Proposal for GSoC 2023

Project Name:

Decentralized vulnerability data peer-review

https://www.tdcommons.org/cgi/viewcontent.cgi?article=6738&context=dpubs_series

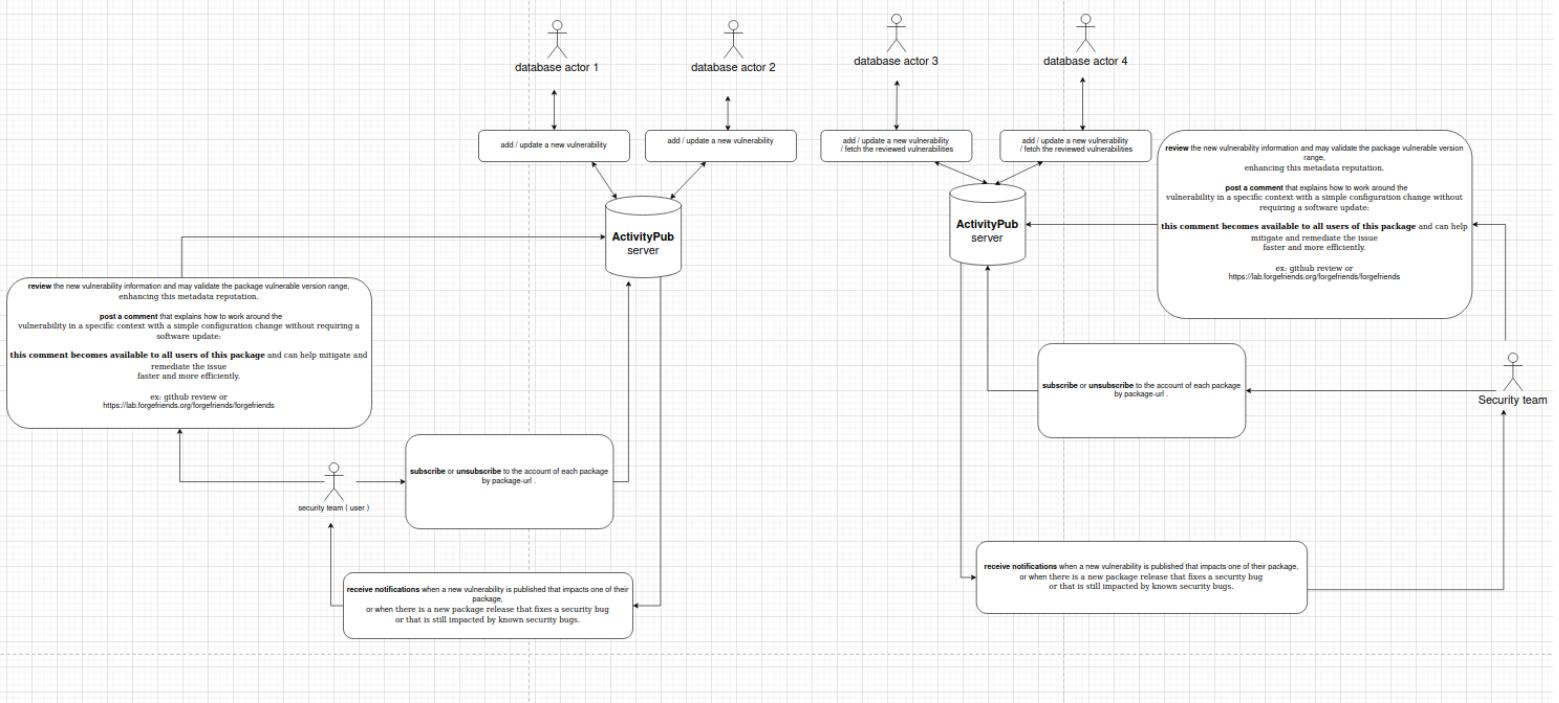
Project Description:

Software packages vulnerabilities:

let's say we have a security team that wants to track new vulnerabilities in the open source software packages, the security team subscribes to the account of each package by package-url

then security teams review the new vulnerability information and validate the package vulnerable version range of x database by posting reviews and comments. Every project could get its own ActivityPub account, typically identified by CVE. The security teams could get their own using any ActivityPub server.

Basic Architecture:



1 - Multiple independent user systems consuming and producing metadata.
producing:
let's say we have a vulnerablecode database locally and privately installed
we extract the vulnerabilities details (improver result) from postgres
then the database actor pushes this result in the git repository
consuming:
The security teams can fetch the vulnerabilities details.
they could also contribute by adding a review and notes

2 - multiple distributed and versioned document databases
Every activitypub server must have Git on the Server (Smart HTTP)

Every database actor has a git repository in the activitypub server
and has the right to update the repository. (add/update a new vulnerability)
https://github.com/dvdotsenko/git_http_backend.py

3 - advertising, dissemination, notification, and sharing federated system

The security team can fetch any file, and add reviews, and notes.

The security team can subscribe to purl and get a notification if purl is affected by a new vulnerability.

Activity Vocabulary:

Actor objects:

- security team type Person
- database type Organization

Actor objects *security team: MUST* have (inbox, outbox)
Inbox, outbox *MUST* be an OrderedCollection
The security team SHOULD have *(* following).
The database SHOULD have *(* followers).

Basic Activity Types:
https://www.w3.org/TR/activitystreams-vocabulary/#activity-types

database actor:

- *Create* a new vulnerability (using git add a new commit)
- *Update* a new vulnerability (using git add a new commit)

Security team actor:

- *Follow* a package-urls of x database
- *Undo* following a package-url
- *Create* a new *review (Collection of notes)*
- *Like/Dislike a review*

Object and Link Types:

- Notes
- Image

Client to Server Interactions:

Clients *MUST* discover the URL of the actor's outbox from their profile and then *MUST* make an HTTP POST request to this URL with the Content-Type of

application/ld+json; profile="https://www.w3.org/ns/activitystreams"

The request *MUST* be authenticated with the credentials of the user to whom the outbox belongs.

The body of the POST request *MUST* contain a single Activity

Reviewing Vulnerability:

let's say the security team wants to review a vulnerability the security team goes to the git repository ui click on a vulnerability file (VCID-wk4p-pp8w-aaag.json)

a list of all vulnerability reviews will show up and the status of the reviews then the security team clicks the create review button and adds a new review.

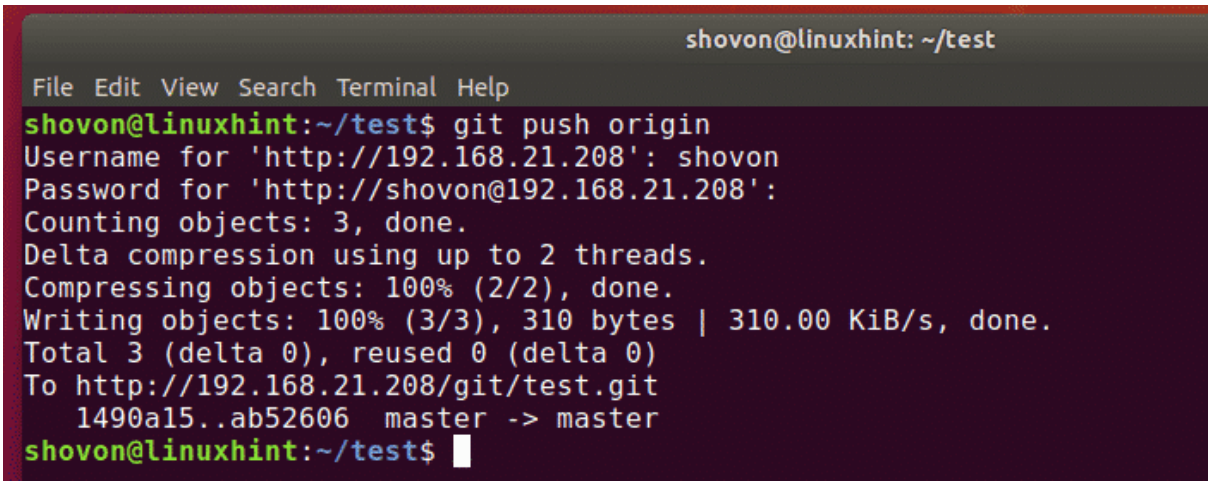
then the database actor accepts/rejects the review
If the review gets accepted, the review data is merged

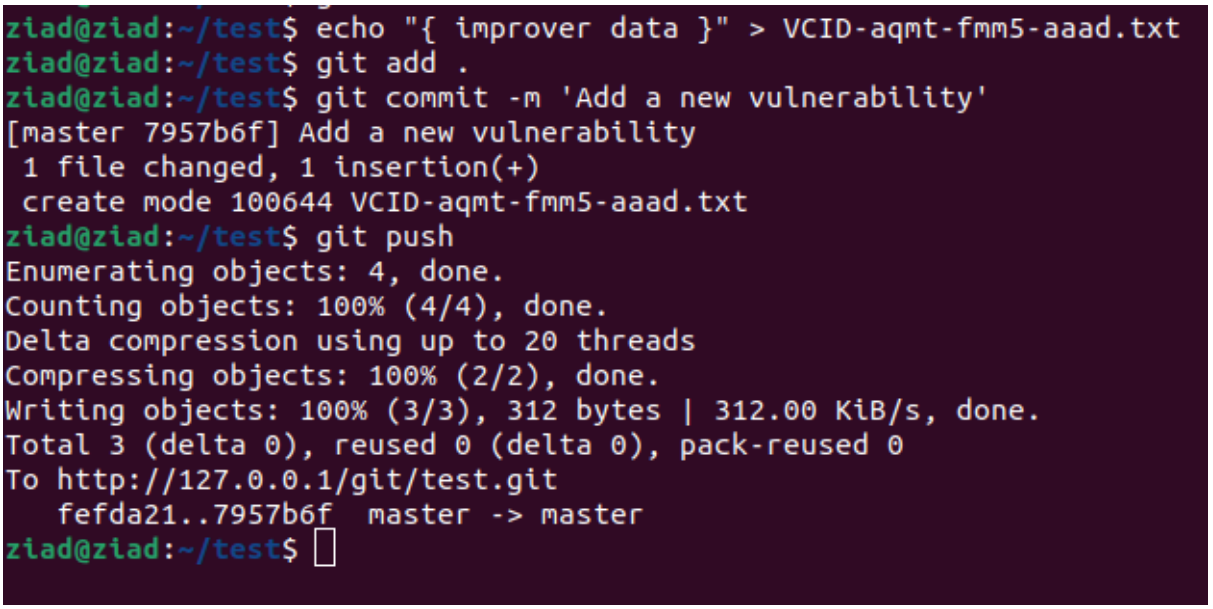
with review status enum types**: Accept, Reject, Pending**

notes: a collection of comments (small discussion form) .

Sync between database actors:

database actor has access to every git repository he creates .
database actors can (pull , fetch , push , …) and he is responsible for sync repo but can’t edit others' git repositories .
(enable anonymous read access but authentication and authorization write access)
ex:





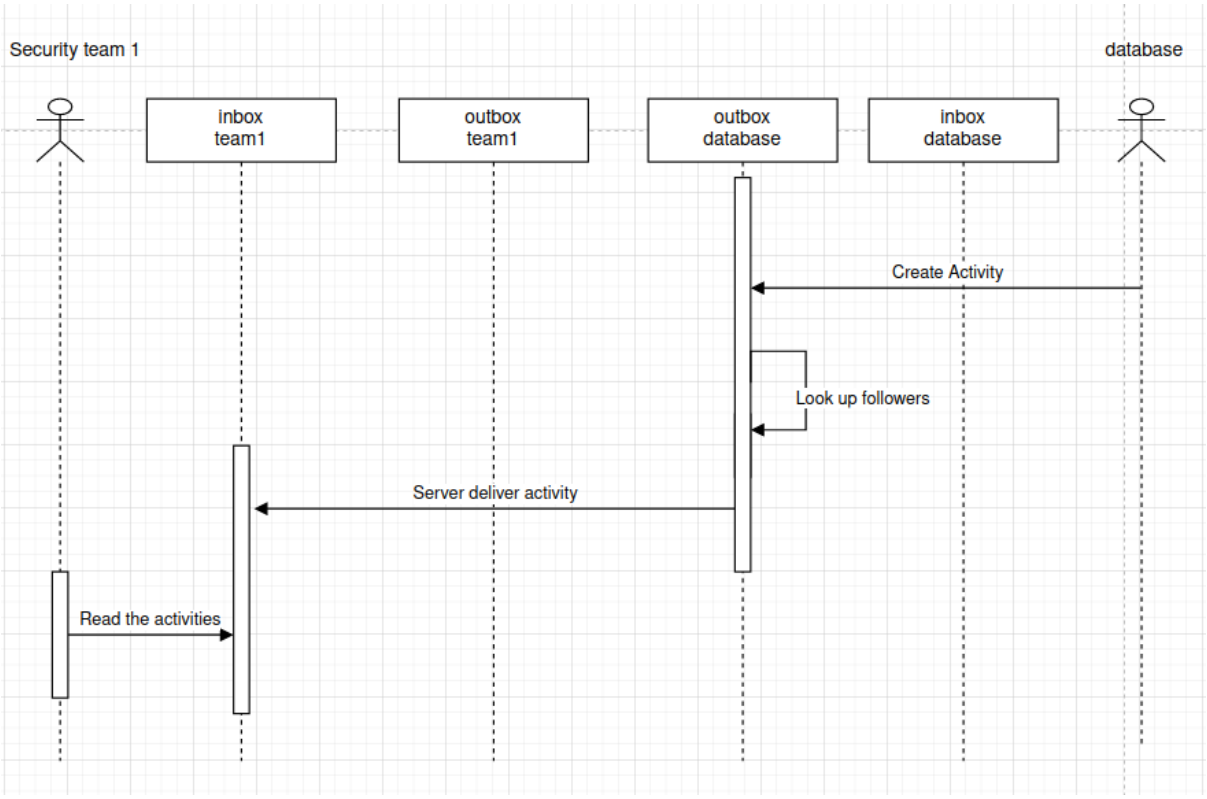
```
ziad@ziad:~$ git clone http://127.0.0.1/git/test.git
Cloning into 'test'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
```

https://linuxhint.com/git_server_http_ubuntu/

https://manpages.ubuntu.com/manpages/trusty/man1/git-http-backend.1.html

https://github.com/dvdotsenko/git_http_backend.py

Receive notifications when a database creates a new activity:



the database creates a new vulnerability (add a new commit)

the activitypub server checks if the database affected purl followed by a security team,
then the server sends a notification to the inbox of the security team
also maybe send an email to inform the security team .

Following a package-url :

https://github.com/w3c/activitypub/blob/gh-pages/userstories/following-a-person.md

let's say the package URL is pkg:maven/org.apache.commons/io@1.3.4/

The security team decided to subscribe to this purl.
the security team submits a post to the database outbox declaring that he would like to subscribe to this (package-url)

Server to Server Interactions:

POST requests *MUST* be made with a Content-Type of application/ld+json; profile="https://www.w3.org/ns/activitystreams"

GET requests with an Accept header of application/ld+json; profile="https://www.w3.org/ns/activitystreams"

Let's say we have two activity pub servers.
(security team 1, database 1) use server 1.
(security team 2, database 2) use server 2.
security team 1 follows the purls of database 2 . (security team 1 sends a request to server 1, server 1 sends a request to server 2 asks about the database 2 inbox/outbox using web finger.)
Then the security team sends the following request to the inbox.

Authentication and Authorization:

there are no strongly agreed upon mechanisms for authentication
https://www.w3.org/TR/activitypub/#authorization
Some possible directions
https://www.w3.org/wiki/SocialCG/ActivityPub/Authentication_Authorization

Client to Server
OAuth 2.0
Server to Server:
Signing requests using HTTP Signatures
https://datatracker.ietf.org/doc/html/draft-cavage-http-signatures-10

ER Diagram

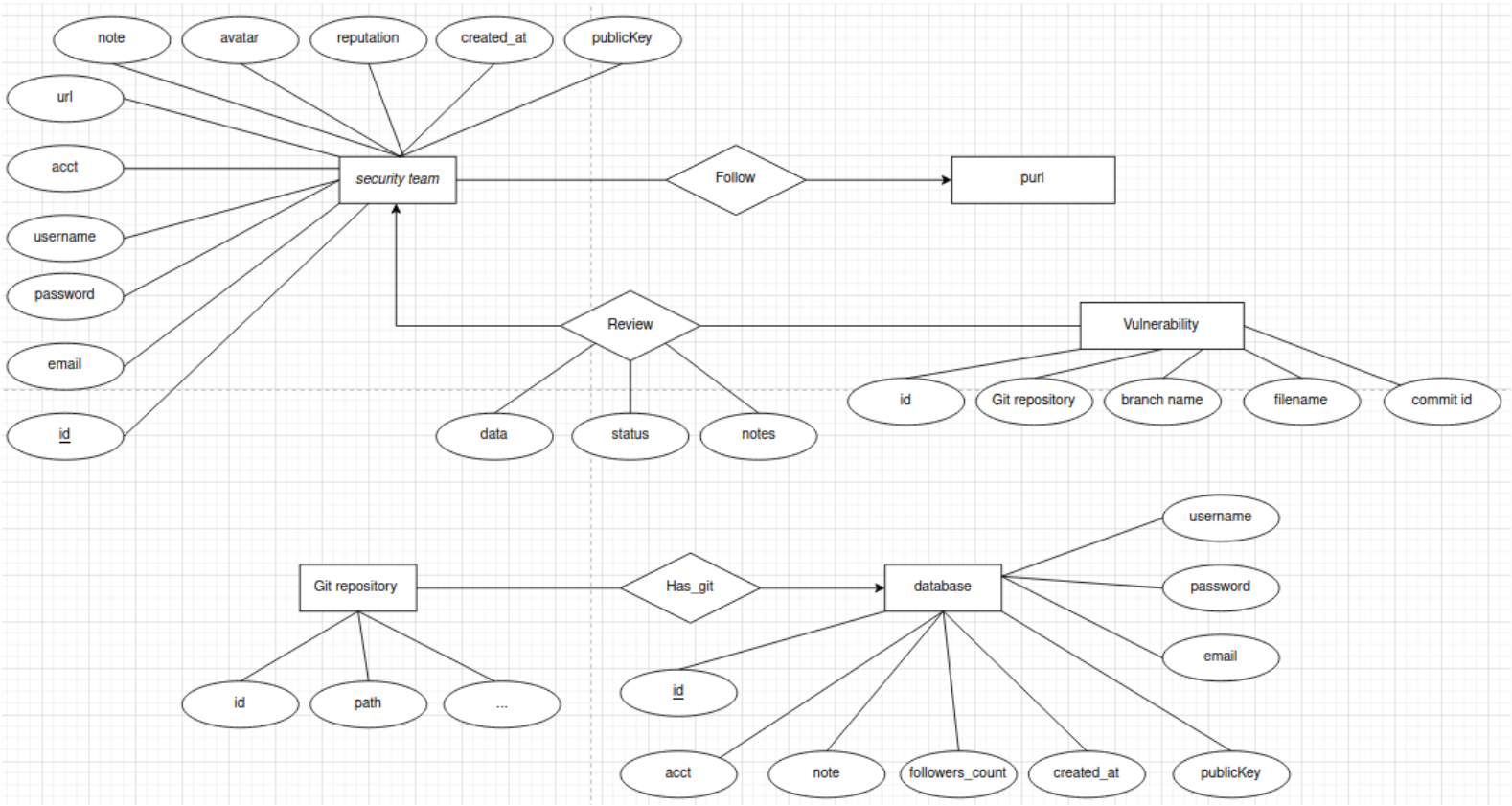


Table 1: Security team entity

id	The account id header
username	The username of the account, not including domain
acct	The Webfinger account URI. Equal to username for local users, or username@domain for remote users.
url	The location of the team's profile page.
email	the security team email
password	the security team password
note	The profile description.
avatar	The profile image
reputation	if someone like your review you will get +1, dislike : -1
created_at	When the account was created
publicKey	The profile public key

Table 2: **Database entity:**

id	The database id header
username	The username of the account, not including domain
email	The database email
password	The database password
acct	The Webfinger account.
note	The database description.
followers_count	The reported followers of this profile.
created_at	When the account was created
publicKey	The database public key

Table 3: **Vulnerability:**
Vulnerability(id#, git repository, branch name, filename)

Table 4: **Follow:**
Follow(id#, security team id, purl)

Table 5: **Review:**
Review(id#, security team id, vulnerability id, data, notes, status)

Table 5: **Has_git:**
Has_git(id#, security team id, vulnerability id, data, ….)

Example of fetch security team profile URL:

GET /team/vurnablecode/
Host: example.com
Content-Type: text/html

```
<!DOCTYPE hml>
<html>
  <head>
    <script type="application/ld+json">
     {
      "@context": [
        "http://www.w3.org/ns/activitystreams",
        "http://www.w3.org/ns/activitypub"
      ],
     "@type": "Person", # Team
     "@id": "https://vurnablecode.example/team/vurnablecode/",
     "following": "https://vurnablecode.example/api/team/vurnablecode/following",
     "inbox": "https://vurnablecode.example/api/team/vurnablecode/inbox",
     "outbox": "https://vurnablecode.example/api/team/vurnablecode/outbox",
      ....
     "icon": [
     "https://example.com/image/avatar"
      ] ,

      ...
     }
    </script>
  </head>
  <body>
    <!-- Content goes here! --!>
  </body>
</html>
```

Following a package-url Response Example:

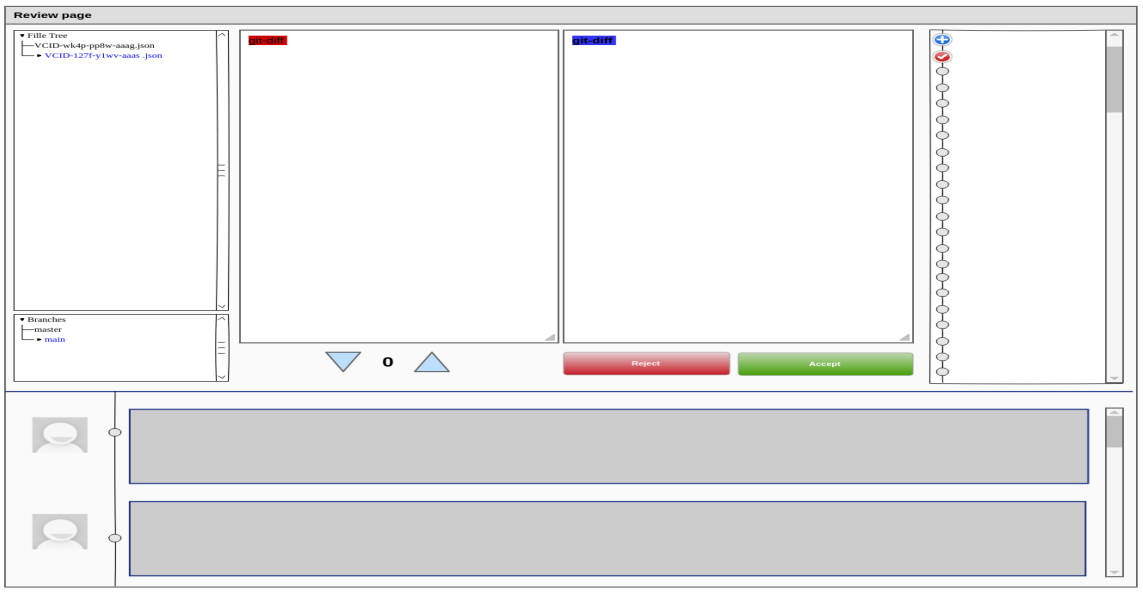
POST /api/team/outbox HTTP/1.1
Host: example.com
Content-Type: application/activitystreams+json
Authorization: Bearer xx-bearer-token-here-xx

{

{

"@context": "http://www.w3.org/ns/activitystreams",

"@type": "Follow",



"actor": {

"@type": "Person",, # team

"@id": "https://example.com/team/vcio/",

},

"object": {

"@type": "Notes",

....

},

"to": [{

"@type": "Organization", # database

"@id": "https://example.com/team/vurnablecode/"

}]

....

}

Generate public and private key :

>> openssl genrsa -out private.pem 2048
>> openssl rsa -in private.pem -outform PEM -pubout -out public.pem

Server to Server Interactions Endpoints	Method	Response Message
/.well-known/webfinger?resource=acct:team@example.com	GET	{ "subject": "acct:vcio@example.coml", "aliases": ["https://mastodon.social/@vcio", "https://mastodon.social/team/vcio"], "links": [{ "rel": "self", "type": "application/activity+json", "href": "https://example.com/team/vcip" }] }
/team	GET	{ "@context": ["https://www.w3.org/ns/activitystreams", "https://w3id.org/security/v1"], "id": "https://example.com/team/vcio", "type": "Person", "following": "https://example.com/team/vcio/following", "followers": "https://example.com/team/vcio/followers", "inbox": "https://example.com/team/vcio/inbox", "outbox": "https://example.com/team/vcio/outbox", "name": "vcio", "url": "https://example.com/@vcio", "publicKey": { "id": "https://example.com/team/vcio#main-key", "owner": "https://example.com/team/vcio", "publicKeyPem": "PUBLIC KEY" }, … }
/database	GET	{ "@context": ["https://www.w3.org/ns/activitystreams", "https://w3id.org/security/v1"], "id": "https://example.com/database/vcio", "type": "Person",

		"following": "https://example.com/team/vcio/following", "followers": "https://example.com/team/vcio/followers", "inbox": "https://example.com/team/vcio/inbox", "outbox": "https://example.com/team/vcio/outbox", "name": "vcio", "git-repo": "https://example.com/git/vcio.git", "url": "https://example.com/@vcio", "publicKey": { "id": "https://example.com/team/vcio#main-key", "owner": "https://example.com/team/vcio", "publicKeyPem": "PUBLIC KEY" }, ... }
……	…….	……

Frontend development using django template and jquery:

- login page

security team:

- security team signs up page
- security team profile page
- security team purls page (subscribe - unsubscribe)
- review page

database:

- database profile page
- database git page (for creating a new git repository)
- review page

Timeline :

Week 1 (2023/7/29, 2023/8/4):

- Create a python script that gets the improved data as JSON from a vulnerablecode project and save it in a remote git repository and track the changes and push every new file as a new vulnerability.

Week 2 (2023/./. , 2023/./.):

- implement the ER diagram using the Django model:
  https://docs.djangoproject.com/en/4.1/#the-model-layer
  so every user has the right permission and in the right group

  Using the Django authentication system
  https://docs.djangoproject.com/en/4.1/ref/contrib/auth/#group-model

Week 3 (2023/./. , 2023/./.) :

- use git-http-backend to server a git repositories
  https://modwsgi.readthedocs.io/en/develop/user-guides/access-control-mechanisms.html#apache-authentication-provider
- writing and running tests

Week 4 (2023/./. , 2023/./.) :

- implement authenticate between clients and servers using django-oauth-toolkit
- writing and running tests

Week 5 (2023/./. , 2023/./.) :

- database actor authenticate with git repository using django model .

https://docs.djangoproject.com/en/4.1/howto/deployment/wsgi/apache-auth/#how-to-authenticate-against-django-s-user-database-from-apache

- implement login page and security team sign up page

Week 6 (2023/./. , 2023/./.):

- implement a security team profile page
- implement a database actor profile page

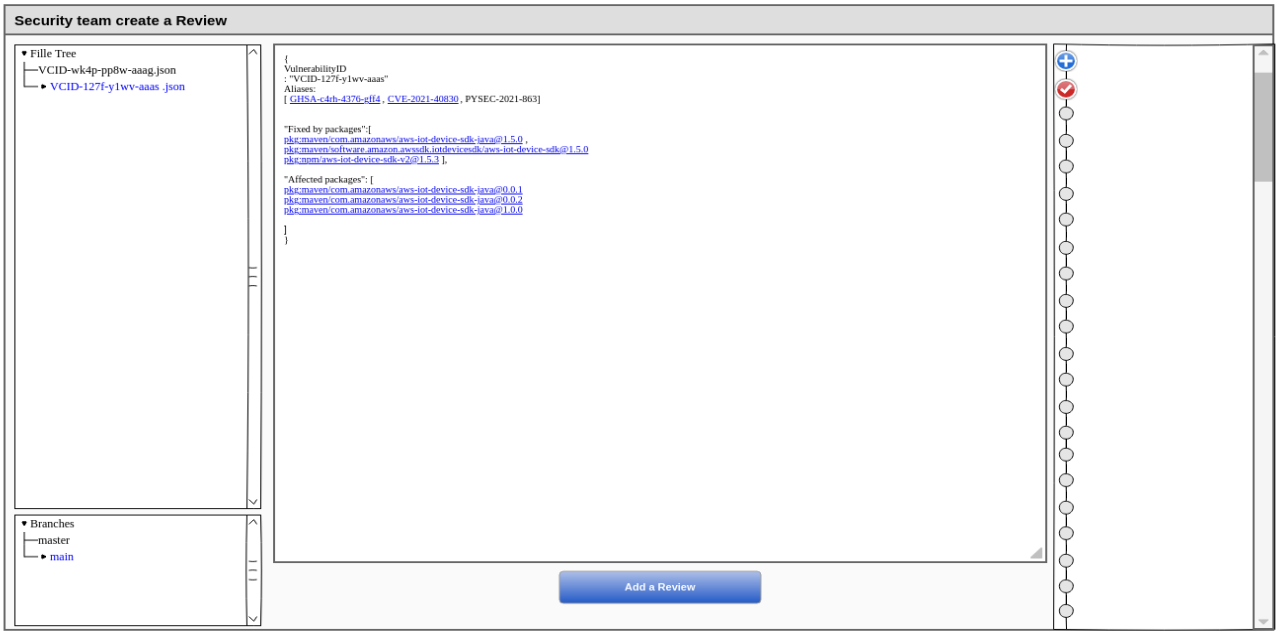
- writing and running tests

Week 7 (2023/./. , 2023/./.):

- implement a create review page
- writing and running tests

Week 8 (2023/./. , 2023/./.):

- implement a review page
- writing and running tests



Week 9 (2023/./. , 2023/./.):

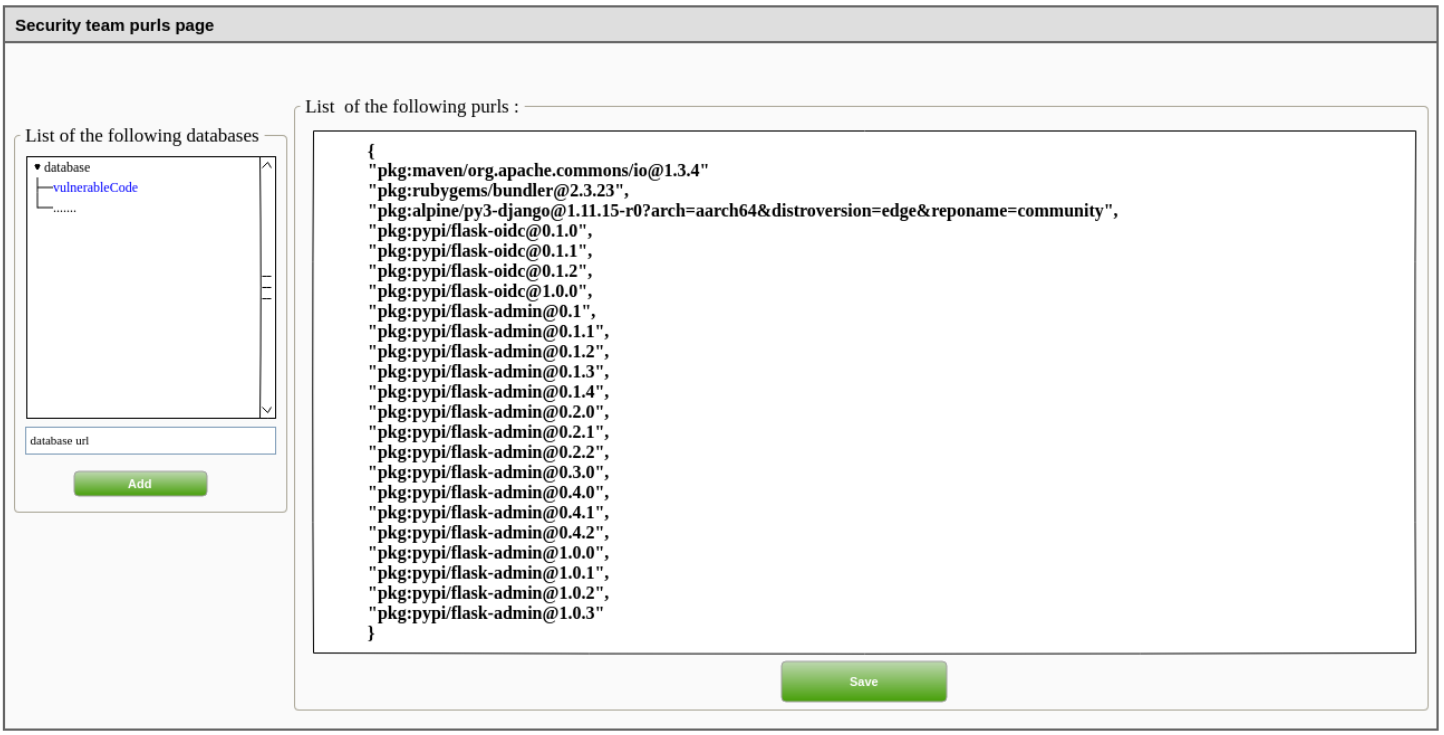
- implement a basic auth between (server to server) using HTTP Signatures
- writing and running tests

Week 10 (2023/./. , 2023/./.):

- implement Server to Server Interactions Endpoints
- writing and running tests

Week 11 (2023/./. , 2023/./.) :

- implement a security team purls page
- writing and running tests



Week 12 (2023/./. , 2023/./.) :

- Create a python script that get the accepted review data and feed it to postgresql
- implement a notification system
- More unit tests

Contact info:

- Name: Ziad hany
- Country: Egypt , Fayoum

-
-
- Github: https://github.com/ziadhany
- LinkedIn: https://www.linkedin.com/in/ziadhany/
- Education: Bachelor of Engineering - BE, Computer Engineering and Systems 2019 - 2024

Timeline:

Do you have any known time conflicts during the official coding period?

No, but my final exams may make small conflicts just one or two weeks maximum on the first of July. After that, I'm a full-time contributor to vulnerablecode .

My Skills:

Python , Django , Flask , React , Redux , numpy , pandas , Html, Css , Javascript , Docker , Jquery , AWS

Extra Information:

I'm a senior student at Fayoum university and

interested in web development and cyber security.

I also have four Udacity certifications

- Cloud DevOps :https://confirm.udacity.com/RDYSGM9L
- Advanced Web development: https://graduation.udacity.com/confirm/4SDKSAEK

- Advanced front end web development: https://graduation.udacity.com/confirm/PLTSHPSK

- Advanced Data analysis: https://graduation.udacity.com/confirm/VUG2QV6V

GSoC participation:

I participated in GSoC 2022 @Aboutcode