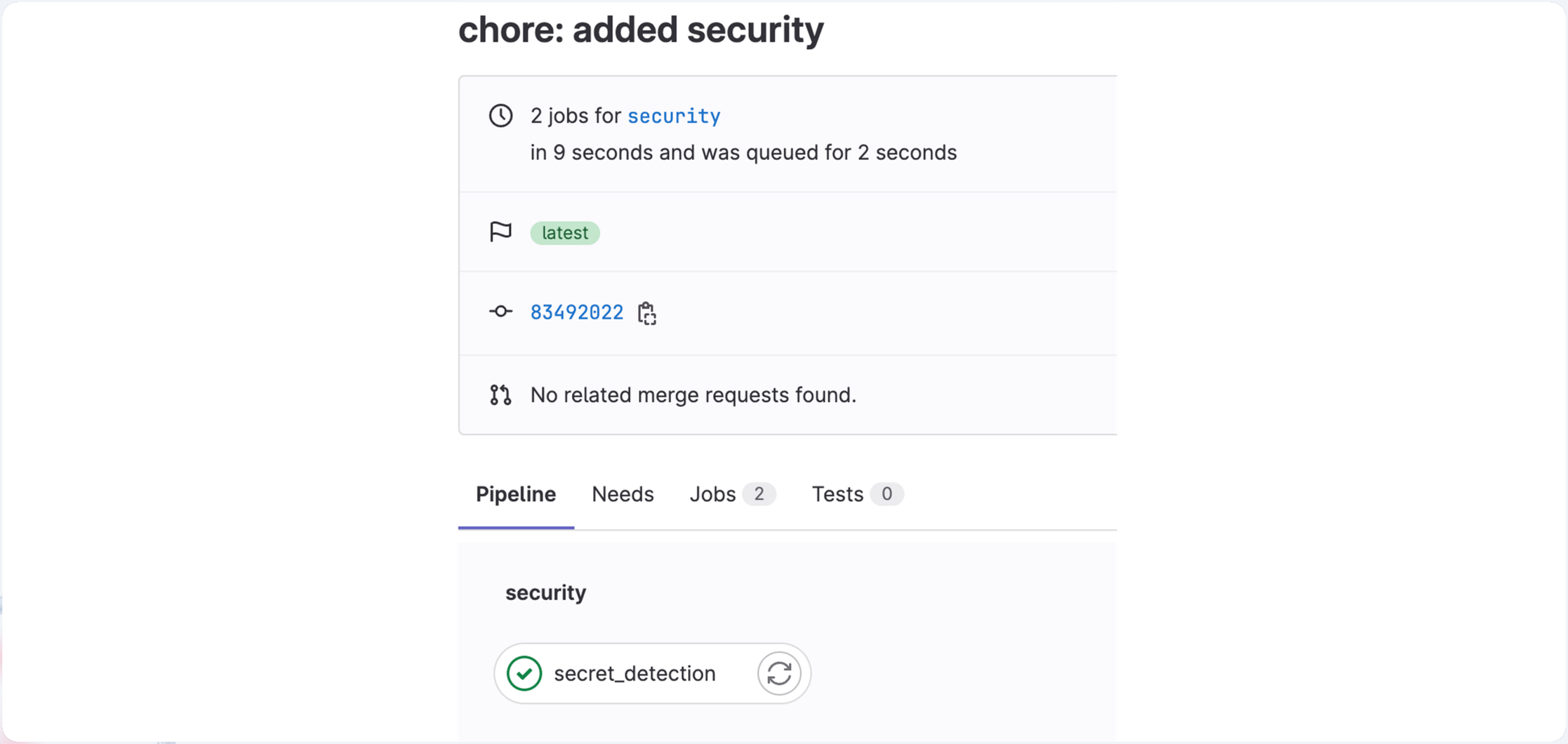


И конечно YAML

security-template.yml

```yaml
secret_detection:
  stage: security
  image:
    name: "registry.resources.cloud.mts.ru/public-security-tools/gitleaks"
  variables:
    GIT_DEPTH: "50"
    SECRET_DETECTION_HISTORIC_SCAN: "true"
  services: []
  allow_failure: true
  tags:
    - devoops-docker
  script:
    - gitleaks detect --exit-code "0" --source . --log-opts="--all" --report-path gitleaks-results.json
  artifacts:
    paths:
      - gitleaks-*.*
```

Secret management



Соберём остальные сканеры

master devoops_sectools / + Find file Web IDE Clone

README Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster Set up CI/CD Add Wiki Configure Integrations

Name	Last commit	Last update
dockle	chore: sec	2 minutes ago
gemnasium	chore: sec	2 minutes ago
gitleaks	chore: sec	2 minutes ago
kics	chore: sec	2 minutes ago
kubesec	chore: sec	2 minutes ago
nodejsscan	chore: sec	2 minutes ago
security-code-scan	chore: sec	2 minutes ago
semgrep	chore: sec	2 minutes ago
README.md	Initial commit	14 hours ago

Допишем нужные джобы

```yaml
.cyclonedx-reports:
  artifacts:
    paths:
      - "**/gl-sbom-*.cdx.json"
      - gl-dependency-scanning-report.json
    reports:
      cyclonedx: "**/gl-sbom-*.cdx.json"

gemnasium-dependency_scanning:
  extends:
    - .ds-analyzer
    - .cyclonedx-reports
  variables:
    DS_ANALYZER_NAME: "gemnasium"
    GEMNASIUM_LIBRARY_SCAN_ENABLED: "true"
```

Допишем нужные джобы

```yaml
nodejs-scan-sast:
  allow_failure: true
  stage: security
  tags:
    - devoops-gitleaks
  image:
    name: "registry.resources.cloud.mts.ru/public-security-tools/nodejsscan"
  script:
    - njsscan . --json -o ${PWD}/nodejs-results.json
  artifacts:
    paths:
       - nodejs-results.json


kics-iac-sast:
  allow_failure: true
  stage: security
  image:
    name: "registry.resources.cloud.mts.ru/public-security-tools/kics"
    entrypoint: [""]
  tags:
    - devoops-gitleaks
  script:
    - kics scan -s -p ${PWD} -o ./ --disable-secrets --report-formats json -o ${PWD} --output-name kics-results
  artifacts:
    name: kics-results.json
    paths:
      - kics-results.json
    when: always
```

Еще немного YAML

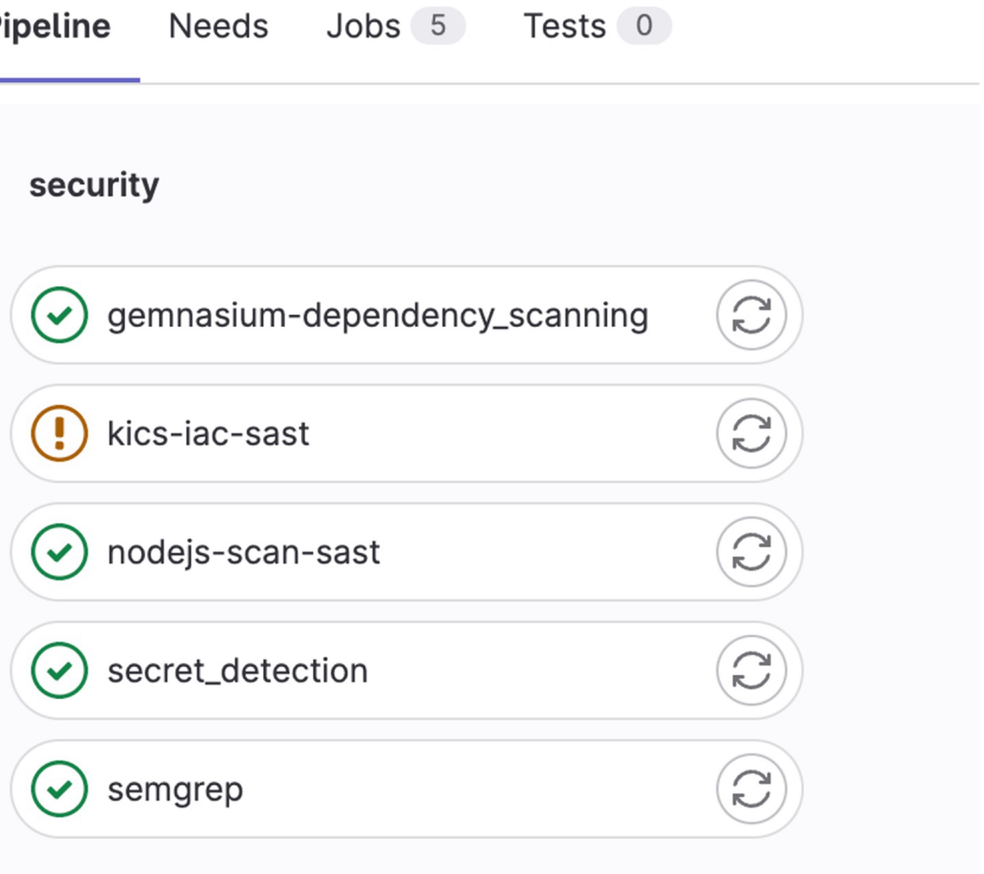
```yaml
semgrep:
  stage: security
  image:
    name: "registry.resources.cloud.mts.ru/public-security-tools/semgrep"
  artifacts:
    paths:
      - gl-sast-report.json
  tags:
    - devoops-gitleaks
  variables:
    SEARCH_MAX_DEPTH: '4'
  rules:
    - if: true
      exists:
        - '**/*.py'
        - '**/*.js'
        - '**/*.jsx'
```

Ну вы поняли

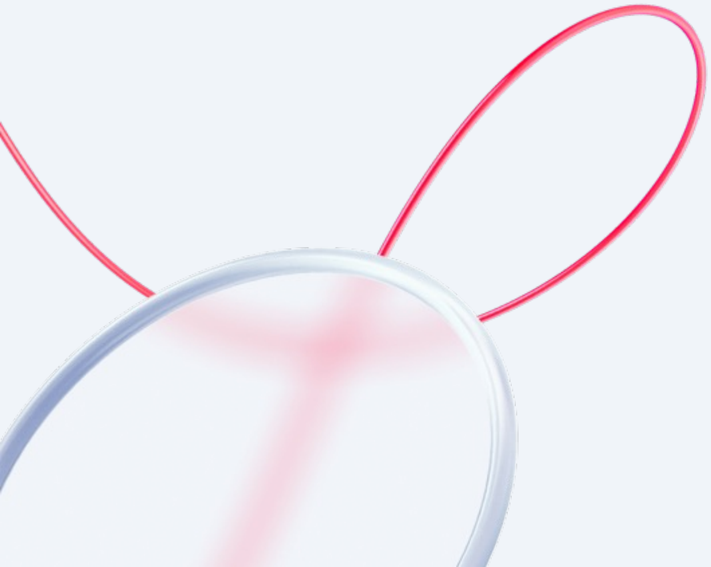
Ура – шаблон работает!

.gitlab-ci.yml 123 bytes

```yaml
stages:
 - build
 - test #needed for template
 - security
 - deploy

include:
  - local: 'security-template.yml'
```

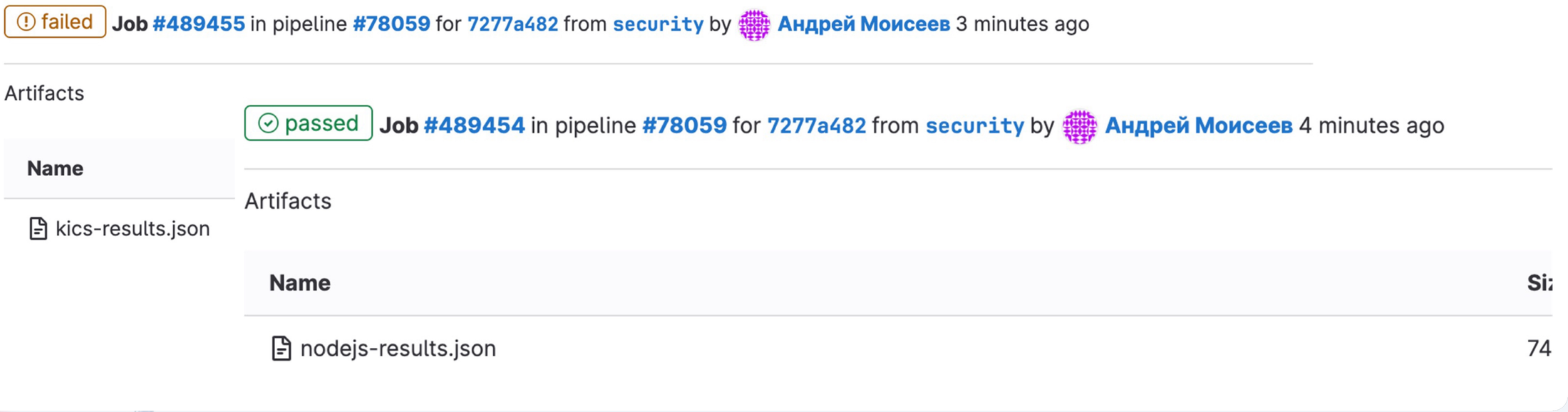


Но есть проблемы



Проблемы

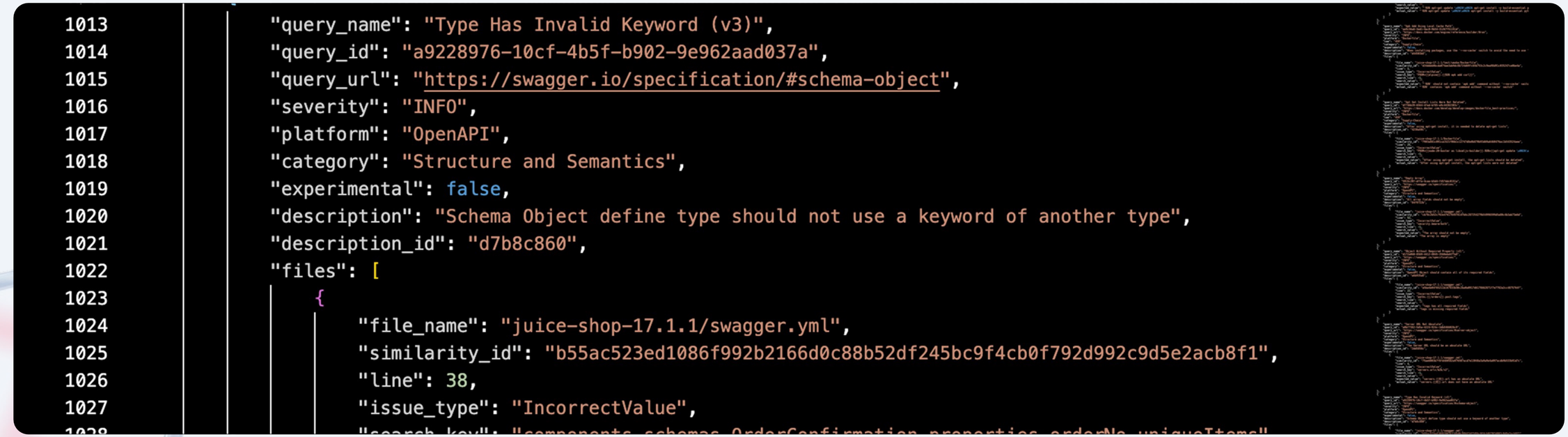
01 Всё помещается в gitlab artifacts



Проблемы

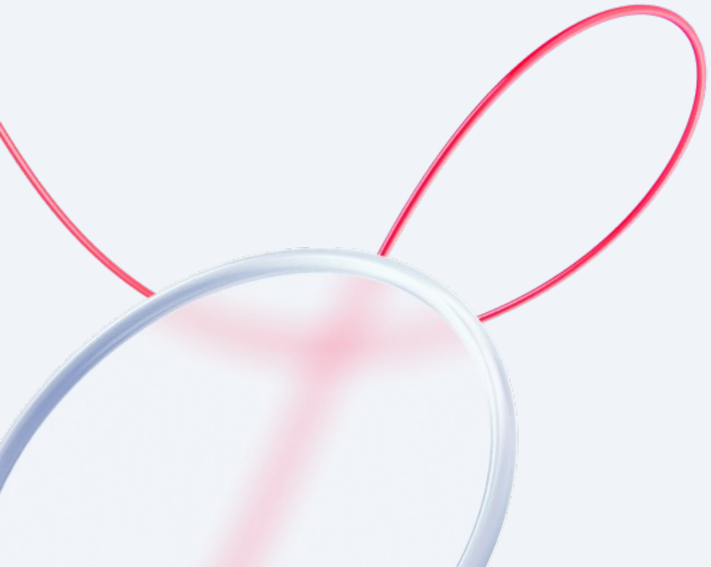
01 Всё помещается в gitlab artifacts

02 Результаты AS IS



Проблемы

01	Всё помещается в gitlab artifacts
02	Результаты AS IS
03	Много файндингов, пайплайнов



Проблемы

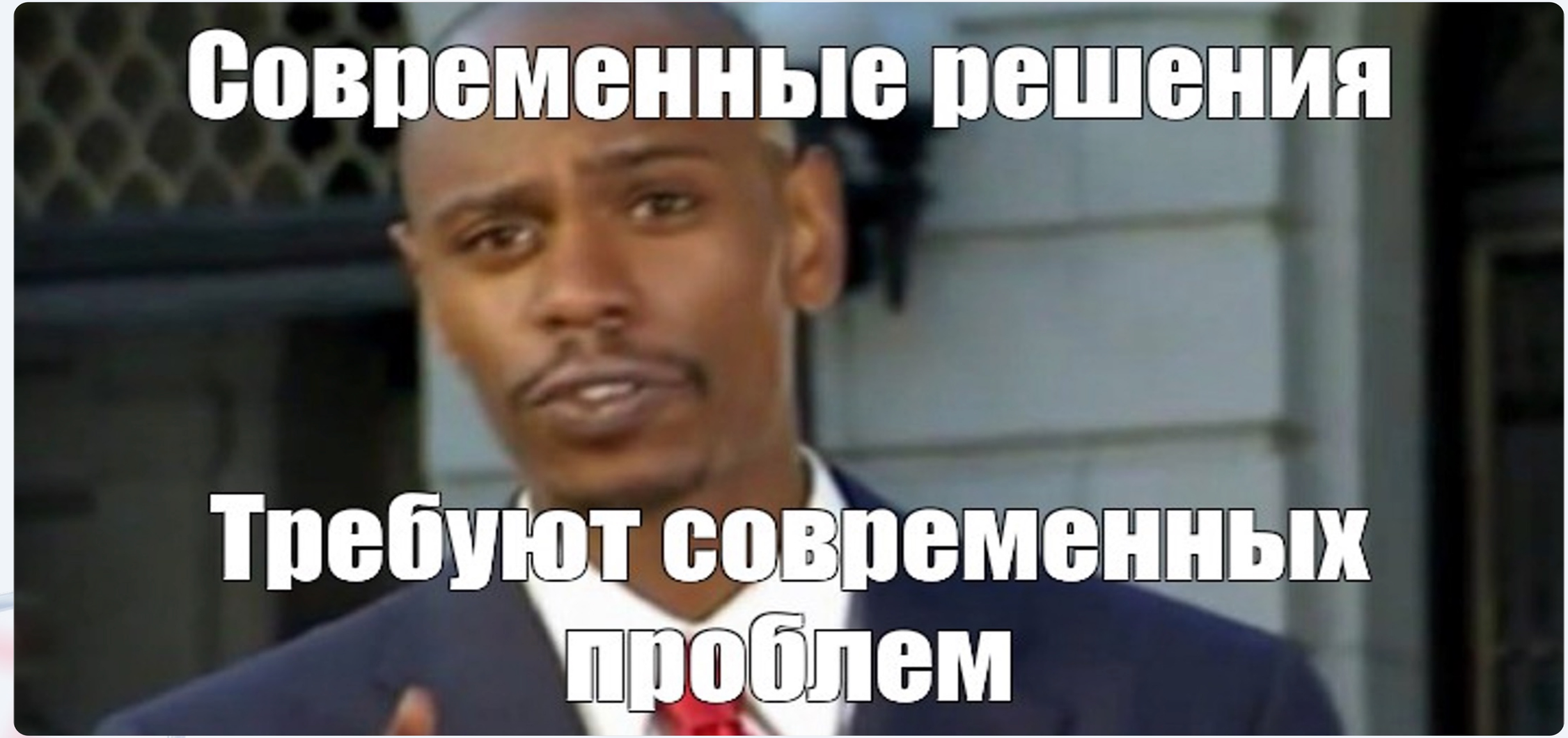
01 Всё помещается в gitlab artifacts

02 Результаты AS IS

03 Много файндингов, пайплайнов

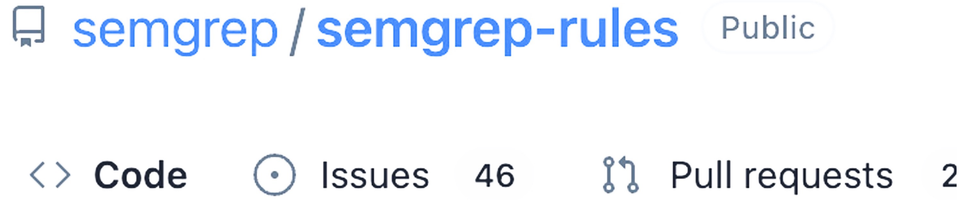
04 **Разработчики когда видят отчет →**

Проблемы надо решать



Что предпримем?

01 Используем минимальные наборы правил



0% 70% 100%

Что предпримем?

01 Используем минимальные наборы правил

02 Реализуем триаж

0%

70%

100%

Что предпримем?

01 Используем минимальные наборы правил

02 Реализуем триаж

03 Согласуем SLA с разработчиками

0% 70% 100%

Что предпримем?

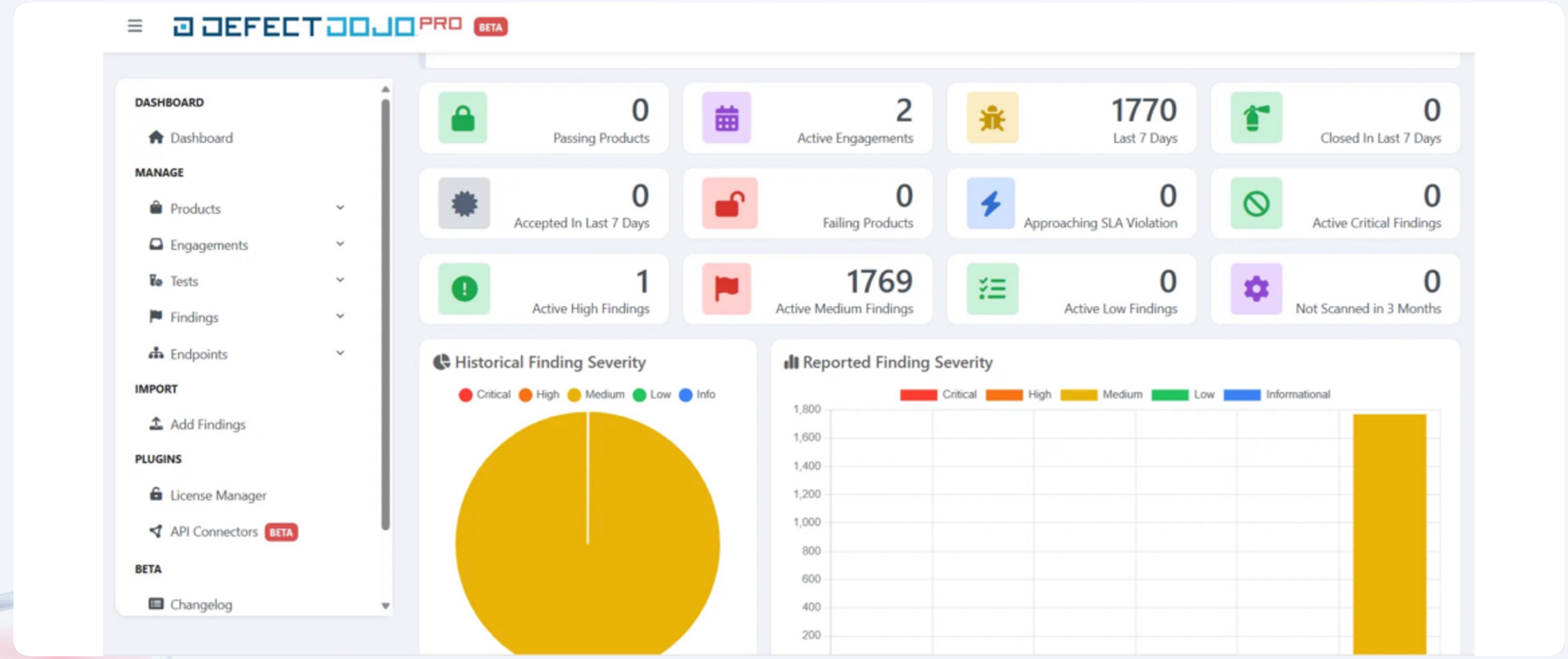
01 Используем минимальные наборы правил

02 Реализуем триаж

03 Согласуем SLA с разработчиками

04 **Создадим удобный дашборд**

Defect dojo



0%

95% 100%

И снова YAML Development

defectdojo.template.yml 5.09 KiB

```yaml
variables:
  DEFECTDOJO_ENGAGEMENT_PERIOD: 7
  DEFECTDOJO_ENGAGEMENT_STATUS: "Not Started"
  DEFECTDOJO_ENGAGEMENT_BUILD_SERVER: "null"
  DEFECTDOJO_ENGAGEMENT_SOURCE_CODE_MANAGEMENT_SERVER: "null"
  DEFECTDOJO_ENGAGEMENT_ORCHESTRATION_ENGINE: "null"
  DEFECTDOJO_ENGAGEMENT_DEDUPLICATION_ON_ENGAGEMENT: "false"
  DEFECTDOJO_ENGAGEMENT_THREAT_MODEL: "true"
  DEFECTDOJO_ENGAGEMENT_API_TEST: "true"
  DEFECTDOJO_ENGAGEMENT_PEN_TEST: "true"
  DEFECTDOJO_ENGAGEMENT_CHECK_LIST: "true"
  DEFECTDOJO_NOT_ON_MASTER: "false"
  DEFECTDOJO_SCAN_MINIMUM_SEVERITY: "Info"
  DEFECTDOJO_SCAN_ACTIVE: "true"
  DEFECTDOJO_SCAN_VERIFIED: "true"
  DEFECTDOJO_SCAN_CLOSE_OLD_FINDINGS: "true"
  DEFECTDOJO_SCAN_PUSH_TO_JIRA: "false"
```

И снова YAML Development

Напишем джобу для создания engagement

```
36  script:
37    - |
38        ENGAGEMENTID=`curl -k --fail --location --request POST "${DEFECTDOJO_URL}/api/v2/engagements/" \
39              --header "Authorization: Token ${DEFECTDOJO_TOKEN}" \
40              --header 'Content-Type: application/json' \
41                --data-raw "{
42                  \"tags\": [\"GITLAB-CI\"],
43                  \"name\": \"#${CI_PIPELINE_ID}\",
44                  \"description\": \"${CI_COMMIT_DESCRIPTION}\",
45                  \"version\": \"${CI_COMMIT_REF_NAME}\",
46                  \"first_contacted\": \"${TODAY}\",
47                  \"target_start\": \"${TODAY}\",
```

И снова YAML Development

Напишем джобу для загрузки отчета в engagement

```
script:
  - |
      curl -k --fail --location --request POST "${DEFECTDOJO_URL}/api/v2/import-scan/" \
          --header "Authorization: Token ${DEFECTDOJO_TOKEN}" \
          --form "scan_date=\"${TODAY}\"" \
          --form "minimum_severity=\"${DEFECTDOJO_SCAN_MINIMUM_SEVERITY}\"" \
          --form "active=\"${DEFECTDOJO_SCAN_ACTIVE}\"" \
          --form "verified=\"${DEFECTDOJO_SCAN_VERIFIED}\"" \
          --form "scan_type=\"${DEFECTDOJO_SCAN_TYPE}\"" \
          --form "engagement=\"${DEFECTDOJO_ENGAGEMENTID}\"" \
          --form "file=@${DEFECTDOJO_SCAN_FILE}" \
          --form "close_old_findings=\"${DEFECTDOJO_SCAN_CLOSE_OLD_FINDINGS}\"" \
          --form "push_to_jira=\"${DEFECTDOJO_SCAN_PUSH_TO_JIRA}\"" \
          --form "test_type=\"${DEFECTDOJO_SCAN_TEST_TYPE}\"" \
          --form "environment=\"${DEFECTDOJO_SCAN_ENVIRONMENT}\""
```

И снова YAML Development

Напишем интеграции с нашими сканерами

defectdojopublish.template.yml 7.50 KiB Blame

```yaml
gemnasium-defectdojo:
  extends: .defectdojo_upload
  needs: ["defectdojo_create_engagement", "gemnasium-dependency_scanning"]
  variables:
    DEFECTDOJO_SCAN_TYPE: "GitLab Dependency Scanning Report"  #DD param used to know report type
    DEFECTDOJO_SCAN_TEST_TYPE: "GitLab-CI Gemnasium"
    DEFECTDOJO_SCAN_FILE: "gl-dependency-scanning-report.json"
  rules:
    - if: $SEC_DISABLED
      when: never
    - if: $SEC_ENABLE_BRANCHES
      when: always
```

И снова YAML Development

```
31  kics-defectdojo:
32    extends: .defectdojo_upload
33    needs: ["defectdojo_create_engagement", "kics-iac-sast"]
34    variables:
35      DEFECTDOJO_SCAN_TYPE: "KICS Scan"
36      DEFECTDOJO_SCAN_TEST_TYPE: "GitLab-CI KICS Scan"
37      DEFECTDOJO_SCAN_FILE: "./kics-results.json"
```

Всё работает

Title / Type	Date	Lead	Total Findings	Active (Verified)	Mitigated	Duplicates	Notes	Reimports
Dependency Check Scan	Sept. 11, 2024 - Sept. 11, 2024		77	77 (77)	0	0		0
Dockle Scan	Sept. 11, 2024 - Sept. 11, 2024		2	2 (2)	0	0		0
Dockle Scan	Sept. 11, 2024 - Sept. 11, 2024		2	2 (2)	0	0		0
GitLab Dependency Scanning Report	Sept. 11, 2024 - Sept. 11, 2024		55	55 (55)	0	0		0
GitLab Secret Detection Report	Sept. 11, 2024 - Sept. 11, 2024		1	1 (1)	0	0		0
Semgrep Scan (GitLab SAST Report)	Sept. 11, 2024 - Sept. 11, 2024		53	53 (53)	0	0		0
Trivy Scan (GitLab SAST Report)	Sept. 11, 2024 - Sept. 11, 2024		45	45 (45)	0	0		0
Trivy Scan (GitLab SAST Report)	Sept. 11, 2024 - Sept. 11, 2024		138	6 (6)	132	0		0
kics Scan (GitLab SAST Report)	Sept. 11, 2024 - Sept. 11, 2024		217	217 (217)	0	0		0
njsscan Scan (GitLab SAST Report)	Sept. 11, 2024 - Sept. 11, 2024		35	35 (35)	0	0		0

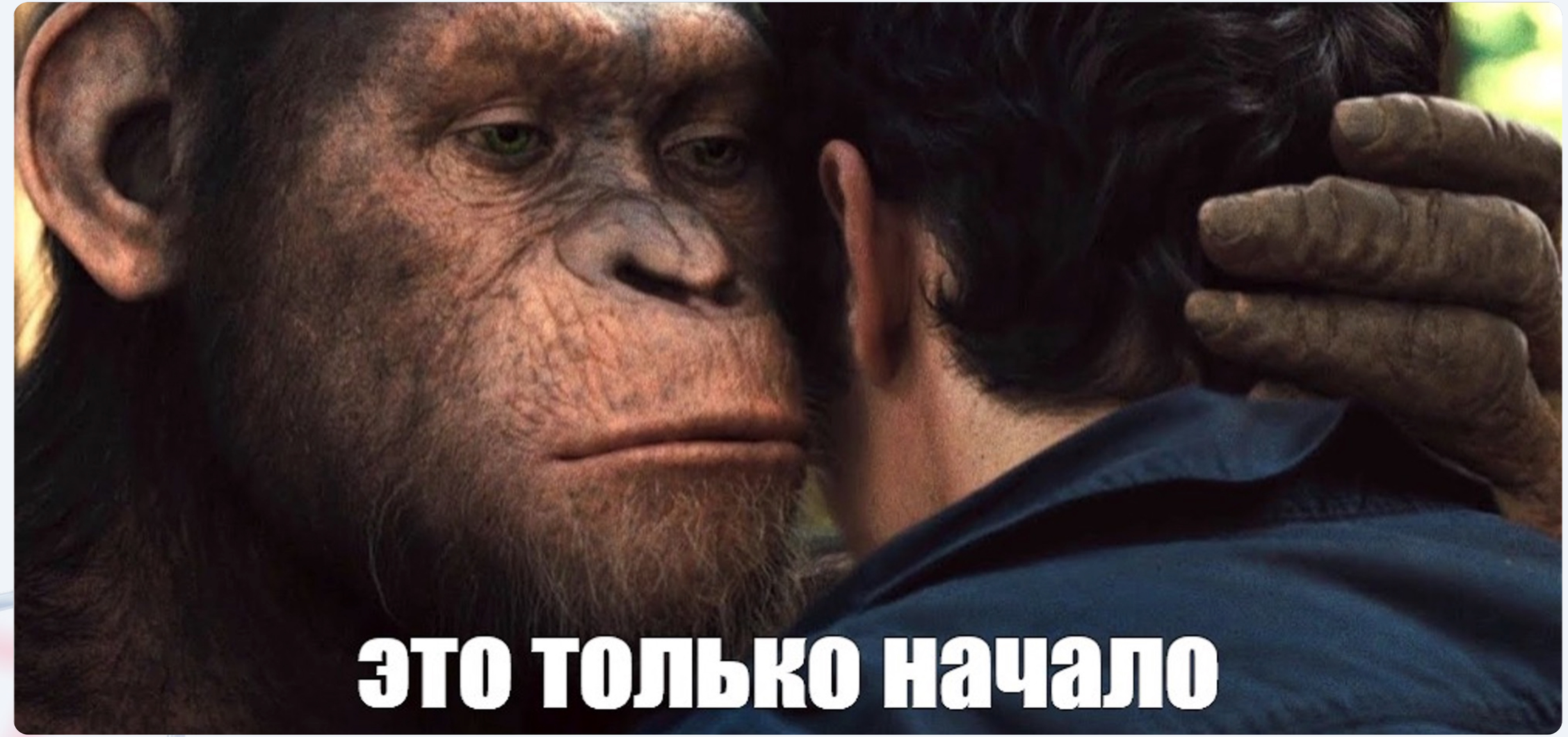
Showing entries 1 to 10 of 10

Page Size

0%

100%

И всё?



Secure by default

0%

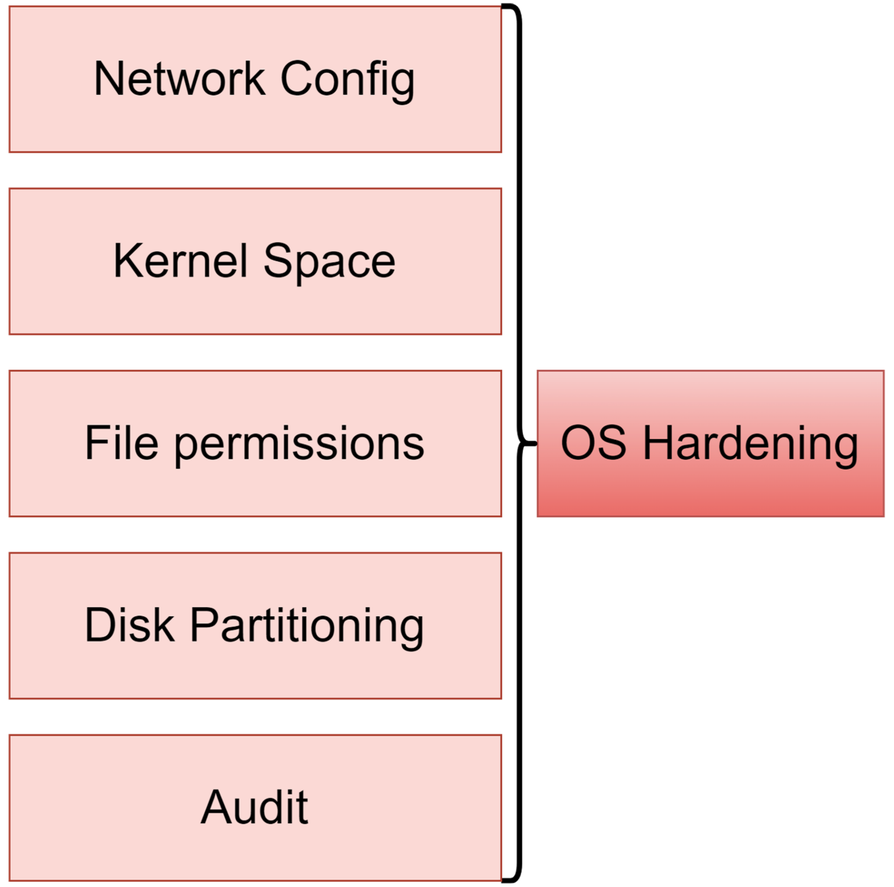
100%

110%

Secure by default

Hardening

01 Операционной системы



Secure by default

Hardening

01 Операционной системы

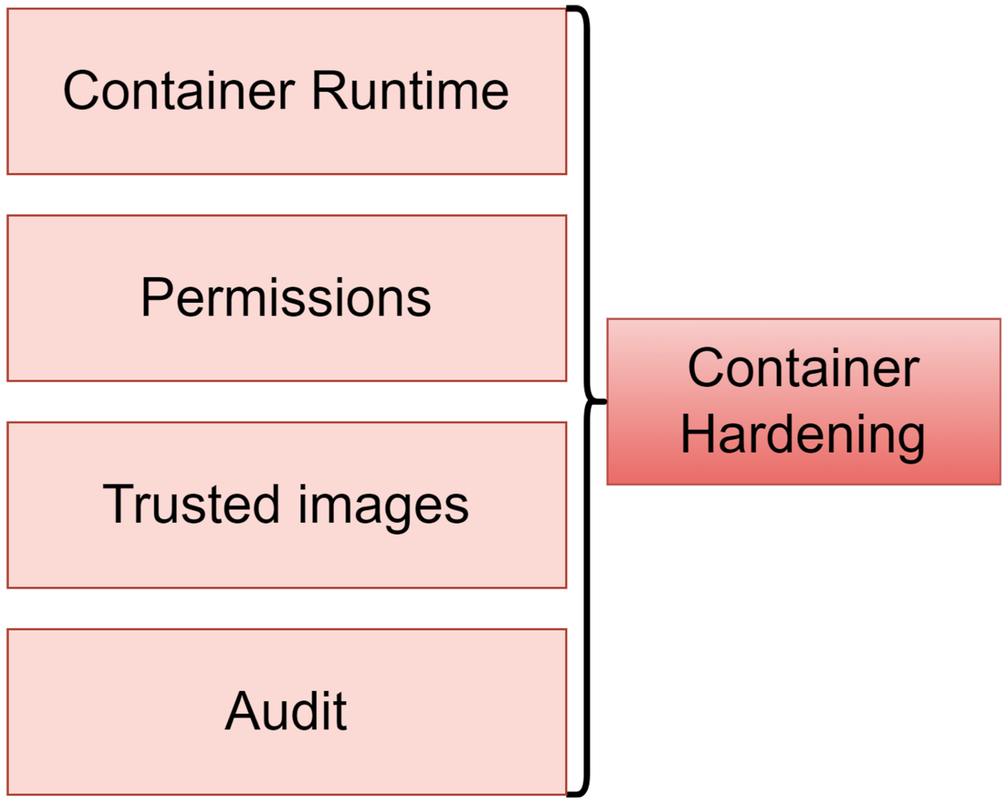
https://github.com/OpenSCAP/openscap

Secure by default

Hardening

01 Операционной системы

02 Контейнеров



Secure by default

Hardening

01 Операционной системы

02 Контейнеров

https://github.com/aquasecurity/trivy

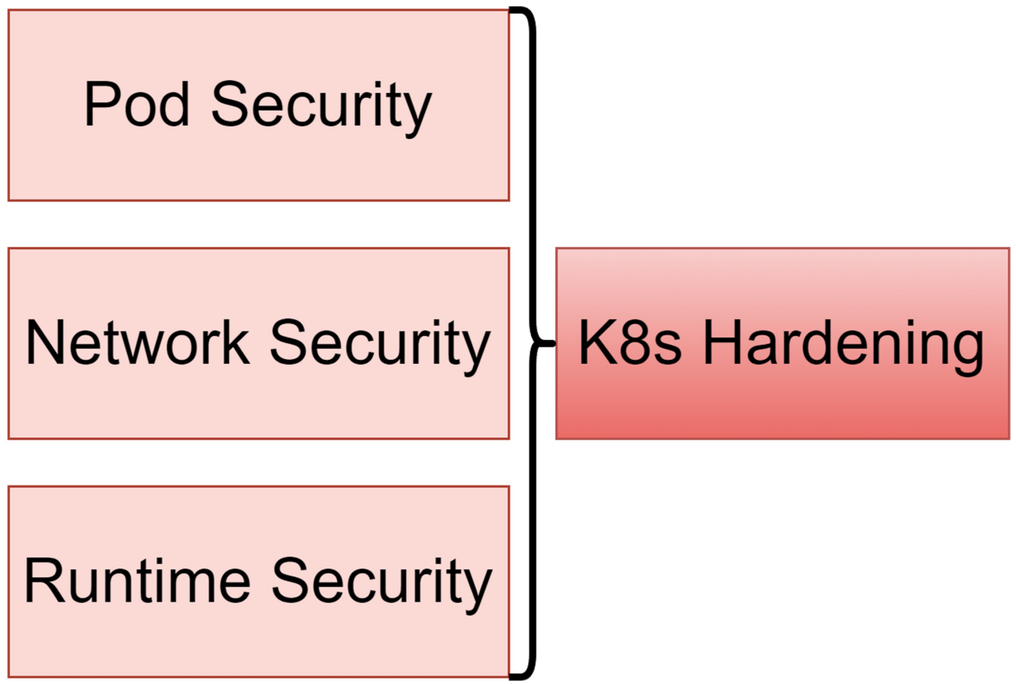
Secure by default

Hardening

01 Операционной системы

02 Контейнеров

03 Кластеров k8s



Secure by default

Hardening

01 Операционной системы

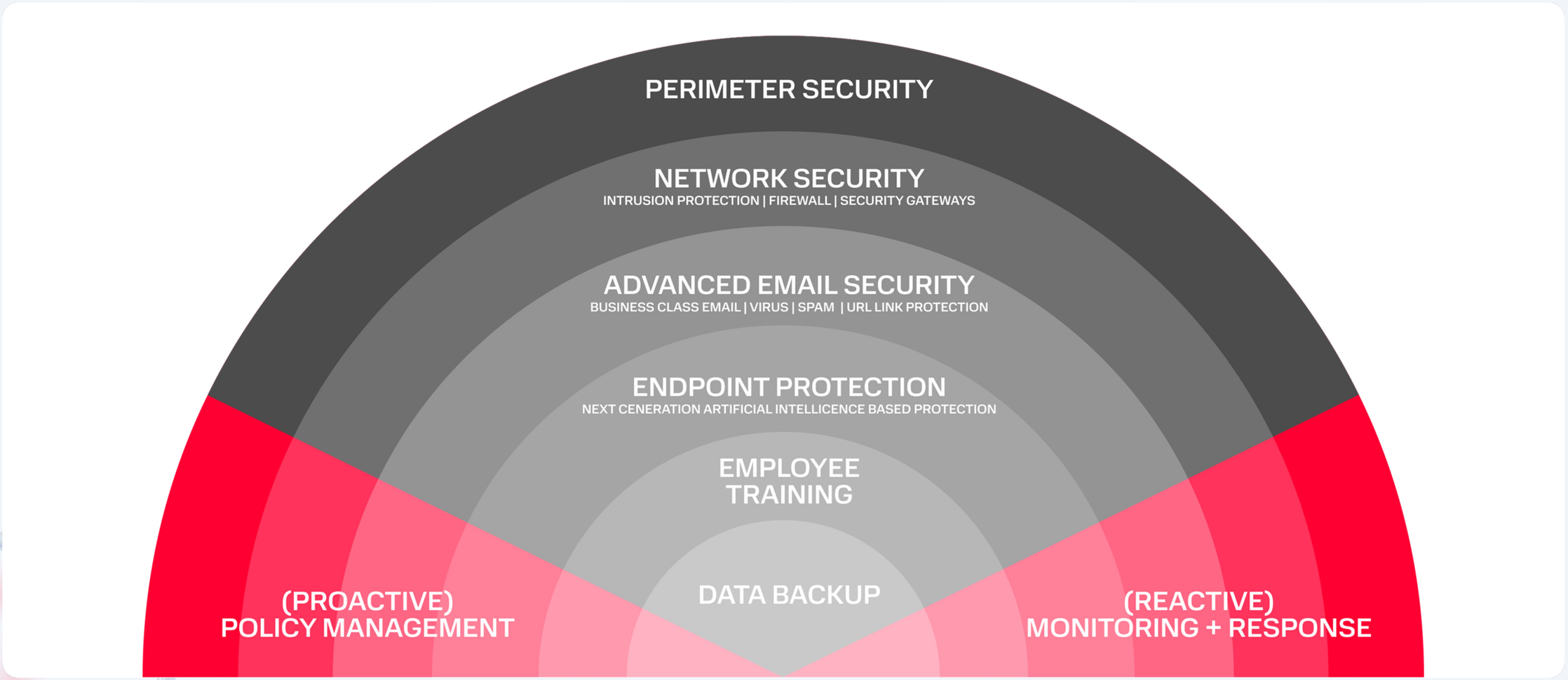
02 Контейнеров

03 Кластеров k8s

https://github.com/kubescape/kubescape

Defense in Depth

Эшелонированная защита



M W

S

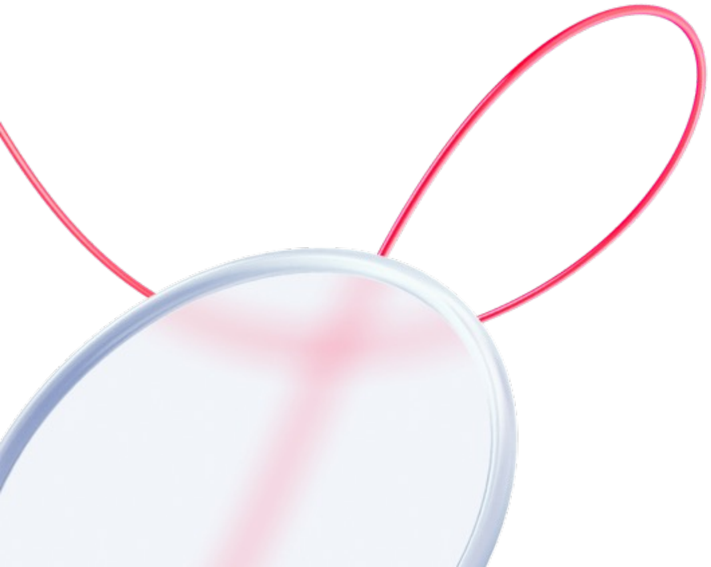
И остальные...

01 Fail Safe

02 Least Privilege

03 Secure by Design

04 Economy of Mechanism



M W

S

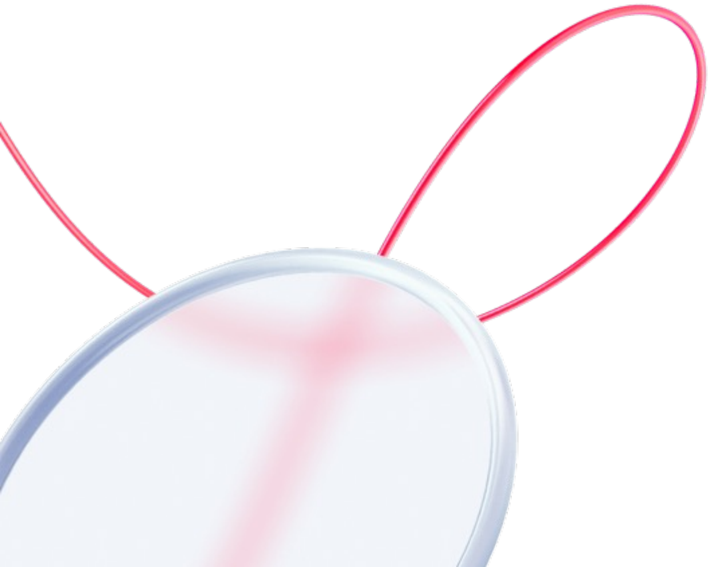
И остальные...

01 Fail Safe

02 Least Privilege

03 Secure by Design

04 Economy of Mechanism



Чего мы добились

01 Внедрили SAST

02 Внедрили Gitlab security templates

03 Собрали свой security template

04 Поняли что пора нанять DevSecOps

0% 100% 146%

ТЕПЕРЬ ВСЁ!