Security Bot

Release 0.1.0

Maxim Sokolov

Jun 22, 2023

CONTENTS

CHAPTER

ONE

GETTING STARTED

On this page, you will find all the necessary information to dive into the Security Bot (SecBot) project to

- set up the service and the documentation generator it uses,
- configure and integrate it with your service,
- ensure communication via API, and
- get familiar with the main concepts and limits.

Yet, we provide detailed descriptions and insights on separate pages of this documentation.

1.1 Prerequisites

Since SecBot is a Python application running in a container on Kubernetes, make sure that the relevant components and their packages are installed and available in your local environment.

Kubernetes-related:

- Docker
- Kubernetes
- Kubernetes Cluster
- Container registry, for example Docker Hub
- and other Containerization tools, for example Docker Compose

Python-related:

- Python
- PIP
- Poetry

Additionally, we employ

- Sphinx as a documentation generator and
- draw.io as a tool for creating schemes and diagrams.

1.2 Deployment

Follow these general steps to install and build the SecBot

1. Clone the repository:
    a. visit the project's repository to copy the URL under *Clone*
    b. run the `git clone` command to create a local copy

```
$ git clone path/to/project.git
```

2. Build and run the SecBot service.

```
$ docker-compose up --build
```

1.3 Service Configuration

The `/.env.dev` file defines the location, keys, and other parameters of the internal and external units SecBot communicates with: queues, databases, *Inputs, Scans, Outputs, and Notifiers*.

1. Review this file and make the necessary changes to it based on your environment's peculiarities, such as the variables within `GITLAB_CONFIGS`.
2. Save the file and rebuild the service.

```
$ docker-compose up --build
```

For test and other reasons, you can redefine any parameter in the `/.env.override` file for your separate sandbox environment. To do this,

1. Rename the original file of `/.env.override.example` accordingly
2. Specify the new values of existing variables there, for example modify `DEFECTDOJO__TOKEN=defectdojo_token`

```
# Excerpt from .env.override

...
DEFECTDOJO__TOKEN=my_personal_token
...
```

3. Save the file.
4. Rebuild the service.

```
$ docker-compose up --build
```

Note: For more detailed information on this topic, see *Configuration*.

1.4 Workflow Configuration

The `/app/config.yml` file defines the policies SecBot follows in its work: which Scans to launch to check input entities of a particular type, which Outputs to use to aggregate the Scans' results, and so on. You can take the original version and use it as is or update the file according to your needs. In the latter case, you will need stop and restart the service.

```
$ docker-compose stop
$ docker-compose up -d
```

Note: For more detailed information on this topic, see *Configuration*.

1.5 Integration

Since SecBot communicates with different units via their respective APIs and triggers in response to specific input events, you are expected to

- *obtain authorization with these units* (Inputs, Outputs, and Notifiers) and
- *specify triggers* on your development and distribution platform (Input), such as system hooks (or webhooks) for GitLab.

CHAPTER

TWO

CONFIGURATION

The content of this page is focused on the peculiarities and details of the service and workflow configuration files to let you make more informed decisions when customizing the Security Bot (SecBot) service for your needs.

2.1 Service Configuration

SecBot's configuration implies setting up all the

- internal (queues and databases) and
- external (Inputs, Scans, Outputs, and Notifiers) units

this service collaborates with and specifying other relevant parameters (metrics) as well.

Therefore, you need to review and update the respective `/.env.dev` file in advance to reflect your environment's peculiarities.

```
# Excerpt from .env.dev

...
CELERY_BROKER_URL=redis://redis:6379/0
CELERY_RESULT_BACKEND=redis://redis:6379/0

SECBOT_POSTGRES_DSN=postgresql+asyncpg://secbot:foobar@db:5432/secbot

GITLAB_CONFIGS=[
    {
        "host":"https://git.env.local/",        # GitLab's host (instance)
        "webhook_secret_token":"SecretStr",     # secret token used when a webhook is␣
↪being set up
        "auth_token":"SecretStr",               # token given to the user who will␣
↪communicate with the API to get check results
        "prefix":"GIT_LOCAL"                    # prefix used when a security_check_id␣
↪is being generated
    }
]

DEFECTDOJO__URL=https://defectdojo.env.local    # DefectDojo's host
DEFECTDOJO__TOKEN=defectdojo_token              # token given upon user registration to␣
↪communicate with the DefectDojo's API
DEFECTDOJO__USER=defectdojo_username            # registered user's name
DEFECTDOJO__USER_ID=10                          # registered user's ID
```

(continues on next page)

(continued from previous page)

```
SLACK_TOKEN=token_here                          # token given to the user that is␣
↪allowed to read Slack's channels
...
```

After that, save the file and rebuild the service.

```
$ docker-compose up --build
```

Additionally, according to the `/.gitignore` file, any parameter specified in `/env.dev` can be redefined in `/.env.override` for testing and other purposes.

```
# Excerpt from .gitignore

...
# Personal override env
.env.override
...
```

To do this,

1. Rename the original file of `/.env.override.example` accordingly
2. Specify the new values of existing variables there, for example modify `DEFECTDOJO__TOKEN=defectdojo_token`

```
# Excerpt from .env.override

...
DEFECTDOJO__TOKEN=my_personal_token
...
```

3. Save the file.
4. Rebuild the service.

2.2 Workflow Configuration

The workflow configuration is a set of policies according to which SecBot reacts to incoming events, processes the data, and yields the results. This configuration is based on the `app/config.yml` file that contains two sections: *components* and *jobs*.

The first section introduces the hired external units (Scans, Outputs, and Notifiers) which SecBot collaborates with. It might include a unit's name (handler name) and its settings: format of data it returns, URLs, keys, etc.

The following excerpt shows an example of three units belonging to different types. (The environment variable values refer to the `.env.dev` file.)

```
# Excerpt from app/config.yml

...
components:
    # Scan Gitleaks
    gitleaks:
```

(continues on next page)

(continued from previous page)

```
        handler_name: "gitleaks"
        config:
            format: "json"                      # data format in a response
    # Output DefectDojo
    defectdojo:
        handler_name: "defectdojo"
        env:
            url: "DEFECTDOJO__URL"              # host
            secret_key: "DEFECTDOJO__TOKEN"     # token given upon user registration to␣
↪communicate with the API
            user: "DEFECTDOJO__USER"            # registered user's name
            lead_id: "DEFECTDOJO__USER_ID"      # registered user's ID
    # Notifier Slack
    slack:
        handler_name: "slack"
        config:
            render_limit: 10                    # maximum number of lines (findings) in␣
↪a notification
            channel:                            # channels to report findings
                - test-sec-security-bot
                - my-personal-channel
        env:
            token: "SLACK_TOKEN"                # token given to the user that is␣
↪allowed to read the channels
...
```

Note: For now, these are the only components SecBot collaborates with. However, it must be sufficient for the first version of the product to let you assess its work.

The second section, "jobs", defines the policies SecBot follows in its work to yield the results. When a specific event matching the *rules* of a policy comes up, a processing plan (job) is created. It contains the necessary number of tasks to be sequentially executed by the relevant external units (components).

The following excerpts from the `app/config.yml` file show an example of one job to explain the idea.

```
# Excerpt from app/config.yml

...
jobs:
    # human-readable job name to be used as a reference in logs
    - name: Common merge request event

    # two-level identifier of an input entity
    rules:
        gitlab:                                         # first level
            event_type: "merge_request"                 # second level
            project.path_with_namespace: /gitlab-test/  # second level
```

, where

- **first level** is a development or distribution platform (Input) or any custom workflow name.
- **second level** is *Input entity* (input event) types and keys (JSON path strings) to apply checks and filtration within

the input events. For available keys, refer to the objects from a payload (request body). For example, `project.path_with_parameters` from `POST [host]/v1/gitlab/webhook` enables the filtering of input events originating from specific repositories.

Also, note that the arguments at the second level are joined with logical AND unless they are matching and thus mutually exclusive. That is, if, for example, `event_type: "tag_push"` and `event_type: "merge_request"` are specified in the same job, the last one will be taken.

```
# Excerpt from app/config.yml (continuation)

    # handlers to find leaks, vulnerabilities, and other security-related issues
    scans:
        - gitleaks

    # handlers to aggregate and normalize the Scans' results
    outputs:
        - defectdojo

    # handlers to report the Outputs' results
    notifications:
        - slack
...
```

Note: For now, only one job is allowed to cover a particular event. If your configuration implies that two or more jobs can be created to serve the same event, it will result in an error.

Information from the `app/config.yml` file is read once as soon as SecBot starts. Therefore, if you make any changes to it, you need to stop and restart the service to apply them.

```
$ docker-compose stop
$ docker-compose up -d
```

CHAPTER

THREE

INTEGRATION

On this page, you will find an adequate checklist and step-by-step instructions to ensure the successful integration of your service with the Security Bot (SecBot) solution.

3.1 Configuration Files

The service (`/.env.dev`) and workflow (`/app/config.yml`) *configuration files* are updated to meet your environment's peculiarities and issue processing needs.

If you have made additional alterations, rebuild SecBot and start it again.

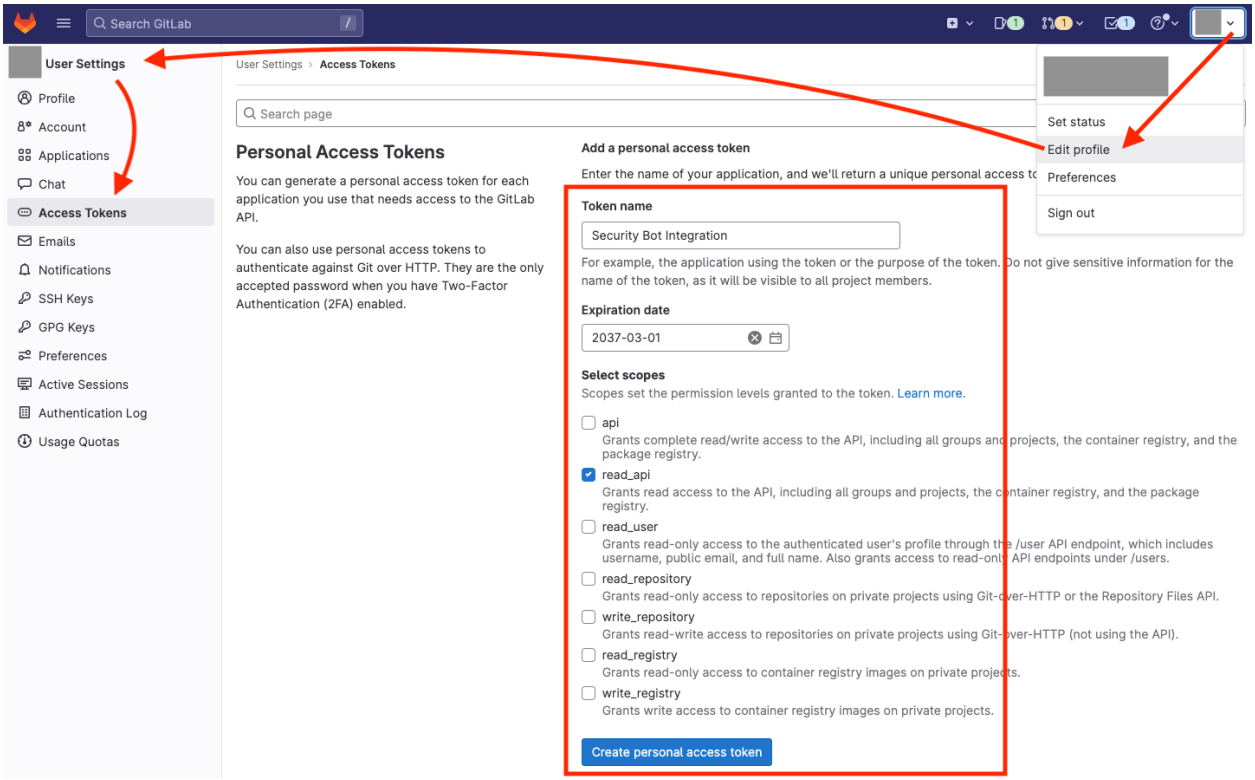
```
$ docker-compose stop
$ docker-compose up --build
$ docker-compose up -d
```

3.2 Authorization

All external units that SecBot communicates with via APIs require authorization. Therefore, the user that performs the communication should have the respective tokens.

For GitLab

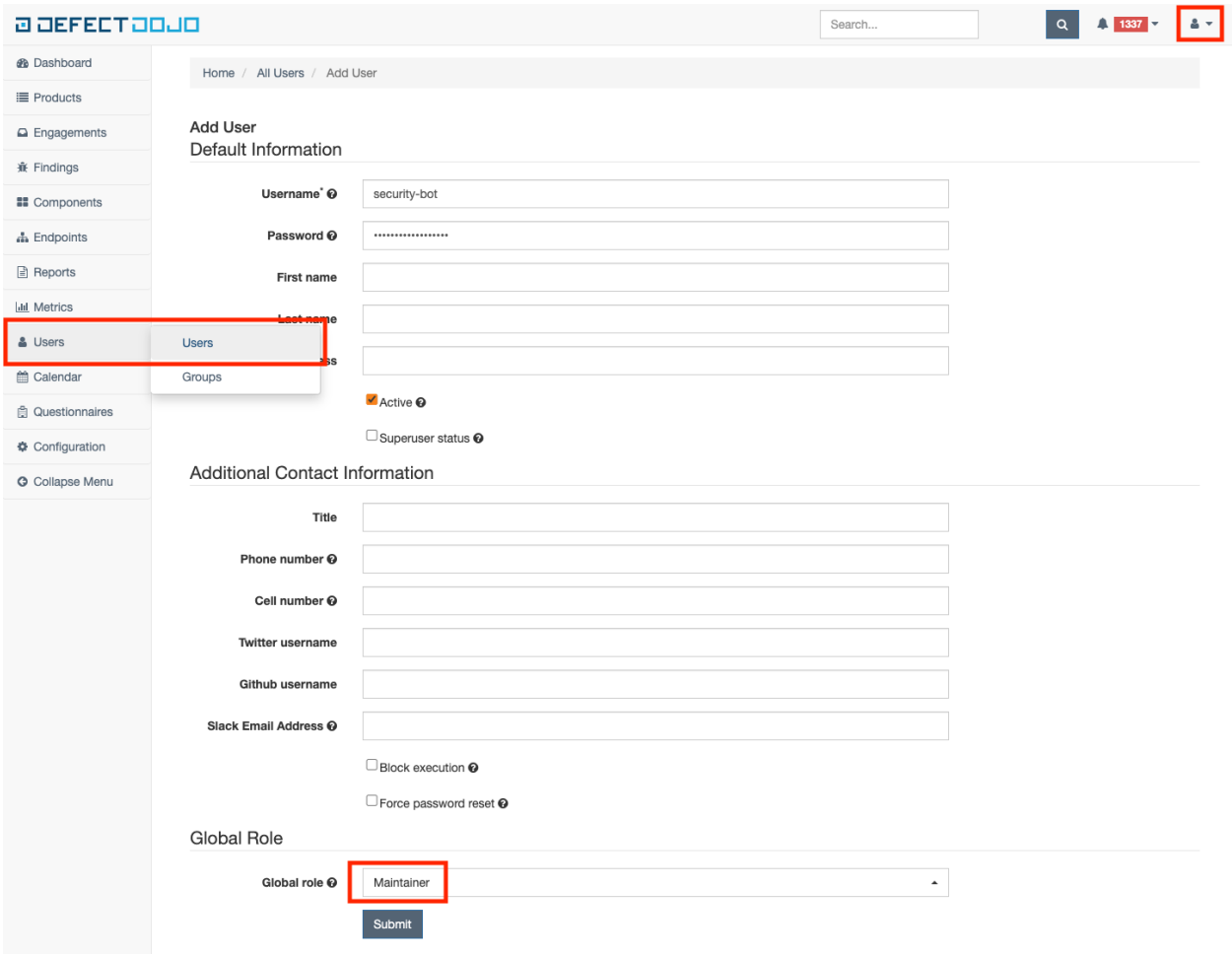
1. Sign in to your account
2. Click the profile icon in the upper-right corner and then click **Edit profile**
3. Click **Access Token** on the **User Settings** left-side menu
4. On the open page, enter your *Token name*, check the *Expiration date*, select `read_api` under **Select scopes**, and then click the *Create personal access token* button below



5. Copy *Your new personal access token* generated by GitLab and ensure you securely store it, as it will not be displayed again.

For DefectDojo

1. Log in under the admin's (superuser's) account
2. Point to the *Users* left-side menu and then click **Users**
3. On the open page, click the *Settings* toolbar button and then click **New User**
4. On the **Add User** page, enter the following parameters of the user that will communicate with the API:
   a. *Username* and *Password* under **Default Information**
   b. `Maintainer` as the *Global role* under **Global Role**
5. Click the *Submit* button and make sure that the *User has been added successfully*



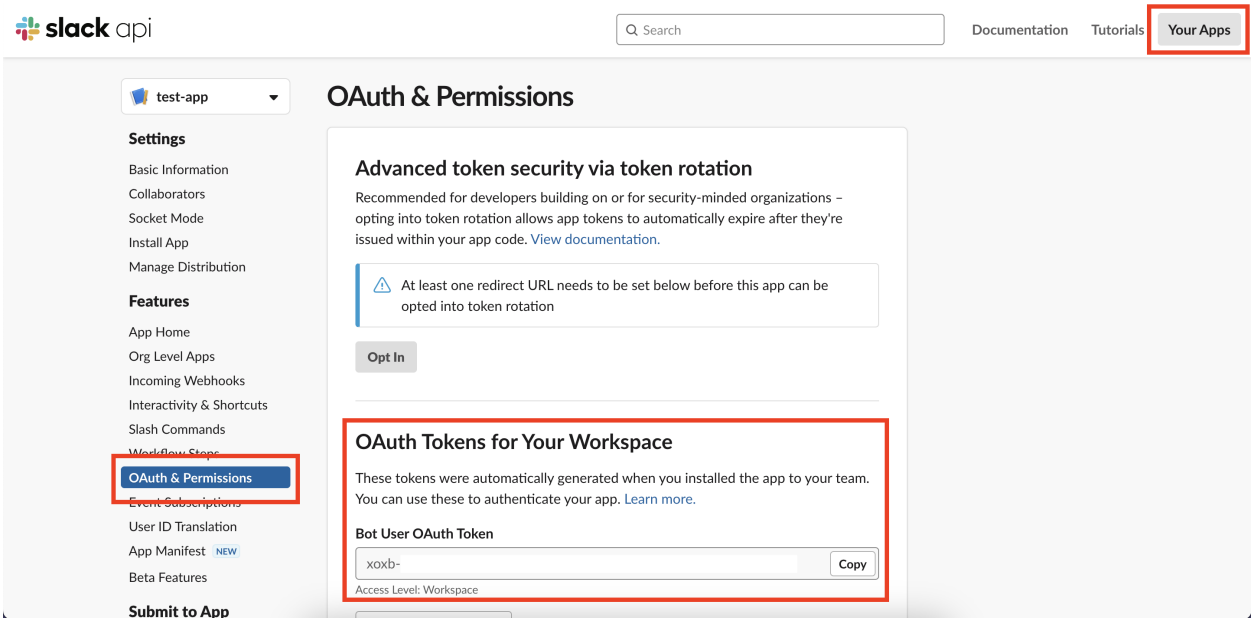
6. Log in under the added user's account
7. Click the profile icon in the upper-right corner and then click **API v2 Key**
8. On the open page, copy *Your current API key* generated by DefectDojo and ensure you securely store it.

For Slack

1. Go to `https://api.slack.com` to sign in to your workspace
2. Create an app; to do it,
    a. Click the *Your Apps* menu
    b. Click the *Create New App* button on the open page
    c. Choose **From scratch** in the open **Create an app** dialog box
    d. Enter your *App Name* and *Pick the workspace* where you want to install this app in the following **Name app & choose workspace** dialog box
    e. Click the *Create App* button
3. Click the **OAuth & Permissions** left-side menu under **Features** on the app's dashboard
4. Scroll down to the **Scopes** section and click the *Add an Oath Scope* button under **Bot Token Scopes** to select `chat:write`
5. Scroll up to the **OAuth Tokens for Your Workspace** section and click the *Install to Workspace* button

- Allow the app to access the workspace if requested

6. Copy your *Bot User OAuth Token* generated by Slack and ensure you securely store it.

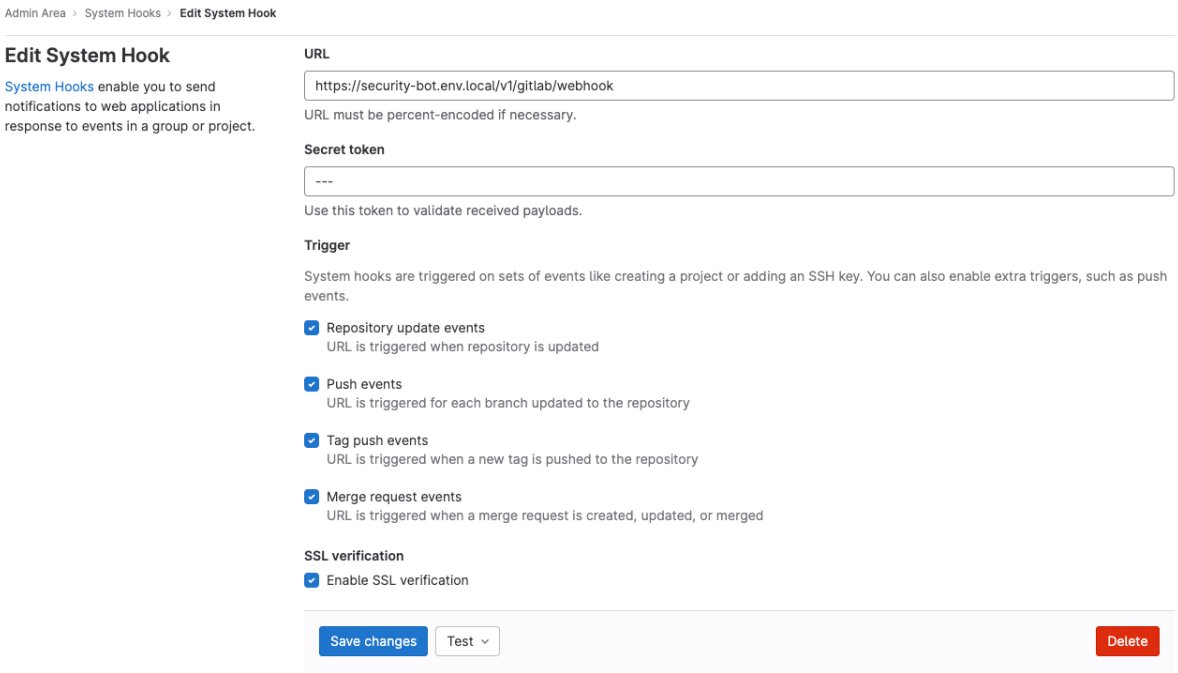


3.3 Input Entity Sources

The SecBot instance responsible for receiving requests to process data triggers as soon as a relevant input event comes up. Thus, you are expected to specify these triggers for supported development and distribution platforms (Inputs).

For GitLab

1. Sign in under an admin's account
2. Click the **System Hooks** left-side menu to add new or update existing system hooks
3. On the open page, enter in the **URL** text box the reference to the method used to receive information on changes made to the repository in your environment
   - *[host]/v1/gitlab/webhook*
4. Enter the *authentication token* for your requests in the **Secret token** text box
5. Select the types of input events you want to be processed under **Trigger**, for example, any supported *Input entity* type like `Push events`, `Tag push events`, and `Merge request events`



6. Click the *Add system hook* or *Save changes* button below.

3.4 API

Communication with SecBot's API involves providing input entities or receiving check results via the dedicated endpoints (instances). Follow

- `[host]:5000/docs` and
- `[host]:5001/docs`, respectively.

In the first case, mind the

1. *Input entity* (input event) type you will specify as the `x-gitlab-event` header parameter and the
2. respective payload in the request body.

A specific result is retrieved by *security_check_id*, which is formed by concatenating the following pieces:

1. input platform (e.g. git) prefix,
2. sha256 of the project path, and
3. complete commit hash.

```
# security_check_id example

GIT_LOCAL_d42052411d2729e637980c355cf6a8ea8e41b8688b98c34a125b71b7f2c7f76e
```

3.5 Pipeline

Integration of SecBot into your pipeline as an additional stage is an option we suggest that you consider. Depending on the status received upon checks, this stage might

- get passed ('success'),
- stay pending ('not_started' or 'in_progress'), or
- fail ('error' or 'fail').

The following excerpt demonstrates a comprehensive example of how this integration can be implemented.

```
# Excerpt from .../pipeline.yml

...
.gate-sec-scripts:
  before_script:
    - apk add curl jq
    - SECURITY_CHECK_URL="https://[gateway_url]/v1/security/gitlab/check"
    - SECURITY_CHECK_UID="GIT_LOCAL_$(echo -n "${CI_SERVER_HOST}:${CI_PROJECT_PATH}_${CI_
↪COMMIT_SHA}" | sha256sum | head -c64)"
  script:
    - SECURITY_CHECK_STATUS=$(curl -k -s -w " %{http_code}" $SECURITY_CHECK_URL/$
↪{SECURITY_CHECK_UID})
    - SECURITY_CHECK_STATUS_JSON=$(echo $SECURITY_CHECK_STATUS | awk '{print $1}')
    - SECURITY_CHECK_STATUS_CODE=$(echo $SECURITY_CHECK_STATUS | awk '{print $2}')
    - |
      if [ "$SECURITY_CHECK_STATUS_CODE" != "200" ]; then
        echo " Something went wrong, status: $SECURITY_CHECK_STATUS"
        exit 1
      fi
    - SECURITY_CHECK_STATUS_JSON_STATUS_DESCRIPTION=''
    - SECURITY_CHECK_STATUS_JSON_STATUS=$(echo $SECURITY_CHECK_STATUS_JSON | jq -r '.
↪status')
    - |
      if [ $SECURITY_CHECK_STATUS_JSON_STATUS = "fail" ]; then
        SECURITY_CHECK_STATUS_JSON_STATUS_DESCRIPTION="--> Vulnerabilities found"
      elif [ $SECURITY_CHECK_STATUS_JSON_STATUS = "success" ]; then
        SECURITY_CHECK_STATUS_JSON_STATUS_DESCRIPTION="--> No vulnerabilities found"
      fi
    - echo " Response Code --> $SECURITY_CHECK_STATUS_CODE"
    - echo " Status --> $SECURITY_CHECK_STATUS_JSON_STATUS $SECURITY_CHECK_STATUS_JSON_
↪STATUS_DESCRIPTION"
...
```

CHAPTER

FOUR

GLOSSARY AND INVENTORY

On this page, you will find the

- brief explanations of the units SecBot's architecture is based on,
- lists of those supported and used in configuration, and
- descriptions of other related concepts.

Input
: Input is a code repository, storage, or development or distribution platform, such as GitLab or Docker Registry, changes to which need extended security-related validation.

Input	Source
`gitlab`	GitLab Docs

Input entity
: Input entity (or input event) is a substantial amount of data (payload) to be validated. This data can be filtered out based on some configuration rules so that only part of it is actually checked. You can specify one or more of the event types we support (see the following table) and any other keys (JSON paths) of your choice.

Event type	Source
`push`	Webhook events: push events
`tag_push`	Webhook events: tag events
`merge_request`	Webhook events: merge request events

```
# Excerpt from /app/config.yml

...
jobs:
  - name: Common merge request event
    rules:
      gitlab:                           # reserved name (Input)
        event_type: "merge_request"     # one of the filtering parameters (Event␣
↪type)
...
```

Scan
: Scan is an external code analysis tool for applying the DevOps and security best practices to development and

integration flows. It, for example, can detect hardcoded secrets (passwords, API keys, or tokens in Git repositories) or evaluate how certain changes might affect the overall quality or performance of your application. The result of its work is raw defect data to be passed to Outputs.

Scan	Source
`gitleaks`	Gitleaks on GitHub

Output
Output is an external defect management system specially integrated with SecBot to aggregate the check results from different Scans, merge the duplicates, and do other relevant things to prepare a normalized readable report for Notifiers. A piece of this report (problem, vulnerability, or any other security issue) is called "finding."

Output	Source
`defectdojo`	DefectDojo on GitHub

Findings
For findings, see "Output."

Notifier
Notifier (referred to as "notification" in the */app/config.py* file) is an instant messaging program integrated with SecBot to inform interested parties of detected security issues (findings).

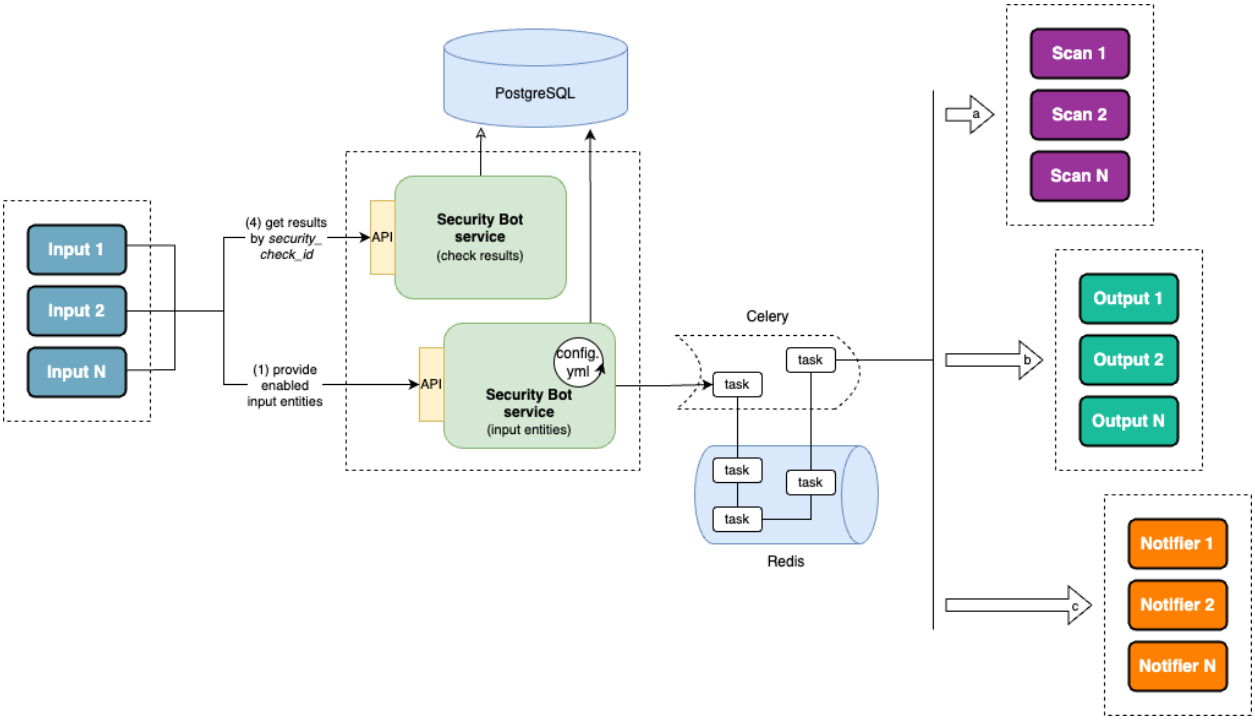
Notifier	Source
`slack`	Slack Website

Job
Job is three sets of tasks, at least one for a Scan, one for an Output, and one for a Notifier, to be executed sequentially to process a particular input entity type and yield the relevant results (findings).

The **Security Bot** (SecBot) is an orchestration service designed to communicate with various external units (see the *following scheme*) to detect security-related issues in developers' code. It can be implemented as an extra pipeline stage to be passed (along with linting, unit-tests, and build) or used in any other way.

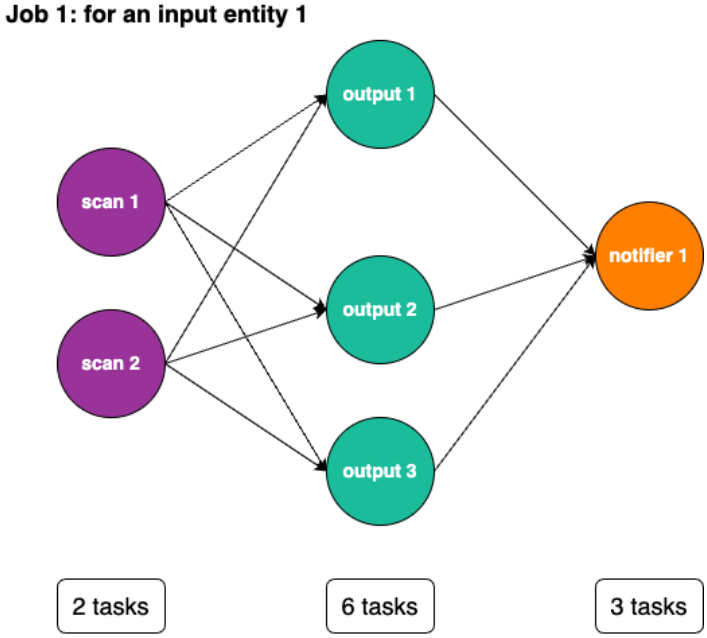
In its work, this service

1. receives from development and distribution platforms ("Inputs") information on changes that a software engineer contributes ("input entity")
2. based on its configuration and the input entity's type, draws up a processing plan ("job"; see an example of it *later*)
3. according to this plan, creates a necessary number of tasks for different units to be successively performed to
    a. scan the input entity with code analysis tools ("Scans")
    b. aggregate the found security issues from Scans, merge duplicates, and do other relevant things with defect management systems ("Outputs")
    c. inform the interested parties of the results by means of instant messaging ("Notifiers")
4. provides the "Input" platforms with the check results (status) on request. (Based on this status—"success" or "fail"—the changes being contributed are allowed or blocked.)



Note: As the scheme suggests, SecBot is split into two instances running in separate containers to ensure high availability and distribute the load. One instance is responsible for receiving requests to process data, whereas the other is dedicated to providing the results of this processing.

The following example of a processing plan, presented as a graph, implies that SecBot's job is configured to use two Scans, three Outputs, and one Notifier. The overall number of tasks is 11.



INDEX