Report

Secure Programming Assignment 2

Student Name	Student Number	email address
Sebastian Konefal	**B00168561**	**b00168561@mytudublin.ie**

Digital Forensics and Cyber Security

TU765

Secure Programming 2024

Date: 8th of December 2024

Table of Contents

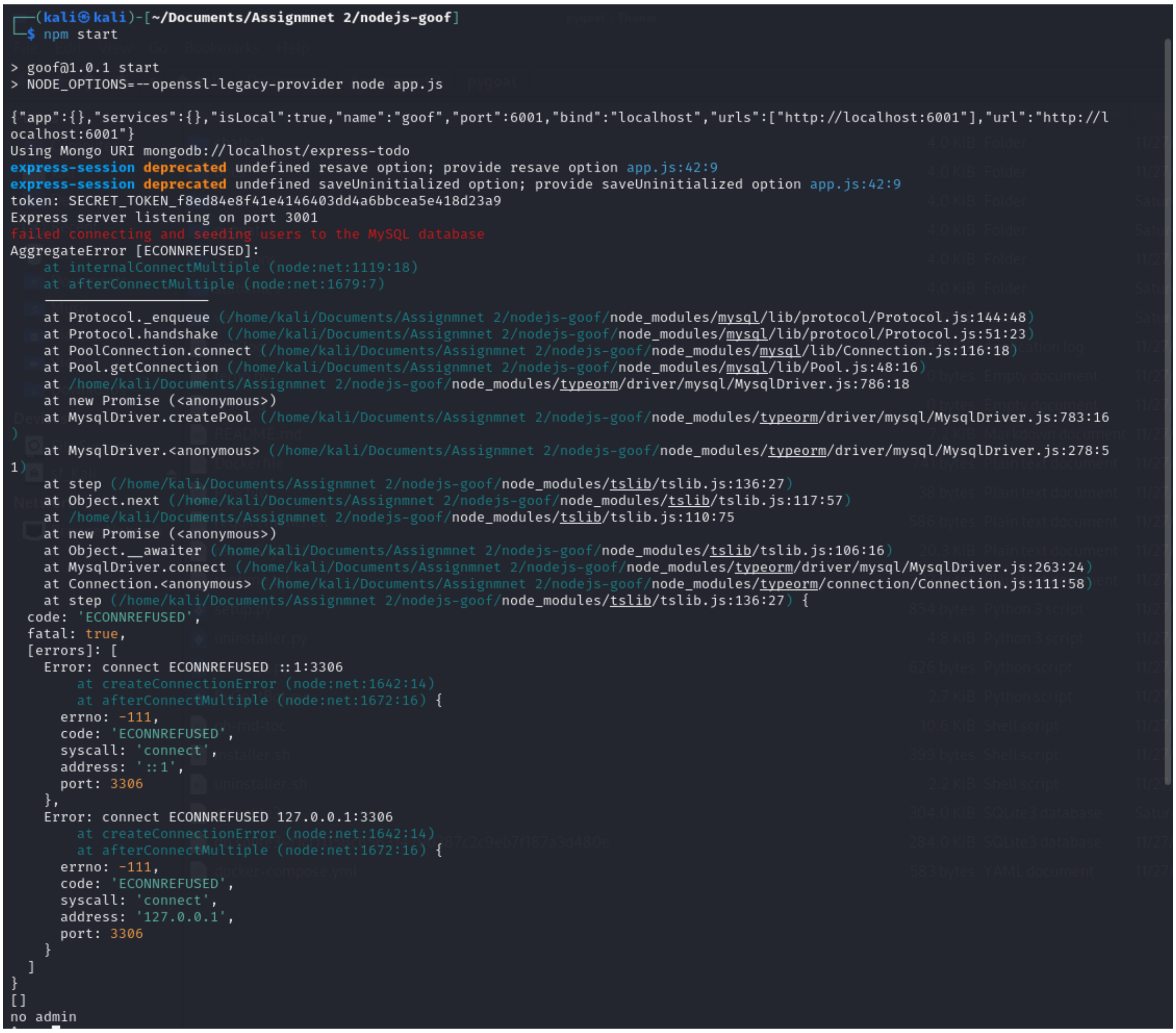
1. Section 1

1.1. Find a Web Application

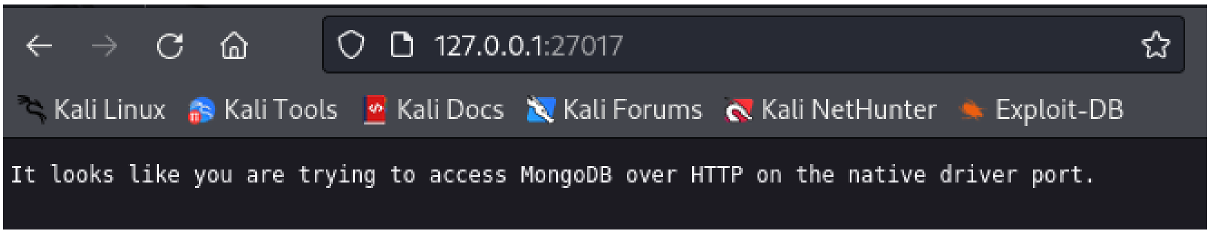
I decided that the potential application must have the following qualities:

- Recent maintained open-source web application, preferably with commits within the last year.
- Sufficient medium to advanced vulnerabilities to explore.
- Ability/capacity to compile/run the application.

There were several applications that I considered as most appropriate. I was unable to run some of the application as they relied on configured databases as below notwithstanding that I had MongoDB running locally.



```
┌──(kali㉿kali)-[~/Documents/Assignmnet 2/nodejs-goof]
└─$ npm start

> goof@1.0.1 start
> NODE_OPTIONS=--openssl-legacy-provider node app.js

{"app":{},"services":{},"isLocal":true,"name":"goof","port":6001,"bind":"localhost","urls":["http://localhost:6001"],"url":"http://l
ocalhost:6001"}
Using Mongo URI mongodb://localhost/express-todo
express-session deprecated undefined resave option; provide resave option app.js:42:9
express-session deprecated undefined saveUninitialized option; provide saveUninitialized option app.js:42:9
token: SECRET_TOKEN_f8ed84e8f41e4146403dd4a6bbcea5e418d23a9
Express server listening on port 3001
failed connecting and seeding users to the MySQL database
AggregateError [ECONNREFUSED]:
    at internalConnectMultiple (node:net:1119:18)
    at afterConnectMultiple (node:net:1679:7)
    ----------------
    at Protocol._enqueue (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/mysql/lib/protocol/Protocol.js:144:48)
    at Protocol.handshake (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/mysql/lib/protocol/Protocol.js:51:23)
    at PoolConnection.connect (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/mysql/lib/Connection.js:116:18)
    at Pool.getConnection (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/mysql/lib/Pool.js:48:16)
    at /home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/typeorm/driver/mysql/MysqlDriver.js:786:18
    at new Promise (<anonymous>)
    at MysqlDriver.createPool (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/typeorm/driver/mysql/MysqlDriver.js:783:16
)
    at MysqlDriver.<anonymous> (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/typeorm/driver/mysql/MysqlDriver.js:278:5
1)
    at step (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/tslib/tslib.js:136:27)
    at Object.next (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/tslib/tslib.js:117:57)
    at /home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/tslib/tslib.js:110:75
    at new Promise (<anonymous>)
    at Object.__awaiter (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/tslib/tslib.js:106:16)
    at MysqlDriver.connect (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/typeorm/driver/mysql/MysqlDriver.js:263:24)
    at Connection.<anonymous> (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/typeorm/connection/Connection.js:111:58)
    at step (/home/kali/Documents/Assignmnet 2/nodejs-goof/node_modules/tslib/tslib.js:136:27) {
  code: 'ECONNREFUSED',
  fatal: true,
  [errors]: [
    Error: connect ECONNREFUSED ::1:3306
        at createConnectionError (node:net:1642:14)
        at afterConnectMultiple (node:net:1672:16) {
      errno: -111,
      code: 'ECONNREFUSED',
      syscall: 'connect',
      address: '::1',
      port: 3306
    },
    Error: connect ECONNREFUSED 127.0.0.1:3306
        at createConnectionError (node:net:1642:14)
        at afterConnectMultiple (node:net:1672:16) {
      errno: -111,
      code: 'ECONNREFUSED',
      syscall: 'connect',
      address: '127.0.0.1',
      port: 3306
    }
  ]
}
[]
no admin
```



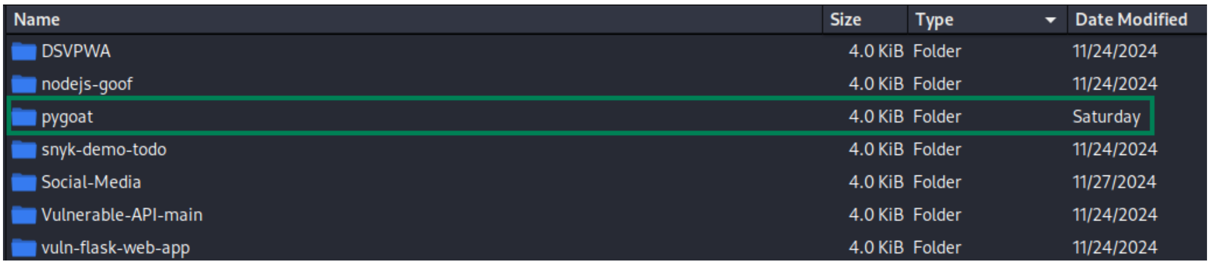
It looks like you are trying to access MongoDB over HTTP on the native driver port.

Trouble shooting somebody else code and DB error turned out to be quite time consuming and frustrating. Therefore, I decided to find another project that will compile and run. I have found two smaller projects that I managed to successfully run on my machine:

- https://github.com/michealkeines/Vulnerable-API
- https://github.com/guiadeappsec/vuln-flask-web-app

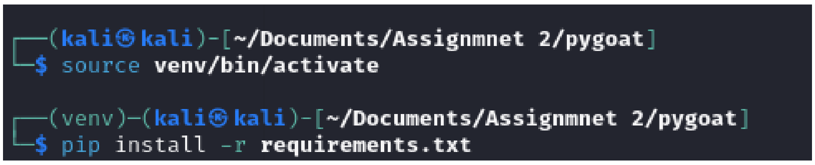
However, the vulnerabilities available and its complexity did not satisfy author's requirements.

After testing 7 applications in total, I **decided to choose PyGoat** available at https://github.com/adeyosemanputra/pygoat as this project fulfilled all the pre-requisite requirements for the assignment:

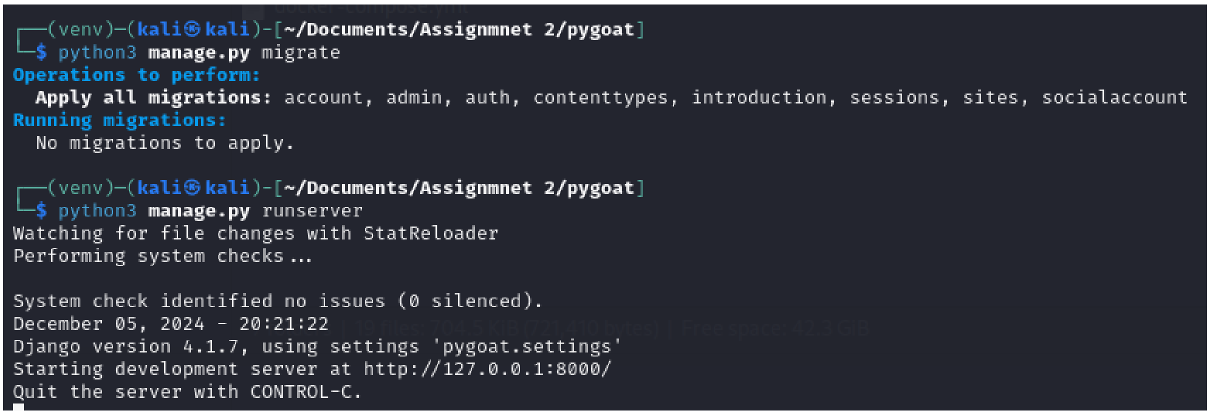


Name	Size	Type	Date Modified
DSVPWA	4.0 KiB	Folder	11/24/2024
nodejs-goof	4.0 KiB	Folder	11/24/2024
pygoat	4.0 KiB	Folder	Saturday
snyk-demo-todo	4.0 KiB	Folder	11/24/2024
Social-Media	4.0 KiB	Folder	11/27/2024
Vulnerable-API-main	4.0 KiB	Folder	11/24/2024
vuln-flask-web-app	4.0 KiB	Folder	11/24/2024

The application was run the using the following commands, firstly by creating a virtual environment using command *python3 -m venv venv*:

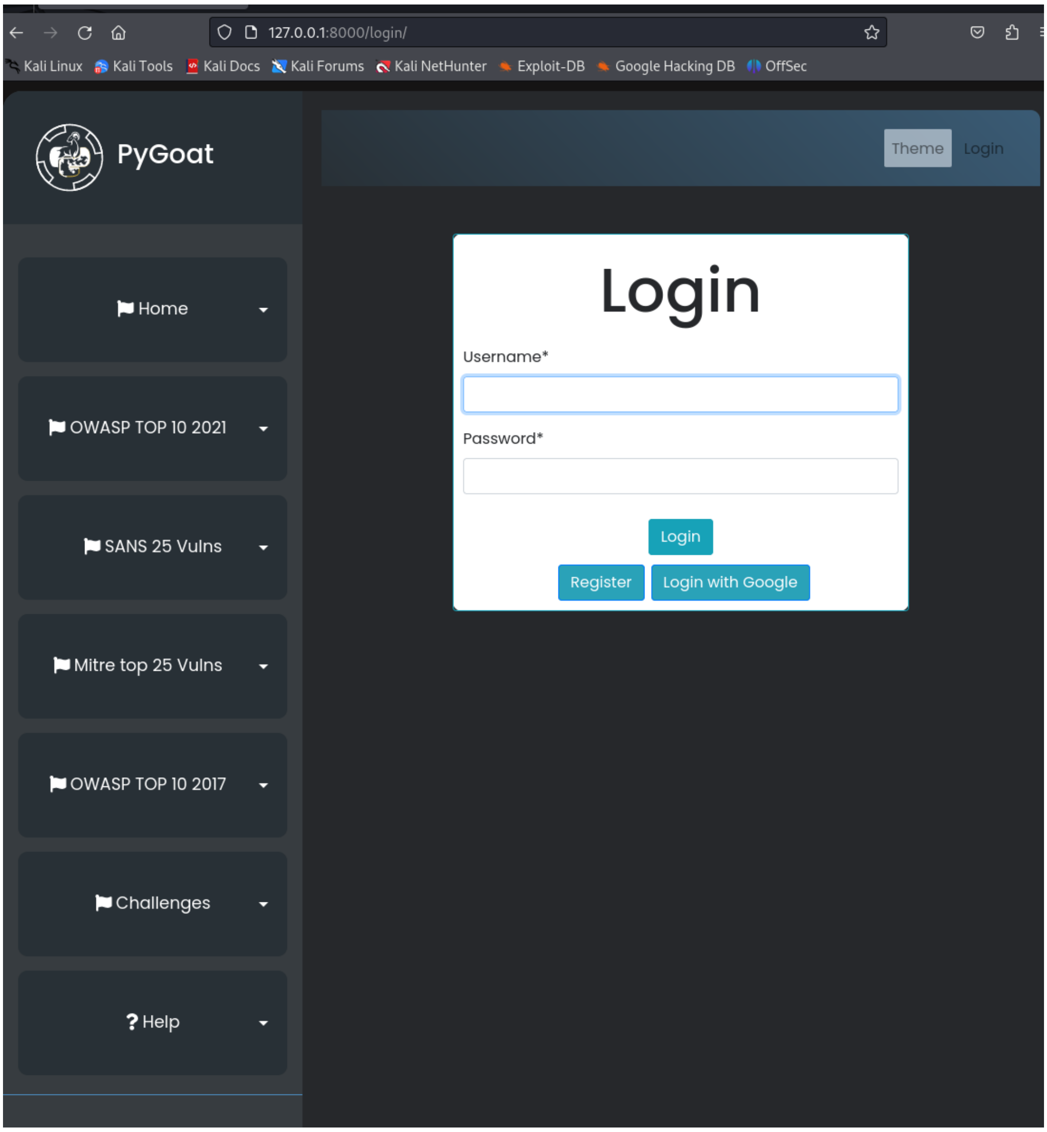


```
┌──(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ source venv/bin/activate

┌──(venv)─(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ pip install -r requirements.txt
```



```
┌──(venv)─(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: account, admin, auth, contenttypes, introduction, sessions, sites, socialaccount
Running migrations:
  No migrations to apply.

┌──(venv)─(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
December 05, 2024 - 20:21:22
Django version 4.1.7, using settings 'pygoat.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

The application website run on the loopback IP 127.0.0.1:8000 and the had following interface:



Login

Username*

Password*

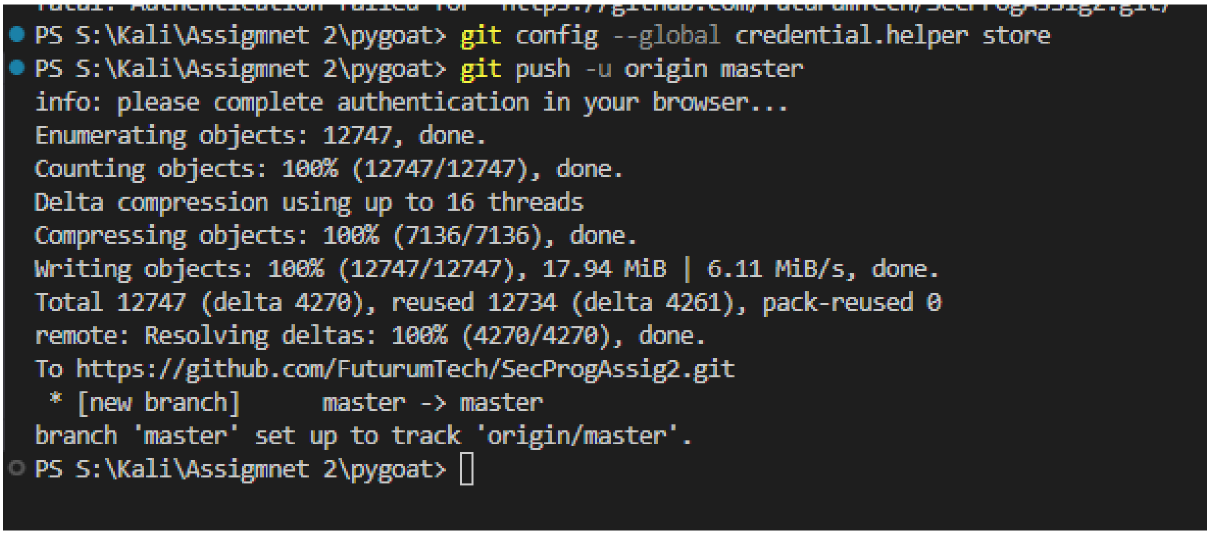
Login

Register

Login with Google

1.2. Static Analyser Comparison

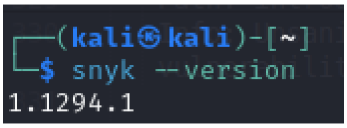
As a pre-requisite, to run the applications scanners via the website interface as opposed to only local scanners that have limited functionality, the scanners required supplying the relevant code via Git repository which has been created and authorised for scans in Snyk and Semgrep application Below is a screenshot confirming push to the GitHub repository:



```
PS S:\Kali\Assignmnet 2\pygoat> git config --global credential.helper store
PS S:\Kali\Assignmnet 2\pygoat> git push -u origin master
info: please complete authentication in your browser...
Enumerating objects: 12747, done.
Counting objects: 100% (12747/12747), done.
Delta compression using up to 16 threads
Compressing objects: 100% (7136/7136), done.
Writing objects: 100% (12747/12747), 17.94 MiB | 6.11 MiB/s, done.
Total 12747 (delta 4270), reused 12734 (delta 4261), pack-reused 0
remote: Resolving deltas: 100% (4270/4270), done.
To https://github.com/FuturumTech/SecProgAssig2.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
PS S:\Kali\Assignmnet 2\pygoat>
```

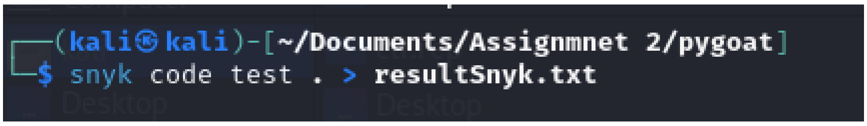
1.2.1. Snyk

Following local installation and configuration of Snyk version 1.1294.1:

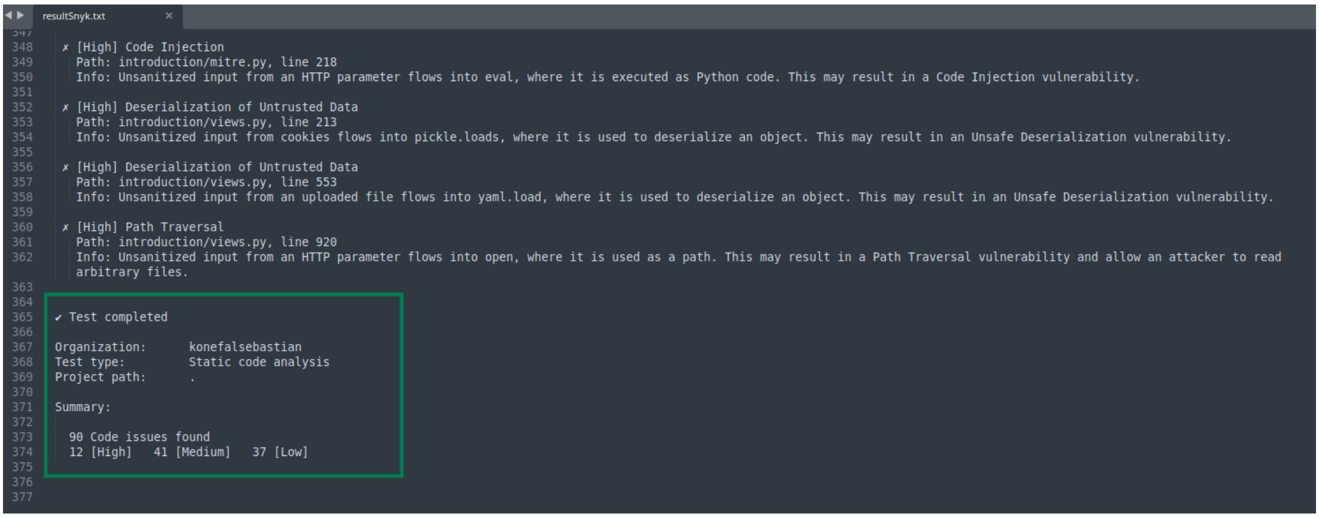


```
┌──(kali㉿kali)-[~]
└─$ snyk --version
1.1294.1
```

Static application security testing was performed:



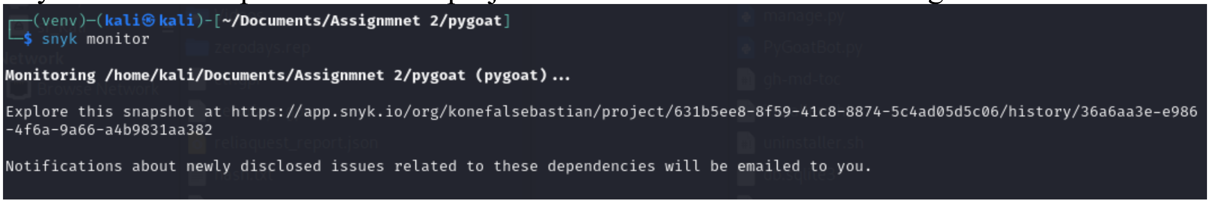
```
┌──(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ snyk code test . > resultSnyk.txt
```



```
 ✗ [High] Code Injection
   Path: introduction/mitre.py, line 218
   Info: Unsanitized input from an HTTP parameter flows into eval, where it is executed as Python code. This may result in a Code Injection vulnerability.

 ✗ [High] Deserialization of Untrusted Data
   Path: introduction/views.py, line 213
   Info: Unsanitized input from cookies flows into pickle.loads, where it is used to deserialize an object. This may result in an Unsafe Deserialization vulnerability.

 ✗ [High] Deserialization of Untrusted Data
   Path: introduction/views.py, line 553
   Info: Unsanitized input from an uploaded file flows into yaml.load, where it is used to deserialize an object. This may result in an Unsafe Deserialization vulnerability.

 ✗ [High] Path Traversal
   Path: introduction/views.py, line 920
   Info: Unsanitized input from an HTTP parameter flows into open, where it is used as a path. This may result in a Path Traversal vulnerability and allow an attacker to read
   arbitrary files.


✔ Test completed

Organization:      konefalsebastian
Test type:         Static code analysis
Project path:      .

Summary:

  90 Code issues found
  12 [High]   41 [Medium]   37 [Low]
```

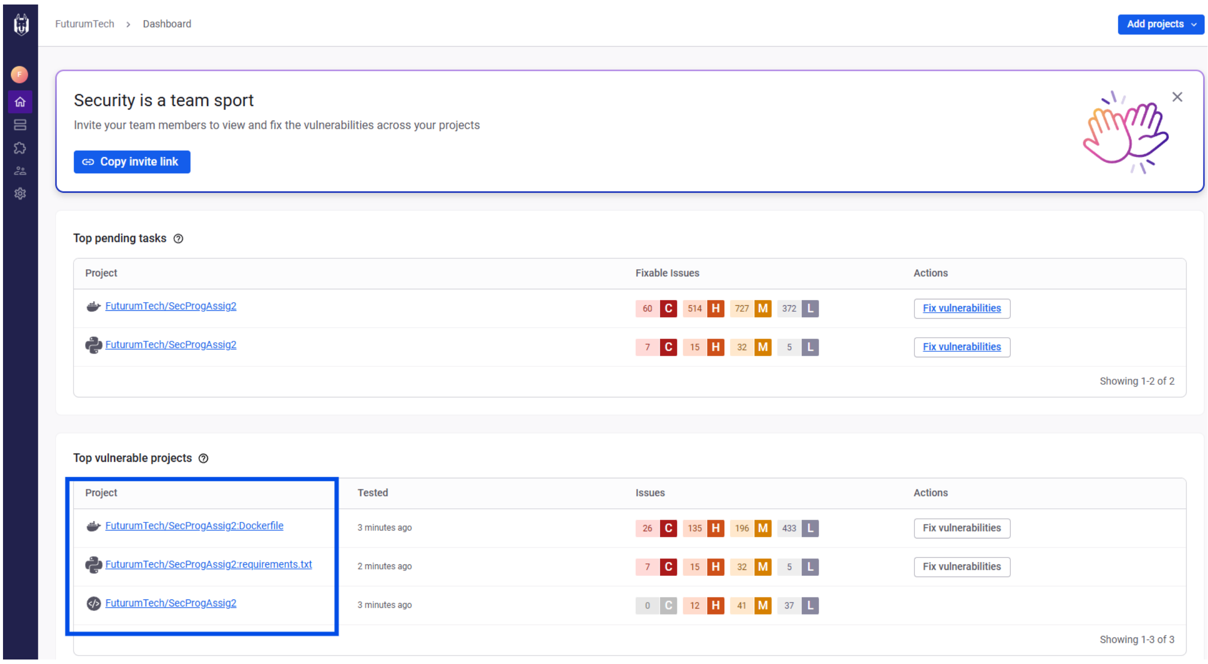
90 code issues were found of which 12 were High, 41 Medium and 37 Low. There was no suggestion on potential fixes of the vulnerabilities in the local scan.

Snyk also has an option to add the project to the dashboard to track changed in vulnerabilities:



```
┌──(venv)─(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ snyk monitor

Monitoring /home/kali/Documents/Assignmnet 2/pygoat (pygoat)...

Explore this snapshot at https://app.snyk.io/org/konefalsebastian/project/631b5ee8-8f59-41c8-8874-5c4ad05d5c06/history/36a6aa3e-e986
-4f6a-9a66-a4b9831aa382

Notifications about newly disclosed issues related to these dependencies will be emailed to you.
```

Screenshot of dashboard following activating the Snyk via Github repository was placed below. It can be seen that the scanner distinguished between results from Dockerfile, requirements.txt and the code source in the repository. The assigned will focus on vulnerabilities in the source code.



Security is a team sport

Invite your team members to view and fix the vulnerabilities across your projects

Top pending tasks

Project	Fixable Issues	Actions
FuturumTech/SecProgAssig2	60 C 514 H 727 M 372 L	Fix vulnerabilities
FuturumTech/SecProgAssig2	7 C 15 H 32 M 5 L	Fix vulnerabilities

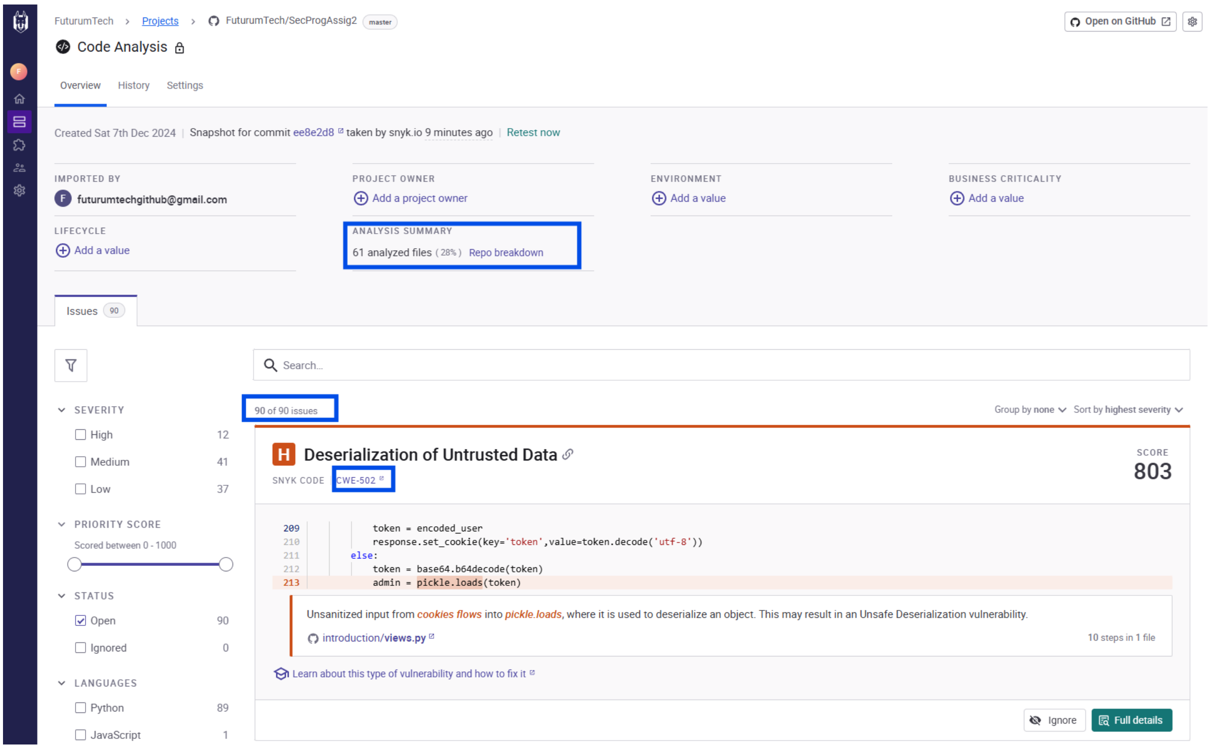
Showing 1-2 of 2

Top vulnerable projects

Project	Tested	Issues	Actions
FuturumTech/SecProgAssig2:Dockerfile	3 minutes ago	26 C 135 H 196 M 433 L	Fix vulnerabilities
FuturumTech/SecProgAssig2:requirements.txt	2 minutes ago	7 C 15 H 32 M 5 L	Fix vulnerabilities
FuturumTech/SecProgAssig2	3 minutes ago	0 C 12 H 41 M 37 L	

Showing 1-3 of 3

There are additional features available on the website such as sorting and scoring vulnerabilities. The local scan and website overview provided the same number of issues of the source code in the amount of 90 vulnerabilities indicating that 61 files were suitable for code analysis as containing source code and has very convenient filter of vulnerability types. The vulnerability is also mapped to CWE ID:



FuturumTech › Projects › FuturumTech/SecProgAssig2 master

Code Analysis

Created Sat 7th Dec 2024 | Snapshot for commit ee8e2d8 taken by snyk.io 9 minutes ago | Retest now

IMPORTED BY: futurumtechgithub@gmail.com

PROJECT OWNER: Add a project owner

ENVIRONMENT: Add a value

BUSINESS CRITICALITY: Add a value

LIFECYCLE: Add a value

ANALYSIS SUMMARY: 61 analyzed files (28%) Repo breakdown

Issues 90

SEVERITY
- □ High 12
- □ Medium 41
- □ Low 37

PRIORITY SCORE

Scored between 0 - 1000

STATUS
- ☑ Open 90
- □ Ignored 0

LANGUAGES
- □ Python 89
- □ JavaScript 1

90 of 90 issues

Group by none ⌄ Sort by highest severity ⌄

H Deserialization of Untrusted Data

SNYK CODE CWE-502

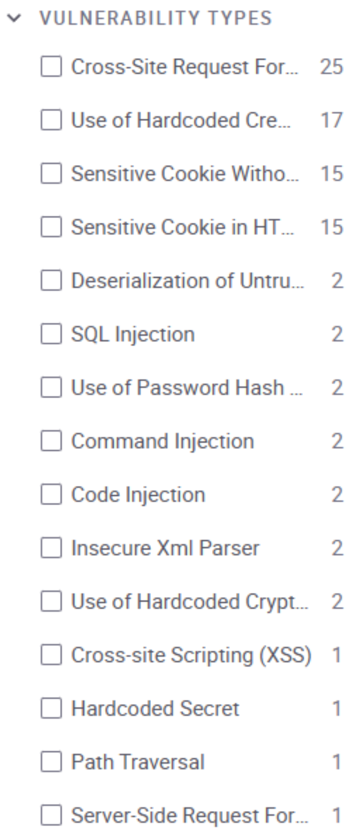
SCORE 803

```
            token = encoded_user
            response.set_cookie(key='token',value=token.decode('utf-8'))
        else:
            token = base64.b64decode(token)
            admin = pickle.loads(token)
```

Unsanitized input from *cookies flows* into *pickle.loads*, where it is used to deserialize an object. This may result in an Unsafe Deserialization vulnerability.

introduction/views.py 10 steps in 1 file

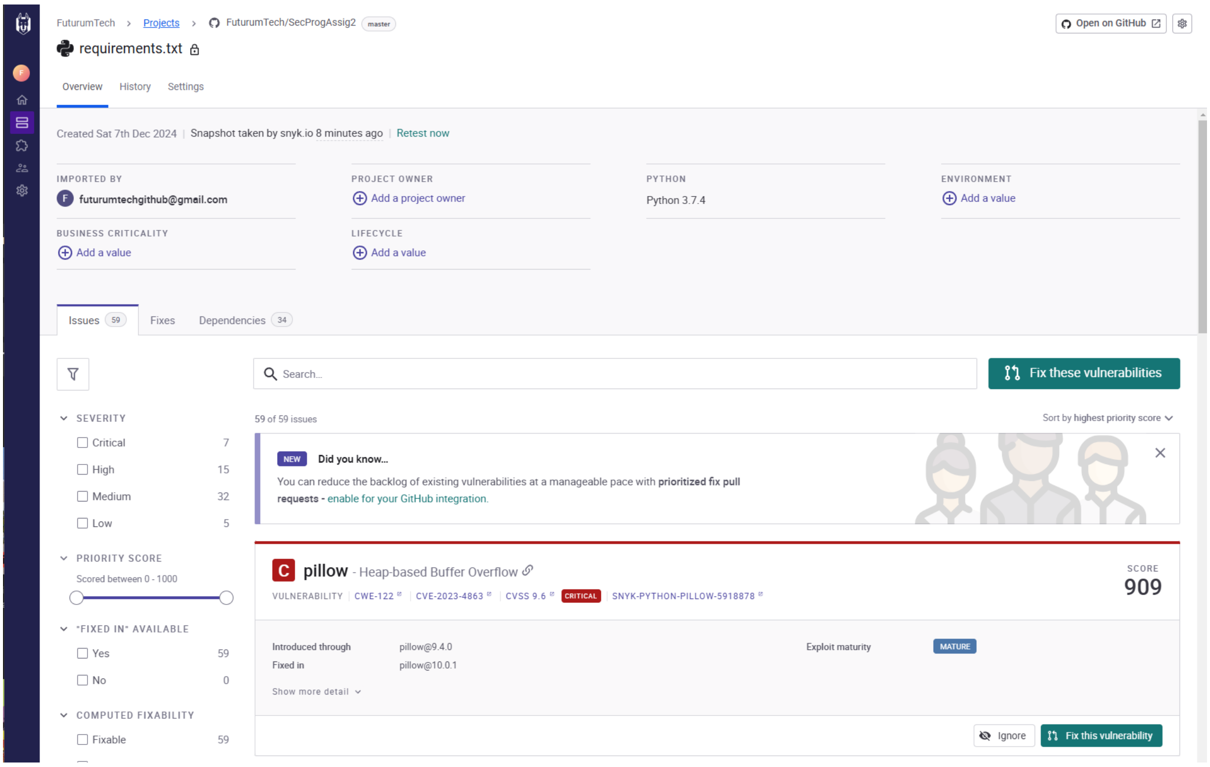
Learn about this type of vulnerability and how to fix it



VULNERABILITY TYPES

- □ Cross-Site Request For... 25
- □ Use of Hardcoded Cre... 17
- □ Sensitive Cookie Witho... 15
- □ Sensitive Cookie in HT... 15
- □ Deserialization of Untru... 2
- □ SQL Injection 2
- □ Use of Password Hash ... 2
- □ Command Injection 2
- □ Code Injection 2
- □ Insecure Xml Parser 2
- □ Use of Hardcoded Crypt... 2
- □ Cross-site Scripting (XSS) 1
- □ Hardcoded Secret 1
- □ Path Traversal 1
- □ Server-Side Request For... 1

Snyk also provides tracking of supply chain vulnerabilities: **open-source vulnerabilities and license issues** (Snyk Limited, 2024):



requirements.txt

Created Sat 7th Dec 2024 | Snapshot taken by snyk.io 8 minutes ago

IMPORTED BY: futurumtechgithub@gmail.com

PROJECT OWNER: Add a project owner

PYTHON: Python 3.7.4

ENVIRONMENT: Add a value

BUSINESS CRITICALITY: Add a value

LIFECYCLE: Add a value

SEVERITY

□ Critical	7
□ High	15
□ Medium	32
□ Low	5

PRIORITY SCORE

Scored between 0 - 1000

"FIXED IN" AVAILABLE

□ Yes	59
□ No	0

COMPUTED FIXABILITY

□ Fixable	59

59 of 59 issues

Sort by highest priority score

Did you know...

You can reduce the backlog of existing vulnerabilities at a manageable pace with **prioritized fix pull requests** - enable for your GitHub integration.

C **pillow** - Heap-based Buffer Overflow

VULNERABILITY | CWE-122 | CVE-2023-4863 | CVSS 9.6 | CRITICAL | SNYK-PYTHON-PILLOW-5918878

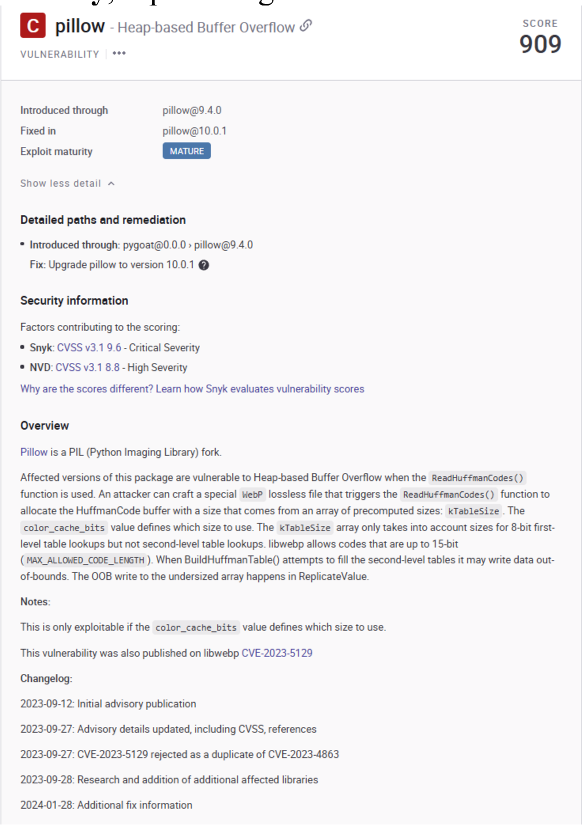
SCORE 909

Introduced through: pillow@9.4.0

Fixed in: pillow@10.0.1

Exploit maturity: MATURE

Additional information is also available of each vulnerability noting **CWE, CVE and CVSS severity**, providing an overview of the vulnerabilities and suggesting fixes:



C **pillow** - Heap-based Buffer Overflow

VULNERABILITY | •••

SCORE 909

Introduced through: pillow@9.4.0

Fixed in: pillow@10.0.1

Exploit maturity: MATURE

Detailed paths and remediation

- **Introduced through**: pygoat@0.0.0 › pillow@9.4.0
  Fix: Upgrade pillow to version 10.0.1

Security information

Factors contributing to the scoring:

- **Snyk**: CVSS v3.1 9.6 - Critical Severity
- **NVD**: CVSS v3.1 8.8 - High Severity

Why are the scores different? Learn how Snyk evaluates vulnerability scores

Overview

Pillow is a PIL (Python Imaging Library) fork.

Affected versions of this package are vulnerable to Heap-based Buffer Overflow when the `ReadHuffmanCodes()` function is used. An attacker can craft a special `WebP` lossless file that triggers the `ReadHuffmanCodes()` function to allocate the HuffmanCode buffer with a size that comes from an array of precomputed sizes: `kTableSize`. The `color_cache_bits` value defines which size to use. The `kTableSize` array only takes into account sizes for 8-bit first-level table lookups but not second-level table lookups. libwebp allows codes that are up to 15-bit (`MAX_ALLOWED_CODE_LENGTH`). When BuildHuffmanTable() attempts to fill the second-level tables it may write data out-of-bounds. The OOB write to the undersized array happens in ReplicateValue.

Notes:

This is only exploitable if the `color_cache_bits` value defines which size to use.

This vulnerability was also published on libwebp CVE-2023-5129

Changelog:

2023-09-12: Initial advisory publication

2023-09-27: Advisory details updated, including CVSS, references

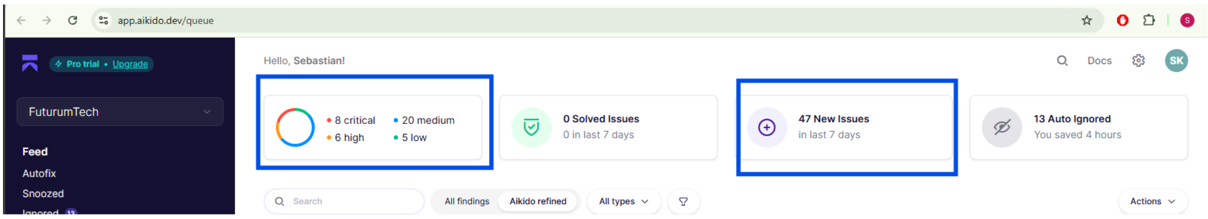
2023-09-27: CVE-2023-5129 rejected as a duplicate of CVE-2023-4863

2023-09-28: Research and addition of additional affected libraries

2024-01-28: Additional fix information

1.2.2. Semgrep

. The project also considered scan via Aikido SAST, however, the results of Aikido were unsatisfactory with only 47 issues found:



app.aikido.dev/queue

Pro trial • Upgrade

FutureumTech

Feed

Autofix

Snoozed

Ignored 13

Hello, Sebastian!

Docs

SK

● 8 critical ● 20 medium
● 6 high ● 5 low

0 Solved Issues
0 in last 7 days

47 New Issues
in last 7 days

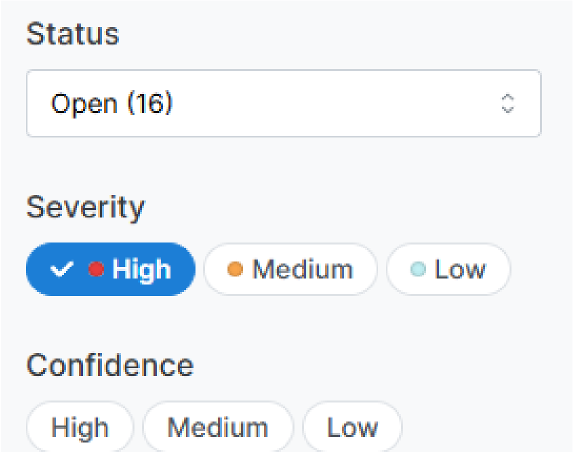
13 Auto Ignored
You saved 4 hours

Search

All findings | Aikido refined | All types

Actions

Therefore, Semgrep was chosen as second vulnerability scanner due to its maturity and reliance. As can be seen below, the Semgrep application found **101 vulnerable code** of which 16 were categorised as high, 76 medium and 9 low severity:



Status

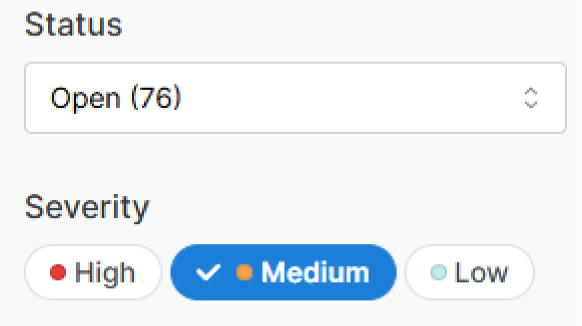
Open (16)

Severity

✓ ● High | ● Medium | ● Low

Confidence

High | Medium | Low

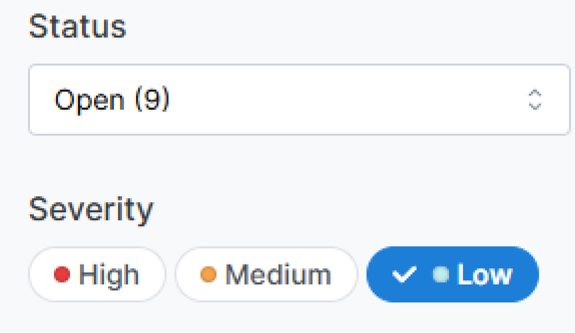


Status

Open (76)

Severity

● High | ✓ ● Medium | ● Low



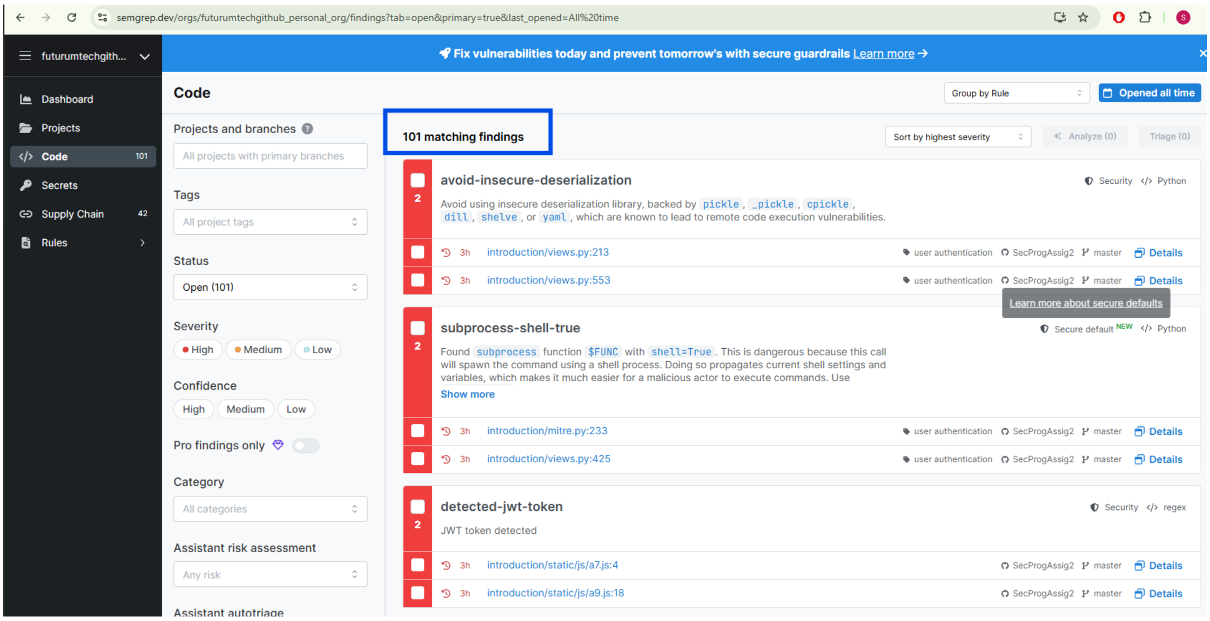
Status

Open (9)

Severity

● High | ● Medium | ✓ ● Low

The application has very intuitive interface and clearly shows what part of the application was deemed vulnerable and highlighted the relevant part of the code with some brief explanation of the vulnerability:



101 matching findings

avoid-insecure-deserialization (2) — Security · Python

Avoid using insecure deserialization library, backed by `pickle`, `_pickle`, `cpickle`, `dill`, `shelve`, or `yaml`, which are known to lead to remote code execution vulnerabilities.

- 3h introduction/views.py:213 — user authentication · SecProgAssig2 · master · Details
- 3h introduction/views.py:553 — user authentication · SecProgAssig2 · master · Details

subprocess-shell-true (2) — Secure default NEW · Python

Found `subprocess` function `$FUNC` with `shell=True`. This is dangerous because this call will spawn the command using a shell process. Doing so propagates current shell settings and variables, which makes it much easier for a malicious actor to execute commands. Use

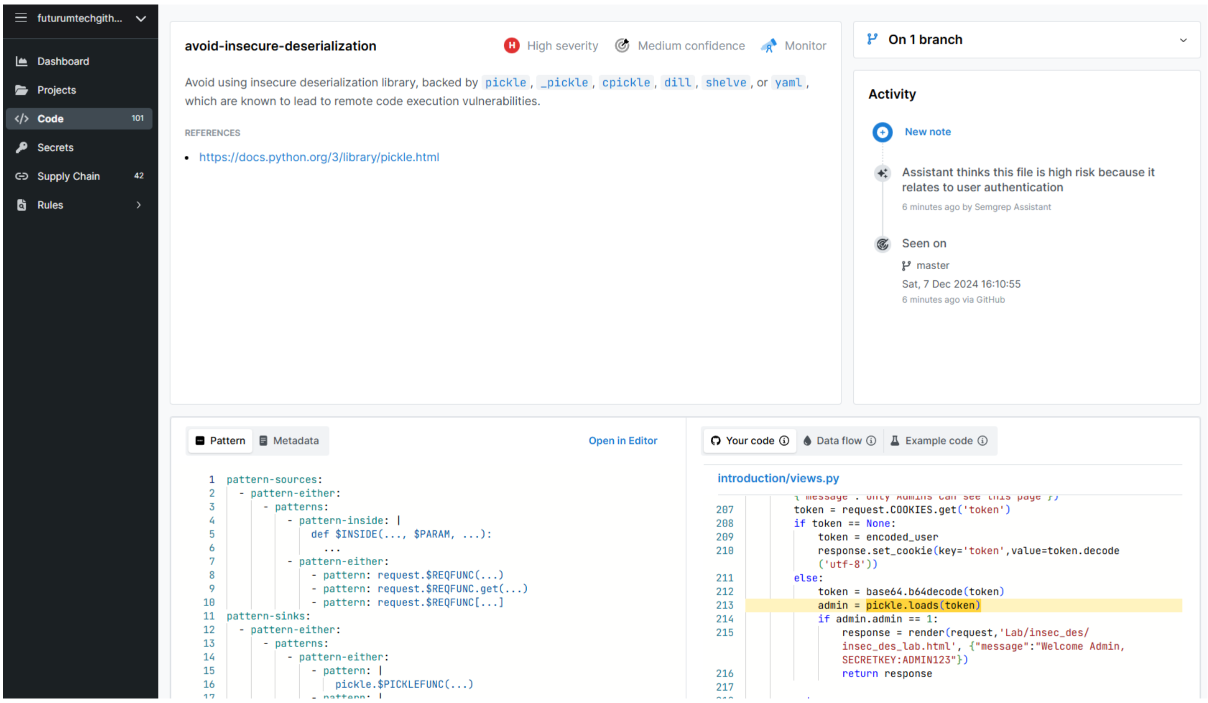
Show more

- 3h introduction/mitre.py:233 — user authentication · SecProgAssig2 · master · Details
- 3h introduction/views.py:425 — user authentication · SecProgAssig2 · master · Details

detected-jwt-token (2) — Security · regex

JWT token detected

- 3h introduction/static/js/a7.js:4 — SecProgAssig2 · master · Details
- 3h introduction/static/js/a9.js:18 — SecProgAssig2 · master · Details



avoid-insecure-deserialization — H High severity · Medium confidence · Monitor

Avoid using insecure deserialization library, backed by `pickle`, `_pickle`, `cpickle`, `dill`, `shelve`, or `yaml`, which are known to lead to remote code execution vulnerabilities.

REFERENCES

- https://docs.python.org/3/library/pickle.html

Activity

Assistant thinks this file is high risk because it relates to user authentication

6 minutes ago by Semgrep Assistant

Seen on

master

Sat, 7 Dec 2024 16:10:55

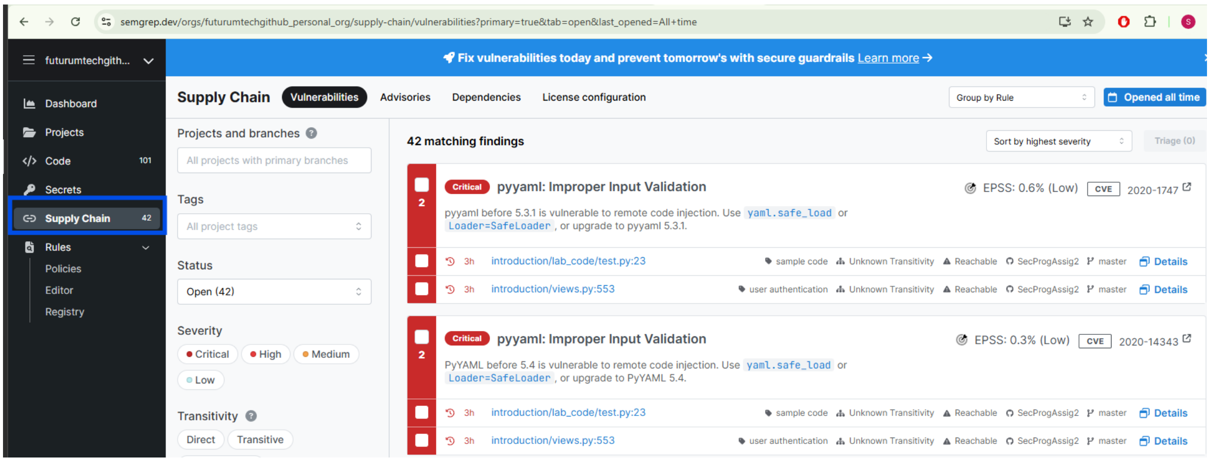
6 minutes ago via GitHub

```
pattern-sources:
  - pattern-either:
      - patterns:
          - pattern-inside: |
              def $INSIDE(..., $PARAM, ...):
                ...
          - pattern-either:
              - pattern: request.$REQFUNC(...)
              - pattern: request.$REQFUNC.get(...)
              - pattern: request.$REQFUNC[...]
pattern-sinks:
  - pattern-either:
      - patterns:
          - pattern-either:
              - pattern: |
                  pickle.$PICKLEFUNC(...)
```

introduction/views.py

```
        token = request.COOKIES.get('token')
        if token == None:
            token = encoded_user
            response.set_cookie(key='token',value=token.decode
            ('utf-8'))
        else:
            token = base64.b64decode(token)
            admin = pickle.loads(token)
            if admin.admin == 1:
                response = render(request,'Lab/insec_des/
                insec_des_lab.html', {"message":"Welcome Admin,
                SECRETKEY:ADMIN123"})
                return response
```

In addition, the application provided scanning of supply chain vulnerabilities and provided explanation and potential fixes mapped to CVE framework:



42 matching findings

Critical 2 pyyaml: Improper Input Validation — EPSS: 0.6% (Low) CVE 2020-1747

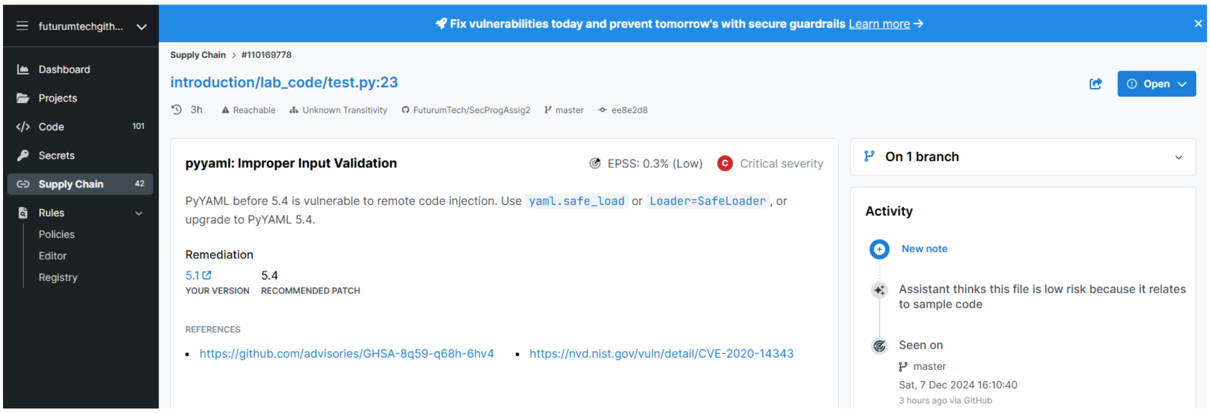
pyyaml before 5.3.1 is vulnerable to remote code injection. Use `yaml.safe_load` or `Loader=SafeLoader`, or upgrade to pyyaml 5.3.1.

- 3h introduction/lab_code/test.py:23 — sample code, Unknown Transitivity, Reachable, SecProgAssig2, master, Details
- 3h introduction/views.py:553 — user authentication, Unknown Transitivity, Reachable, SecProgAssig2, master, Details

Critical 2 pyyaml: Improper Input Validation — EPSS: 0.3% (Low) CVE 2020-14343

PyYAML before 5.4 is vulnerable to remote code injection. Use `yaml.safe_load` or `Loader=SafeLoader`, or upgrade to PyYAML 5.4.

- 3h introduction/lab_code/test.py:23 — sample code, Unknown Transitivity, Reachable, SecProgAssig2, master, Details
- 3h introduction/views.py:553 — user authentication, Unknown Transitivity, Reachable, SecProgAssig2, master, Details



introduction/lab_code/test.py:23

3h Reachable Unknown Transitivity FuturumTech/SecProgAssig2 master ee8e2d8

pyyaml: Improper Input Validation — EPSS: 0.3% (Low) — Critical severity

PyYAML before 5.4 is vulnerable to remote code injection. Use `yaml.safe_load` or `Loader=SafeLoader`, or upgrade to PyYAML 5.4.

Remediation

YOUR VERSION	RECOMMENDED PATCH
5.1	5.4

REFERENCES

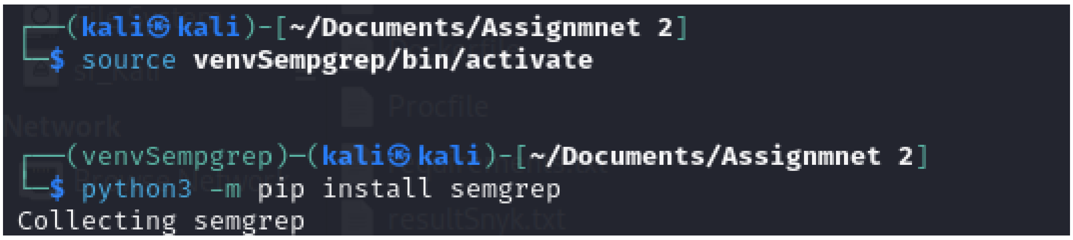
- https://github.com/advisories/GHSA-8q59-q68h-6hv4
- https://nvd.nist.gov/vuln/detail/CVE-2020-14343

Activity

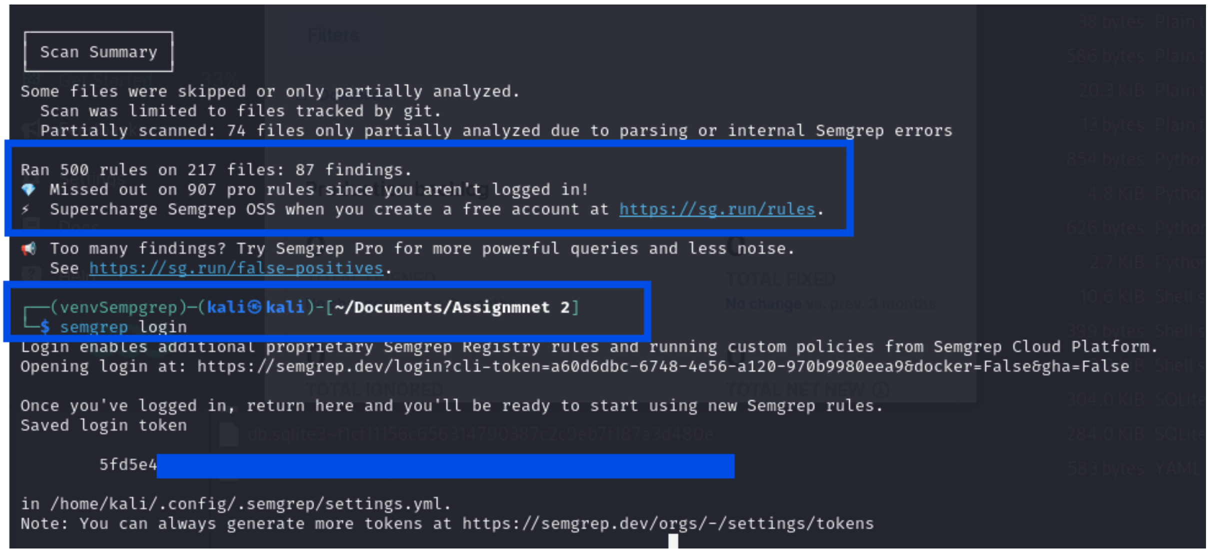
Assistant thinks this file is low risk because it relates to sample code

Seen on
master
Sat, 7 Dec 2024 16:10:40
3 hours ago via GitHub

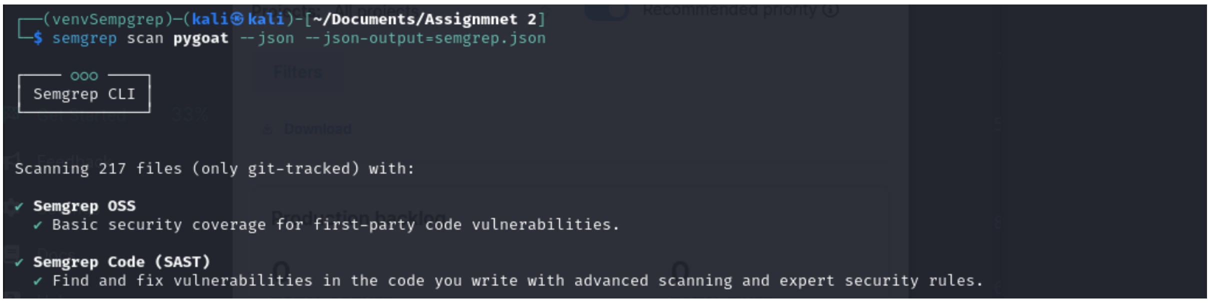
There is also an option to run the application as Local Scanner which was conducted using version 1.99.0, following creating of virtual environment, installation and token authentication, the application produced extremely long json file and found 90 findings. However, due to lower count of vulnerabilities and less user friendly interface of .json file, it was more desirable to follow the web application interface. Below screenshots documents all the steps described above:



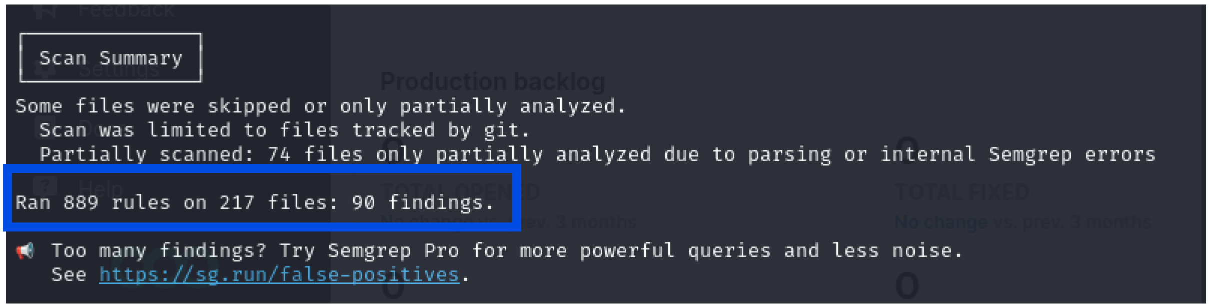
```
┌──(kali㉿kali)-[~/Documents/Assignmnet 2]
└─$ source venvSempgrep/bin/activate

┌──(venvSempgrep)─(kali㉿kali)-[~/Documents/Assignmnet 2]
└─$ python3 -m pip install semgrep
Collecting semgrep
```



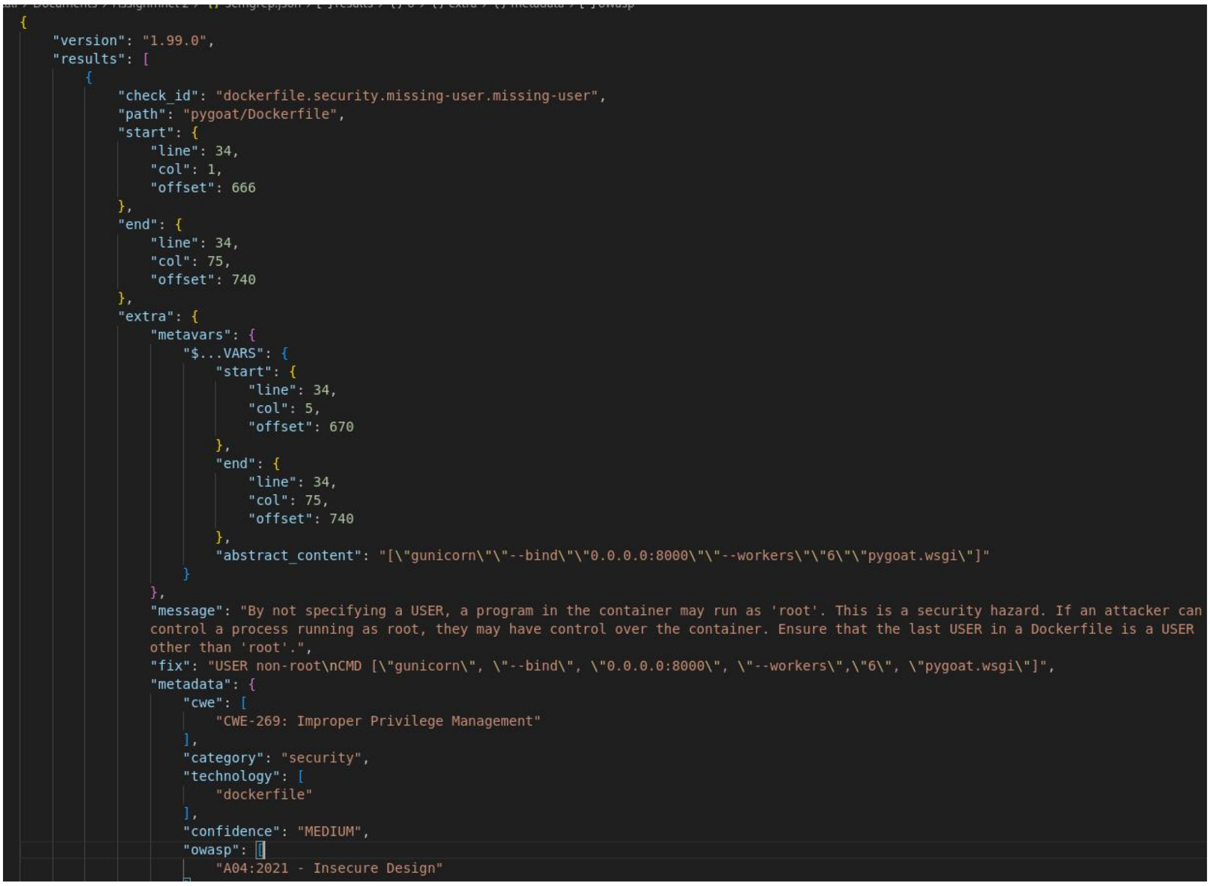
```
┌─────────────┐
│ Scan Summary │
└─────────────┘
Some files were skipped or only partially analyzed.
  Scan was limited to files tracked by git.
  Partially scanned: 74 files only partially analyzed due to parsing or internal Semgrep errors

Ran 500 rules on 217 files: 87 findings.
💎 Missed out on 907 pro rules since you aren't logged in!
⚡ Supercharge Semgrep OSS when you create a free account at https://sg.run/rules.

📢 Too many findings? Try Semgrep Pro for more powerful queries and less noise.
   See https://sg.run/false-positives.

┌──(venvSempgrep)-(kali㉿kali)-[~/Documents/Assignmnet 2]
└─$ semgrep login
Login enables additional proprietary Semgrep Registry rules and running custom policies from Semgrep Cloud Platform.
Opening login at: https://semgrep.dev/login?cli-token=a60d6dbc-6748-4e56-a120-970b9980eea9&docker=False&gha=False

Once you've logged in, return here and you'll be ready to start using new Semgrep rules.
Saved login token

        5fd5e4[redacted]

in /home/kali/.config/.semgrep/settings.yml.
Note: You can always generate more tokens at https://semgrep.dev/orgs/-/settings/tokens
```



```
┌──(venvSempgrep)-(kali㉿kali)-[~/Documents/Assignmnet 2]
└─$ semgrep scan pygoat --json --json-output=semgrep.json

┌──── ○○○ ────┐
│ Semgrep CLI │
└─────────────┘

Scanning 217 files (only git-tracked) with:

✔ Semgrep OSS
  ✔ Basic security coverage for first-party code vulnerabilities.

✔ Semgrep Code (SAST)
  ✔ Find and fix vulnerabilities in the code you write with advanced scanning and expert security rules.
```



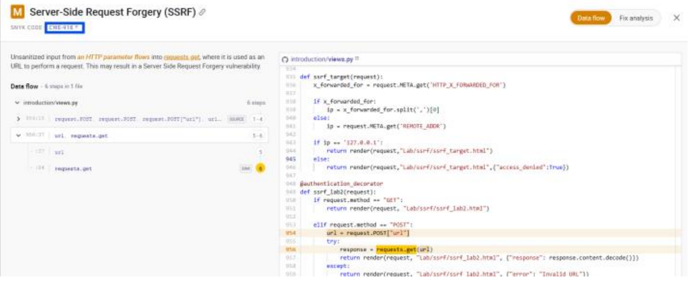
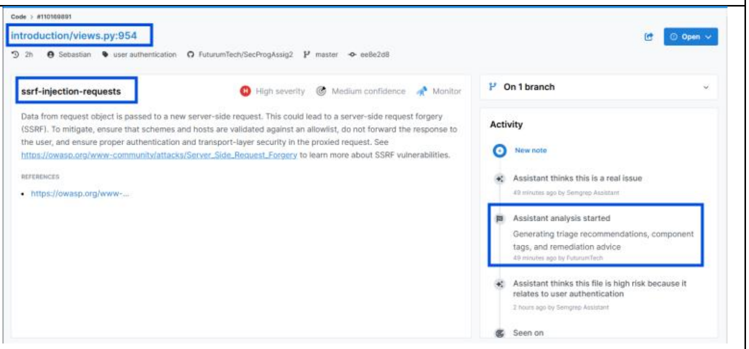
```
┌──────────────┐
│ Scan Summary │
└──────────────┘
Some files were skipped or only partially analyzed.
  Scan was limited to files tracked by git.
  Partially scanned: 74 files only partially analyzed due to parsing or internal Semgrep errors

Ran 889 rules on 217 files: 90 findings.

📢 Too many findings? Try Semgrep Pro for more powerful queries and less noise.
   See https://sg.run/false-positives.
```



```
{
    "version": "1.99.0",
    "results": [
        {
            "check_id": "dockerfile.security.missing-user.missing-user",
            "path": "pygoat/Dockerfile",
            "start": {
                "line": 34,
                "col": 1,
                "offset": 666
            },
            "end": {
                "line": 34,
                "col": 75,
                "offset": 740
            },
            "extra": {
                "metavars": {
                    "$...VARS": {
                        "start": {
                            "line": 34,
                            "col": 5,
                            "offset": 670
                        },
                        "end": {
                            "line": 34,
                            "col": 75,
                            "offset": 740
                        },
                        "abstract_content": "[\"gunicorn\"\"--bind\"\"0.0.0.0:8000\"\"--workers\"\"6\"\"pygoat.wsgi\"]"
                    }
                },
                "message": "By not specifying a USER, a program in the container may run as 'root'. This is a security hazard. If an attacker can
                control a process running as root, they may have control over the container. Ensure that the last USER in a Dockerfile is a USER
                other than 'root'.",
                "fix": "USER non-root\nCMD [\"gunicorn\", \"--bind\", \"0.0.0.0:8000\", \"--workers\",\"6\", \"pygoat.wsgi\"]",
                "metadata": {
                    "cwe": [
                        "CWE-269: Improper Privilege Management"
                    ],
                    "category": "security",
                    "technology": [
                        "dockerfile"
                    ],
                    "confidence": "MEDIUM",
                    "owasp": [
                        "A04:2021 - Insecure Design"
```

1.2.3 Compared results of a critical vulnerability

A single and critical code vulnerability was chosen to demonstrate different approaches of those two Vulnerability scanners. Both scanners detected Server Side Request Forgery (SSRF) and provided the following descriptions:

Server-Side Request Forgery (SSRF) **Views.py line 954**	
Snyk	**Semgrep**
	

Server-Side Request Forgery (SSRF)

SNYK CODE CWE-918

Unsanitized input from an HTTP parameter flows into requests.get, where it is used as an URL to perform a request. This may result in a Server Side Request Forgery vulnerability.

Data flow · 6 steps in 1 file

```
introduction/views.py                                              6 steps
  954:13 | request.POST  request.POST  request.POST["url"]  url...  SOURCE  1-4
  956:37 | url  requests.get                                        5-6
     :37 | url                                                      5
     :34 | requests.get                                     SINK    6
```

introduction/views.py

```
934
935 def ssrf_target(request):
936     x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
937
938     if x_forwarded_for:
939         ip = x_forwarded_for.split(',')[0]
940     else:
941         ip = request.META.get('REMOTE_ADDR')
942
943     if ip == '127.0.0.1':
944         return render(request,"Lab/ssrf/ssrf_target.html")
945     else:
946         return render(request,"Lab/ssrf/ssrf_target.html",{"access_denied":True})
947
948 @authentication_decorator
949 def ssrf_lab2(request):
950     if request.method == "GET":
951         return render(request, "Lab/ssrf/ssrf_lab2.html")
952
953     elif request.method == "POST":
954         url = request.POST["url"]
955         try:
956             response = requests.get(url)
957             return render(request, "Lab/ssrf/ssrf_lab2.html", {"response": response.content.decode()})
958         except:
959             return render(request, "Lab/ssrf/ssrf_lab2.html", {"error": "Invalid URL"})
```

Code > #110169891

introduction/views.py:954

2h · Sebastian · user authentication · FuturumTech/SecProgAssig2 · master · ee8e2d8

ssrf-injection-requests

High severity · Medium confidence · Monitor

Data from request object is passed to a new server-side request. This could lead to a server-side request forgery (SSRF). To mitigate, ensure that schemes and hosts are validated against an allowlist, do not forward the response to the user, and ensure proper authentication and transport-layer security in the proxied request. See https://owasp.org/www-community/attacks/Server_Side_Request_Forgery to learn more about SSRF vulnerabilities.

REFERENCES

- https://owasp.org/www-...

On 1 branch

Activity

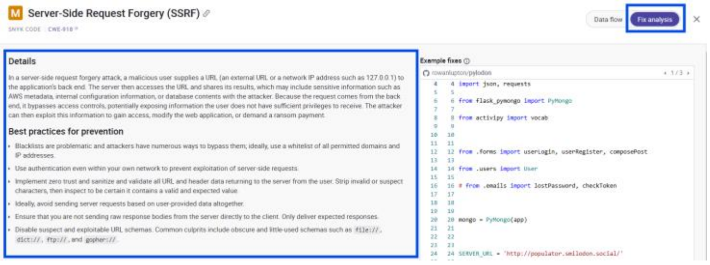
New note

Assistant thinks this is a real issue
49 minutes ago by Semgrep Assistant

Assistant analysis started
Generating triage recommendations, component tags, and remediation advice
49 minutes ago by FuturumTech

Assistant thinks this file is high risk because it relates to user authentication
2 hours ago by Semgrep Assistant

Seen on



Server-Side Request Forgery (SSRF)

SNYK CODE | CWE-918

Details

In a server-side request forgery attack, a malicious user supplies a URL (an external URL or a network IP address such as 127.0.0.1) to the application's back end. The server then accesses the URL and shares its results, which may include sensitive information such as AWS metadata, internal configuration information, or database contents with the attacker. Because the request comes from the back end, it bypasses access controls, potentially exposing information the user does not have sufficient privileges to receive. The attacker can then exploit this information to gain access, modify the web application, or demand a ransom payment.

Best practices for prevention

- Blacklists are problematic and attackers have numerous ways to bypass them; ideally, use a whitelist of all permitted domains and IP addresses.
- Use authentication even within your own network to prevent exploitation of server-side requests.
- Implement zero trust and sanitize and validate all URL and header data returning to the server from the user. Strip invalid or suspect characters, then inspect to be certain it contains a valid and expected value.
- Ideally, avoid sending server requests based on user-provided data altogether.
- Ensure that you are not sending raw response bodies from the server directly to the client. Only deliver expected responses.
- Disable suspect and exploitable URL schemas. Common culprits include obscure and little-used schemas such as `file://`, `dict://`, `ftp://`, and `gopher://`.

Example fixes

rowanlupton/pylodon 1 / 3

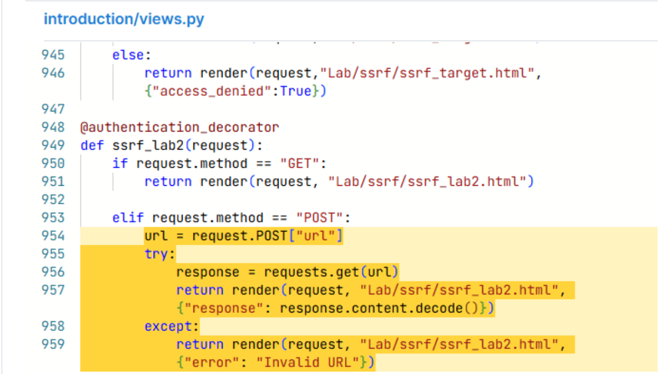
```
import json, requests

from flask_pymongo import PyMongo

from activipy import vocab


from .forms import userLogin, userRegister, composePost

from .users import User

# from .emails import lostPassword, checkToken



mongo = PyMongo(app)



SERVER_URL = 'http://populator.smilodon.social/'
```

"Details

In a server-side request forgery attack, a malicious user supplies a URL (an external URL or a network IP address such as 127.0.0.1) to the application's back end. The server then accesses the URL and shares its results, which may include sensitive information such as AWS metadata, internal configuration information, or database contents with the attacker. Because the request comes from the back end, it bypasses access controls, potentially exposing information the user does not have sufficient privileges to receive. The attacker can then exploit this information to gain access, modify the web application, or demand a ransom payment.

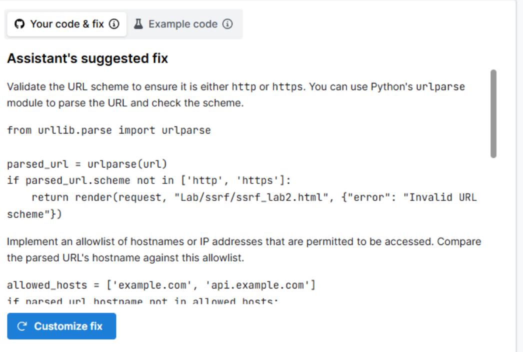
Best practices for prevention

- *Blacklists are problematic and attackers have numerous ways to bypass them; ideally, use a whitelist of all permitted domains and IP addresses.*
- *Use authentication even within your own network to prevent exploitation of server-side requests.*
- *Implement zero trust and sanitize and validate all URL and header data returning to the server from the user. Strip invalid or suspect characters, then inspect to be certain it contains a valid and expected value.*
- *Ideally, avoid sending server requests based on user-provided data altogether.*
- *Ensure that you are not sending raw response bodies from the server directly to the client. Only deliver expected responses.*
- *Disable suspect and exploitable URL schemas. Common culprits include obscure and little-used schemas such as file://, dict://, ftp://, and gopher://."*



introduction/views.py

```
    else:
        return render(request,"Lab/ssrf/ssrf_target.html",
        {"access_denied":True})

@authentication_decorator
def ssrf_lab2(request):
    if request.method == "GET":
        return render(request, "Lab/ssrf/ssrf_lab2.html")

    elif request.method == "POST":
        url = request.POST["url"]
        try:
            response = requests.get(url)
            return render(request, "Lab/ssrf/ssrf_lab2.html",
            {"response": response.content.decode()})
        except:
            return render(request, "Lab/ssrf/ssrf_lab2.html",
            {"error": "Invalid URL"})
```



Assistant's suggested fix

Validate the URL scheme to ensure it is either `http` or `https`. You can use Python's `urlparse` module to parse the URL and check the scheme.

```
from urllib.parse import urlparse

parsed_url = urlparse(url)
if parsed_url.scheme not in ['http', 'https']:
    return render(request, "Lab/ssrf/ssrf_lab2.html", {"error": "Invalid URL
scheme"})
```

Implement an allowlist of hostnames or IP addresses that are permitted to be accessed. Compare the parsed URL's hostname against this allowlist.

```
allowed_hosts = ['example.com', 'api.example.com']
if parsed_url.hostname not in allowed_hosts:
```

"Assistant's suggested fix

Validate the URL scheme to ensure it is either http or https. You can use Python's urlparse module to parse the URL and check the scheme.

from urllib.parse import urlparse

parsed_url = urlparse(url)

if parsed_url.scheme not in ['http', 'https']:

return render(request, "Lab/ssrf/ssrf_lab2.html", {"error": "Invalid URL scheme"})

Implement an allowlist of hostnames or IP addresses that are permitted to be accessed. Compare the parsed URL's hostname against this allowlist.

allowed_hosts = ['example.com', 'api.example.com']

if parsed_url.hostname not in allowed_hosts:

return render(request, "Lab/ssrf/ssrf_lab2.html", {"error": "Host not allowed"})

Ensure that the response from the proxied request is not directly forwarded to the user. Instead, process the response data securely before rendering it.

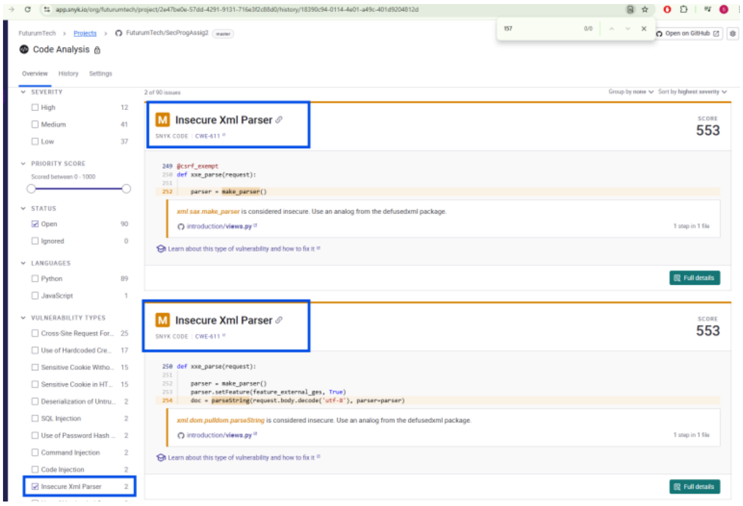
	try: *response = requests.get(url)* *# Process response.content securely* *processed_content = process_response_content(response.content)* *return render(request, "Lab/ssrf/ssrf_lab2.html", {"response": processed_content})* *except:* *return render(request, "Lab/ssrf/ssrf_lab2.html", {"error": "Invalid URL"})* *Ensure that the proxied request uses proper authentication and transport-layer security. This might involve setting headers or using a secure session.* *session = requests.Session()* *session.headers.update({'Authorization': 'Bearer YOUR_TOKEN'})* *response = session.get(url)* *Test the application to ensure that the SSRF vulnerability is mitigated and that the application behaves as expected with valid and invalid URLs.*”

1.2.3. Scanner comparison summary

The findings from the combined analysis using Snyk and Semgrep showed complementary strengths concerning vulnerability detection. Both tools performs really well in identifying vulnerability points in third-party dependencies such as outdated packages where known exploits exist identifying not only CVE (as Semgrep) but Snyk also provided CWE and CVSS rating.

While **Snyk identified 90 vulnerabilities** in the source code, **Semgrep identified 11 more vulnerabilities** although **Snyk identified substantially more vulnerabilities in used dependencies and dockerfile**. Both scans identified a critical risk regarding Server-Side Request Forgery (SSRF) however Semgrep provided more in-dept solution to fix the vulnerability by suggesting changes in the code while in turn, Snyk reduced its recommendation to general guidelines and advices.

However, **Snyk identified two XML vulnerabilities** while **Semgrep failed to recognise** this vulnerability at all:


app.snyk.io/org/futurumtech/project/2e47be0e-57dd-4291-9131-716e3f2c88d0/history/18390c94-0114-4e01-a49c-401d9204812d
157 0/0
FuturumTech › Projects › FuturumTech/SecProgAssig2 master
Open on GitHub
Code Analysis
Overview History Settings

SEVERITY
- ☐ High 12
- ☐ Medium 41
- ☐ Low 37

PRIORITY SCORE
Scored between 0 - 1000

STATUS
- ☑ Open 90
- ☐ Ignored 0

LANGUAGES
- ☐ Python 89
- ☐ JavaScript 1

VULNERABILITY TYPES
- ☐ Cross-Site Request For... 25
- ☐ Use of Hardcoded Cre... 17
- ☐ Sensitive Cookie Witho... 15
- ☐ Sensitive Cookie in HT... 15
- ☐ Deserialization of Untru... 2
- ☐ SQL Injection 2
- ☐ Use of Password Hash ... 2
- ☐ Command Injection 2
- ☐ Code Injection 2
- ☑ Insecure Xml Parser 2

2 of 90 issues — Group by none — Sort by highest severity

M Insecure Xml Parser
SNYK CODE | CWE-611 — SCORE 553

```
@csrf_exempt
def xxe_parse(request):

    parser = make_parser()
```

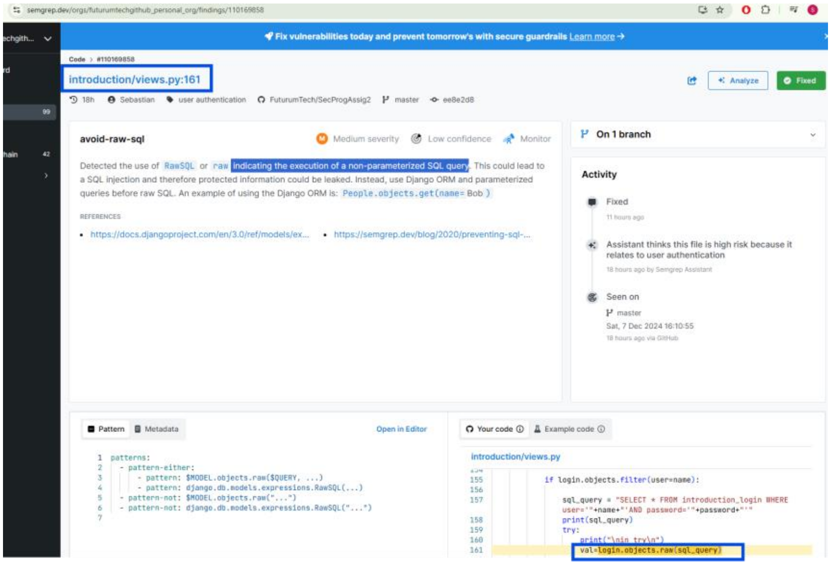
xml.sax.make_parser is considered insecure. Use an analog from the defusedxml package.
introduction/views.py — 1 step in 1 file
Learn about this type of vulnerability and how to fix it
Full details

M Insecure Xml Parser
SNYK CODE | CWE-611 — SCORE 553

```
def xxe_parse(request):

    parser = make_parser()
    parser.setFeature(feature_external_ges, True)
    doc = parseString(request.body.decode('utf-8'), parser=parser)
```

xml.dom.pulldom.parseString is considered insecure. Use an analog from the defusedxml package.
introduction/views.py — 1 step in 1 file
Learn about this type of vulnerability and how to fix it
Full details

On the other hand, **Snyk failed to identify RawSQL vulnerability** indicating the execution of a non-parameterized SQL query while **Semgrep recognised it** at line 161:


semgrep.dev/orgs/futurumtechgithub_personal_org/findings/110169858
Fix vulnerabilities today and prevent tomorrow's with secure guardrails Learn more →
Code › #110169858
Analyze
Fixed
Pattern
Metadata
Open in Editor
Your code
Example code

introduction/views.py:161

18h · Sebastian · user authentication · FuturumTech/SecProgAssig2 · master · ee8e2d8

avoid-raw-sql — Medium severity · Low confidence · Monitor

Detected the use of `RawSQL` or `raw` indicating the execution of a non-parameterized SQL query. This could lead to a SQL injection and therefore protected information could be leaked. Instead, use Django ORM and parameterized queries before raw SQL. An example of using the Django ORM is: `People.objects.get(name= Bob )`

REFERENCES
- https://docs.djangoproject.com/en/3.0/ref/models/ex...
- https://semgrep.dev/blog/2020/preventing-sql-...

On 1 branch

Activity
- Fixed
  11 hours ago
- Assistant thinks this file is high risk because it relates to user authentication
  18 hours ago by Semgrep Assistant
- Seen on
  master
  Sat, 7 Dec 2024 16:10:55
  18 hours ago via GitHub

```
patterns:
  - pattern-either:
      - pattern: $MODEL.objects.raw($QUERY, ...)
      - pattern: django.db.models.expressions.RawSQL(...)
  - pattern-not: $MODEL.objects.raw("...")
  - pattern-not: django.db.models.expressions.RawSQL("...")
```

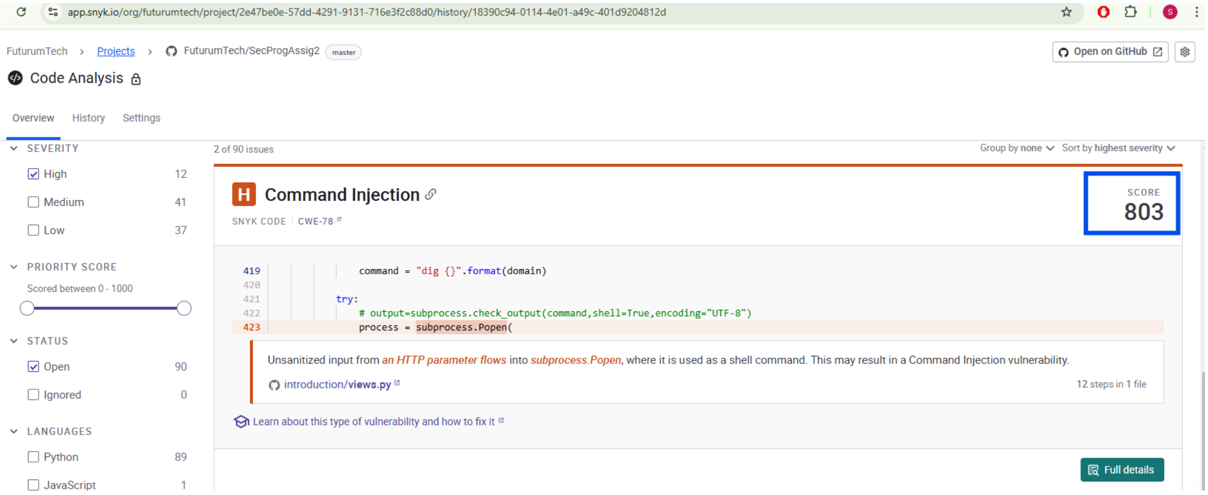
introduction/views.py

```
155          if login.objects.filter(user=name):
156
157              sql_query = "SELECT * FROM introduction_login WHERE
                 user='"+name+"'AND password='"+password+"'"
158              print(sql_query)
159              try:
160                  print("\nin try\n")
161                  val=login.objects.raw(sql_query)
```

To summarise, it appears that Semgrep will be better positioned for direct code analysis because of higher count of vulnerabilities found and better suggested fixes although the best approach would be to use both tools to monitor the same project as it was shown that the tools were able to detect a few different vulnerabilities. Overall, Semgrep proved to be more efficient for the detection of vulnerabilities in application code. This emphasises the complementary nature of both tools in ensuring comprehensive security coverage.

2. Section 2 Fix any two vulnerabilities in the code

2.1. Subprocess-injection



H Command Injection

SNYK CODE | CWE-78

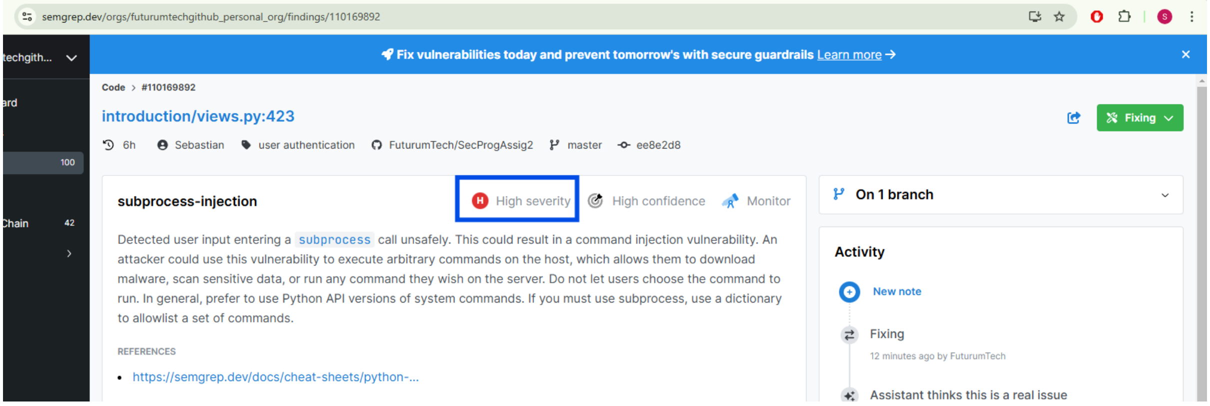
SCORE 803

```
419            command = "dig {}".format(domain)
420
421        try:
422            # output=subprocess.check_output(command,shell=True,encoding="UTF-8")
423            process = subprocess.Popen(
```

Unsanitized input from *an HTTP parameter flows* into *subprocess.Popen*, where it is used as a shell command. This may result in a Command Injection vulnerability.

introduction/views.py — 12 steps in 1 file

Learn about this type of vulnerability and how to fix it


semgrep.dev/orgs/futurumtechgithub_personal_org/findings/110169892
techgith...
Fix vulnerabilities today and prevent tomorrow's with secure guardrails Learn more →
ard
100
Chain 42
Code › #110169892
Fixing
6h
Sebastian
user authentication
FuturumTech/SecProgAssig2
master
ee8e2d8
High severity
High confidence
Monitor
On 1 branch
Activity
New note

introduction/views.py:423

subprocess-injection

Detected user input entering a `subprocess` call unsafely. This could result in a command injection vulnerability. An attacker could use this vulnerability to execute arbitrary commands on the host, which allows them to download malware, scan sensitive data, or run any command they wish on the server. Do not let users choose the command to run. In general, prefer to use Python API versions of system commands. If you must use subprocess, use a dictionary to allowlist a set of commands.

REFERENCES

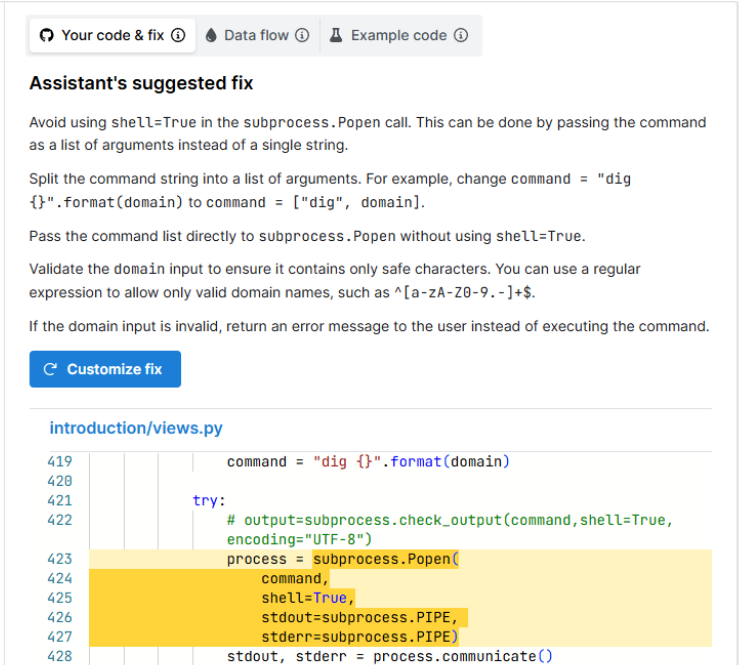
- https://semgrep.dev/docs/cheat-sheets/python-...

Fixing
12 minutes ago by FuturumTech

Assistant thinks this is a real issue

Both Snyk and Semgrep marked the above vulnerability as High severity which consist of: Unsensitised input from HTTP parameters flows into *subprocess.Popen* in functions like *cmd_lab* (line 423). It was categorised by Snyk as CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection').

Suggested fix from SemGrep:



Assistant's suggested fix

Avoid using `shell=True` in the `subprocess.Popen` call. This can be done by passing the command as a list of arguments instead of a single string.

Split the command string into a list of arguments. For example, change `command = "dig {}".format(domain)` to `command = ["dig", domain]`.

Pass the command list directly to `subprocess.Popen` without using `shell=True`.

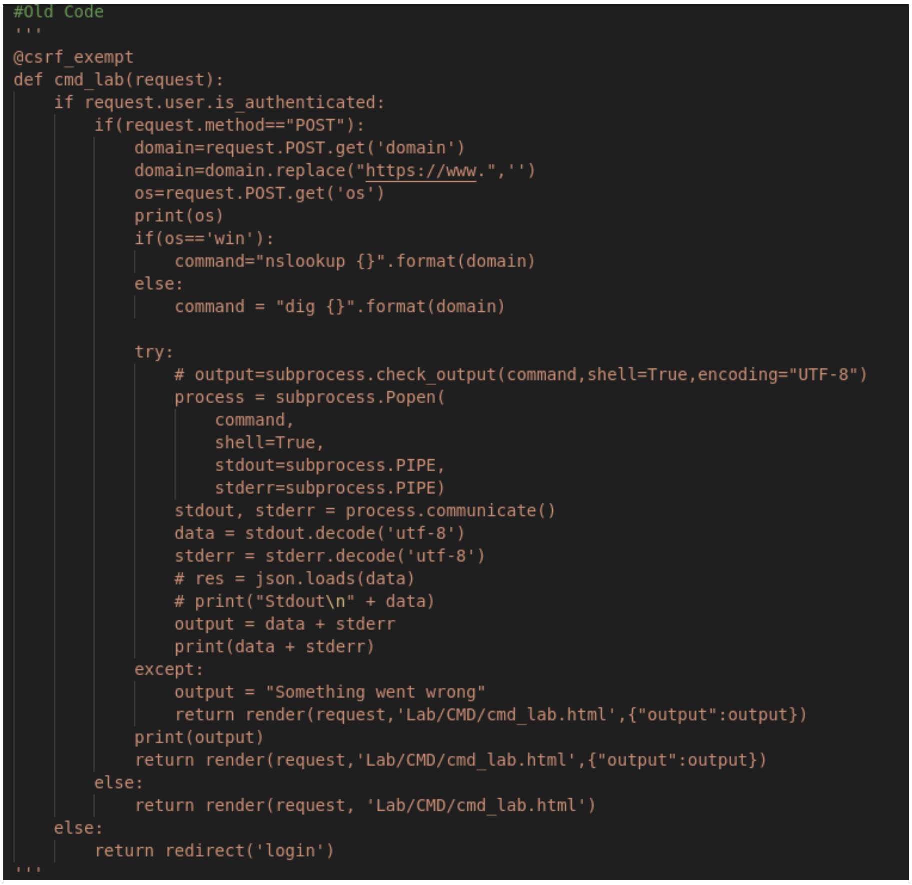
Validate the `domain` input to ensure it contains only safe characters. You can use a regular expression to allow only valid domain names, such as `^[a-zA-Z0-9.-]+$`.

If the domain input is invalid, return an error message to the user instead of executing the command.

introduction/views.py

```
            command = "dig {}".format(domain)

        try:
            # output=subprocess.check_output(command,shell=True,
            encoding="UTF-8")
            process = subprocess.Popen(
                command,
                shell=True,
                stdout=subprocess.PIPE,
                stderr=subprocess.PIPE)
            stdout, stderr = process.communicate()
```

The venerable code was comprised of the following function:

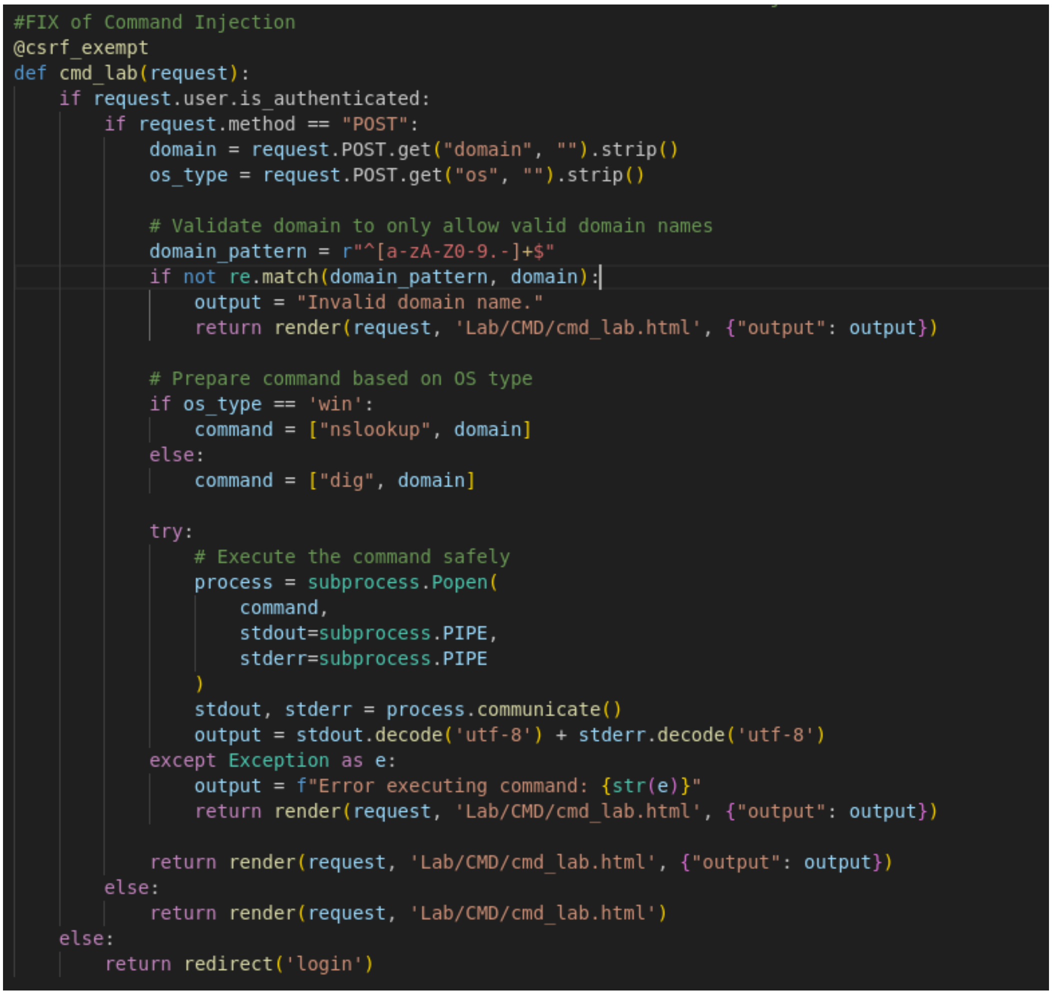


```
#Old Code
'''
@csrf_exempt
def cmd_lab(request):
    if request.user.is_authenticated:
        if(request.method=="POST"):
            domain=request.POST.get('domain')
            domain=domain.replace("https://www.",'')
            os=request.POST.get('os')
            print(os)
            if(os=='win'):
                command="nslookup {}".format(domain)
            else:
                command = "dig {}".format(domain)

            try:
                # output=subprocess.check_output(command,shell=True,encoding="UTF-8")
                process = subprocess.Popen(
                    command,
                    shell=True,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE)
                stdout, stderr = process.communicate()
                data = stdout.decode('utf-8')
                stderr = stderr.decode('utf-8')
                # res = json.loads(data)
                # print("Stdout\n" + data)
                output = data + stderr
                print(data + stderr)
            except:
                output = "Something went wrong"
                return render(request,'Lab/CMD/cmd_lab.html',{"output":output})
            print(output)
            return render(request,'Lab/CMD/cmd_lab.html',{"output":output})
        else:
            return render(request, 'Lab/CMD/cmd_lab.html')
    else:
        return redirect('login')
'''
```

Key Improvements:
- Avoid *shell=True*: Use *subprocess.Popen* with a list of arguments to avoid shell interpretation.

- Input Validation: Use a regular expression to validate the domain to ensure it only contains valid characters for a domain name.

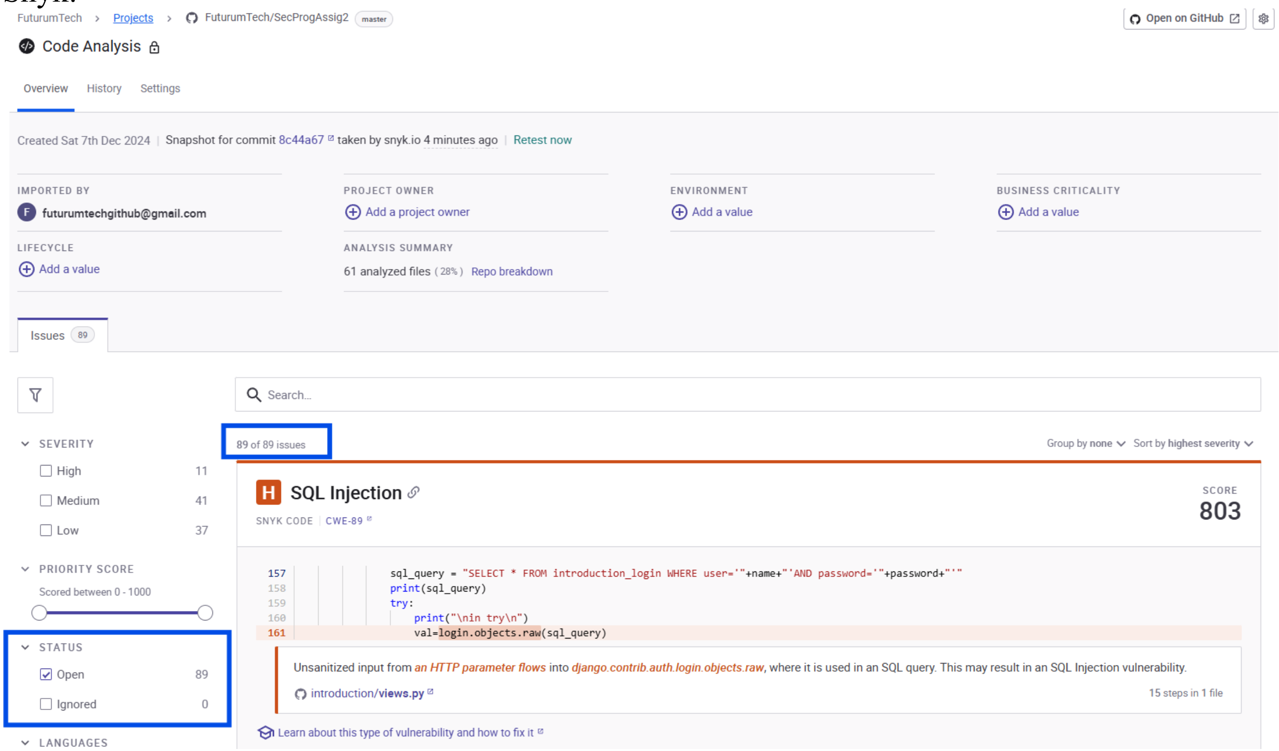
- Command Execution: Pass the command as a list of arguments to *subprocess.Popen*.



```python
#FIX of Command Injection
@csrf_exempt
def cmd_lab(request):
    if request.user.is_authenticated:
        if request.method == "POST":
            domain = request.POST.get("domain", "").strip()
            os_type = request.POST.get("os", "").strip()

            # Validate domain to only allow valid domain names
            domain_pattern = r"^[a-zA-Z0-9.-]+$"
            if not re.match(domain_pattern, domain):
                output = "Invalid domain name."
                return render(request, 'Lab/CMD/cmd_lab.html', {"output": output})

            # Prepare command based on OS type
            if os_type == 'win':
                command = ["nslookup", domain]
            else:
                command = ["dig", domain]

            try:
                # Execute the command safely
                process = subprocess.Popen(
                    command,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE
                )
                stdout, stderr = process.communicate()
                output = stdout.decode('utf-8') + stderr.decode('utf-8')
            except Exception as e:
                output = f"Error executing command: {str(e)}"
                return render(request, 'Lab/CMD/cmd_lab.html', {"output": output})

            return render(request, 'Lab/CMD/cmd_lab.html', {"output": output})
        else:
            return render(request, 'Lab/CMD/cmd_lab.html')
    else:
        return redirect('login')
```

Scan results confirming that the fix decreased the number of vulnerabilities by 1:

Snyk:



FuturumTech › Projects › FuturumTech/SecProgAssig2 master

Open on GitHub

Code Analysis

Created Sat 7th Dec 2024 | Snapshot for commit 8c44a67 taken by snyk.io 4 minutes ago | Retest now

IMPORTED BY: futurumtechgithub@gmail.com

PROJECT OWNER: Add a project owner

ENVIRONMENT: Add a value

BUSINESS CRITICALITY: Add a value

LIFECYCLE: Add a value

ANALYSIS SUMMARY: 61 analyzed files (28%) Repo breakdown

Issues 89

Search...

89 of 89 issues

Group by none Sort by highest severity

SEVERITY

□ High	11
□ Medium	41
□ Low	37

PRIORITY SCORE

Scored between 0 - 1000

STATUS

☑ Open	89
□ Ignored	0

LANGUAGES

H SQL Injection

SNYK CODE | CWE-89

SCORE 803

```
            sql_query = "SELECT * FROM introduction_login WHERE user='"+name+"'AND password='"+password+"'"
            print(sql_query)
            try:
                print("\nin try\n")
                val=login.objects.raw(sql_query)
```

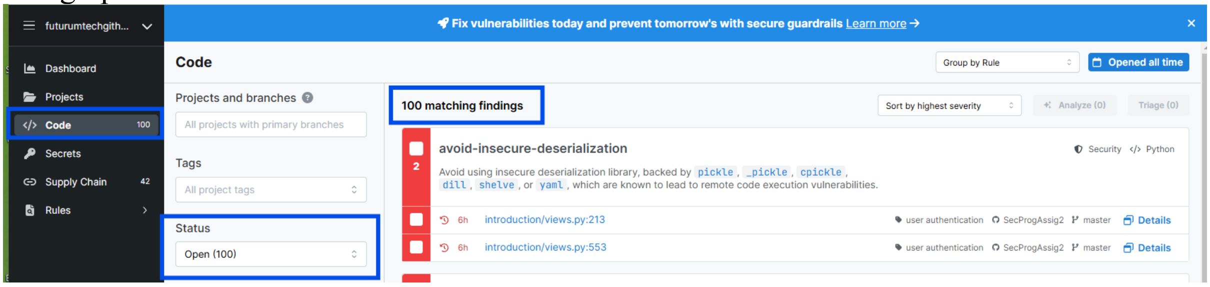
Unsanitized input from *an HTTP parameter flows* into *django.contrib.auth.login.objects.raw*, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

introduction/views.py

15 steps in 1 file

Learn about this type of vulnerability and how to fix it

Semgrep:



Fix vulnerabilities today and prevent tomorrow's with secure guardrails Learn more →

Code

Group by Rule

Opened all time

Projects and branches: All projects with primary branches

Tags: All project tags

Status: Open (100)

100 matching findings

Sort by highest severity

Analyze (0) Triage (0)

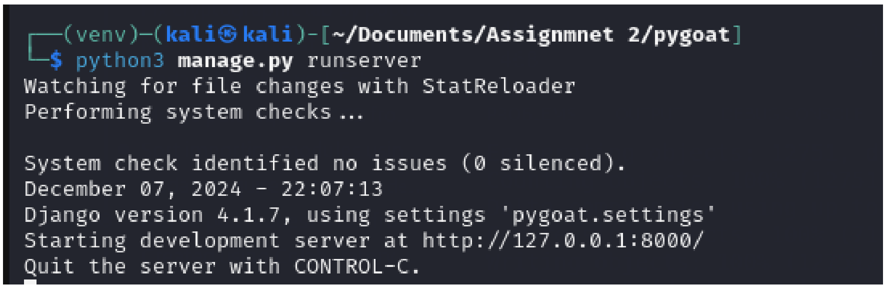
avoid-insecure-deserialization

Security Python

Avoid using insecure deserialization library, backed by `pickle`, `_pickle`, `cpickle`, `dill`, `shelve`, or `yaml`, which are known to lead to remote code execution vulnerabilities.

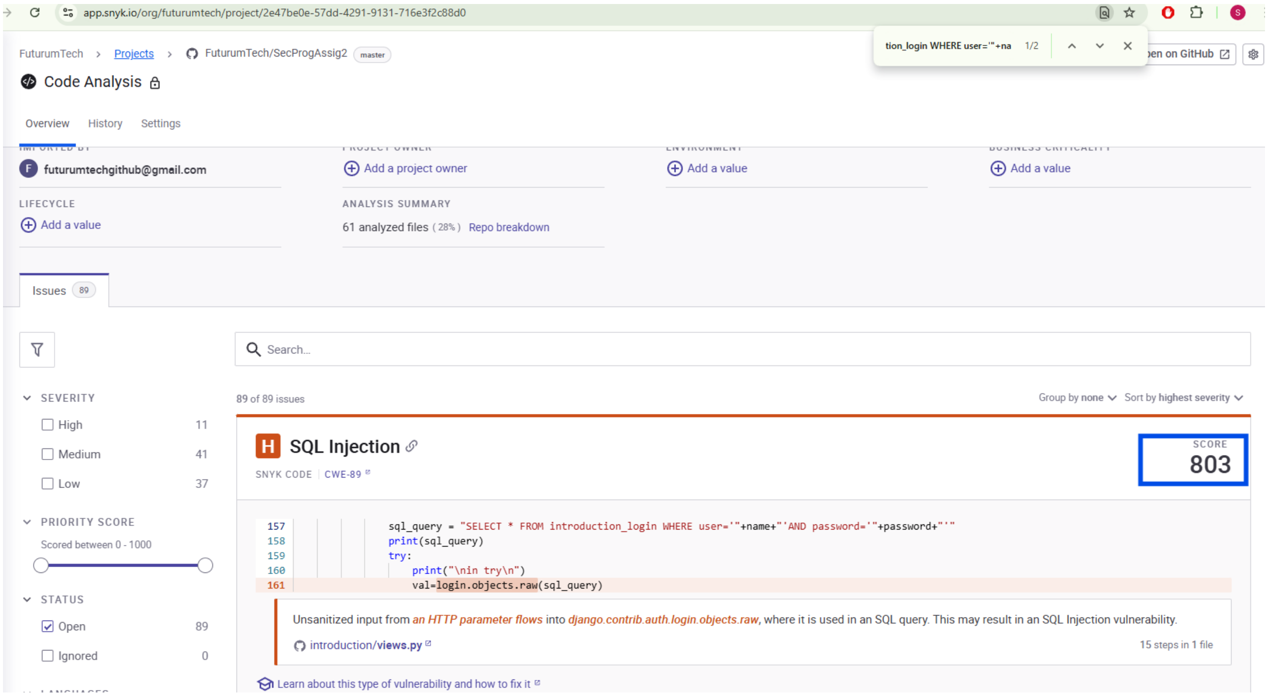
6h	introduction/views.py:213	user authentication	SecProgAssig2	master	Details
6h	introduction/views.py:553	user authentication	SecProgAssig2	master	Details

The application run after fixes without any issues:



```
┌──(venv)─(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
December 07, 2024 - 22:07:13
Django version 4.1.7, using settings 'pygoat.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

2.1. SQL Injection



app.snyk.io/org/futurumtech/project/2e47be0e-57dd-4291-9131-716e3f2c88d0

tion_login WHERE user='"+na 1/2

FuturumTech › Projects › FuturumTech/SecProgAssig2 master

Code Analysis

Overview History Settings

futurumtechgithub@gmail.com

Add a project owner

Add a value

Add a value

LIFECYCLE

Add a value

ANALYSIS SUMMARY

61 analyzed files (28%) Repo breakdown

Issues 89

Search...

SEVERITY

□ High	11
□ Medium	41
□ Low	37

PRIORITY SCORE

Scored between 0 - 1000

STATUS

☑ Open	89
□ Ignored	0

89 of 89 issues

Group by none Sort by highest severity

H SQL Injection

SNYK CODE | CWE-89

SCORE 803

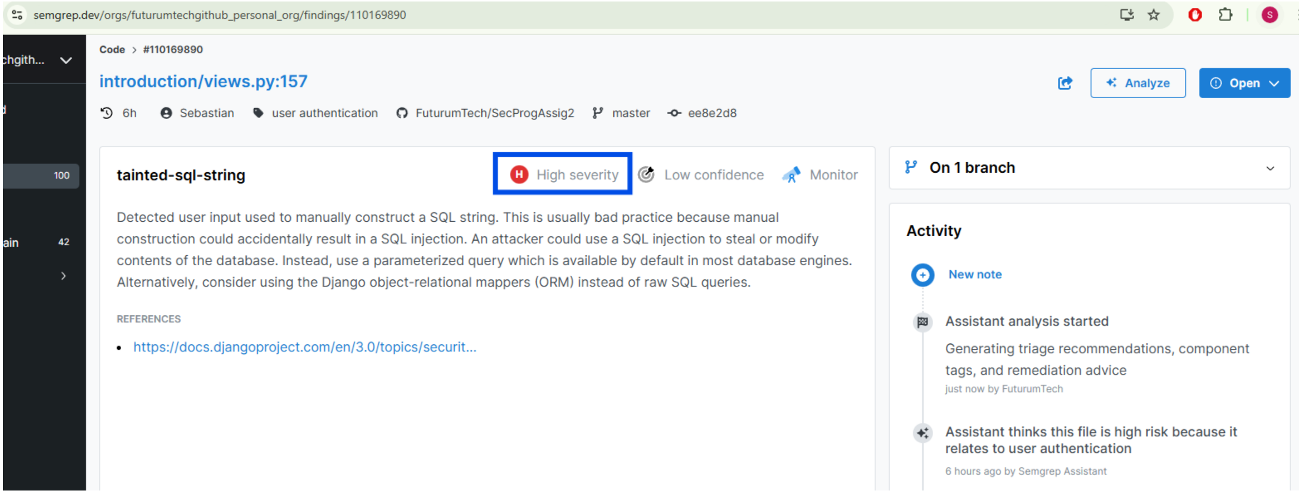
```
157            sql_query = "SELECT * FROM introduction_login WHERE user='"+name+"'AND password='"+password+"'"
158            print(sql_query)
159            try:
160                print("\nin try\n")
161                val=login.objects.raw(sql_query)
```

Unsanitized input from *an HTTP parameter flows* into *django.contrib.auth.login.objects.raw*, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

introduction/views.py

15 steps in 1 file

Learn about this type of vulnerability and how to fix it



semgrep.dev/orgs/futurumtechgithub_personal_org/findings/110169890

Code › #110169890

introduction/views.py:157

6h Sebastian user authentication FuturumTech/SecProgAssig2 master ee8e2d8

Analyze Open

tainted-sql-string

High severity Low confidence Monitor

Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available by default in most database engines. Alternatively, consider using the Django object-relational mappers (ORM) instead of raw SQL queries.

REFERENCES

- https://docs.djangoproject.com/en/3.0/topics/securit...

On 1 branch

Activity

New note

Assistant analysis started

Generating triage recommendations, component tags, and remediation advice

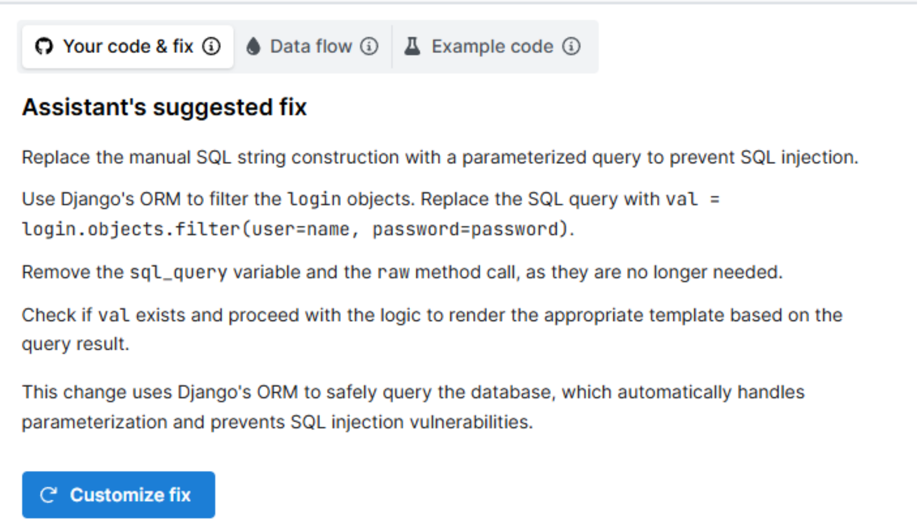
just now by FuturumTech

Assistant thinks this file is high risk because it relates to user authentication

6 hours ago by Semgrep Assistant

Both Snyk and Semgrep marked the above vulnerability as High severity which consist of: Unsanitized input from an HTTP parameter flows into django.contrib.auth.login.objects.raw, where it is used in an SQL query and usage manually construct a SQL string. This may result in an SQL Injection vulnerability (line 157 and 161). Snyk assigned it CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Suggested fix from Semgrep:



Assistant's suggested fix

Replace the manual SQL string construction with a parameterized query to prevent SQL injection.

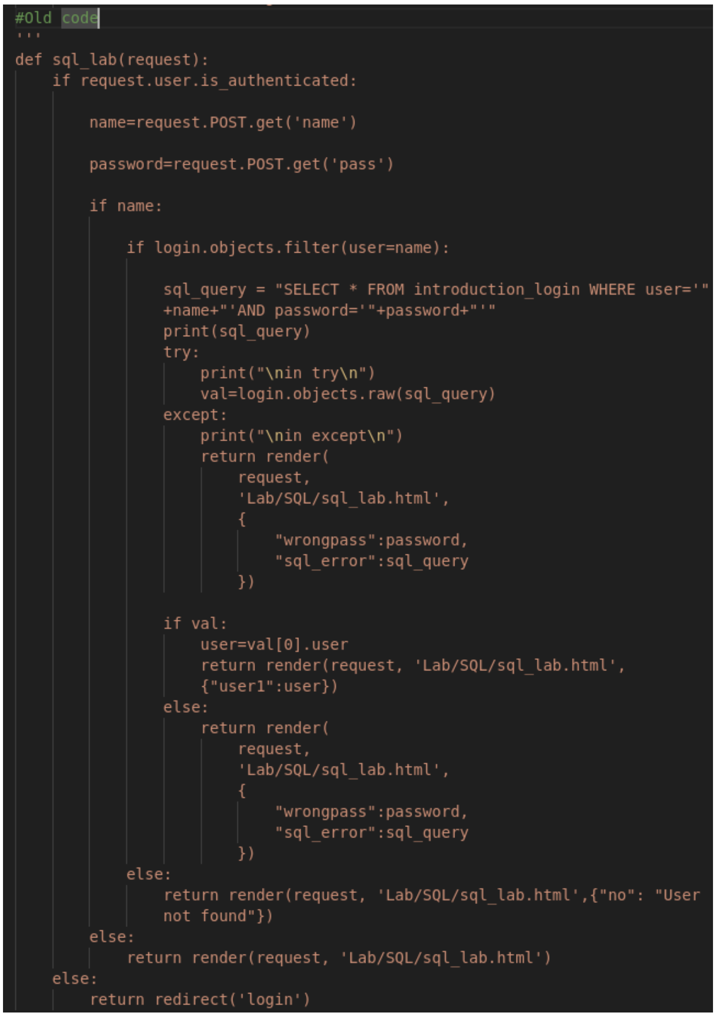
Use Django's ORM to filter the `login` objects. Replace the SQL query with `val = login.objects.filter(user=name, password=password)`.

Remove the `sql_query` variable and the `raw` method call, as they are no longer needed.

Check if `val` exists and proceed with the logic to render the appropriate template based on the query result.

This change uses Django's ORM to safely query the database, which automatically handles parameterization and prevents SQL injection vulnerabilities.

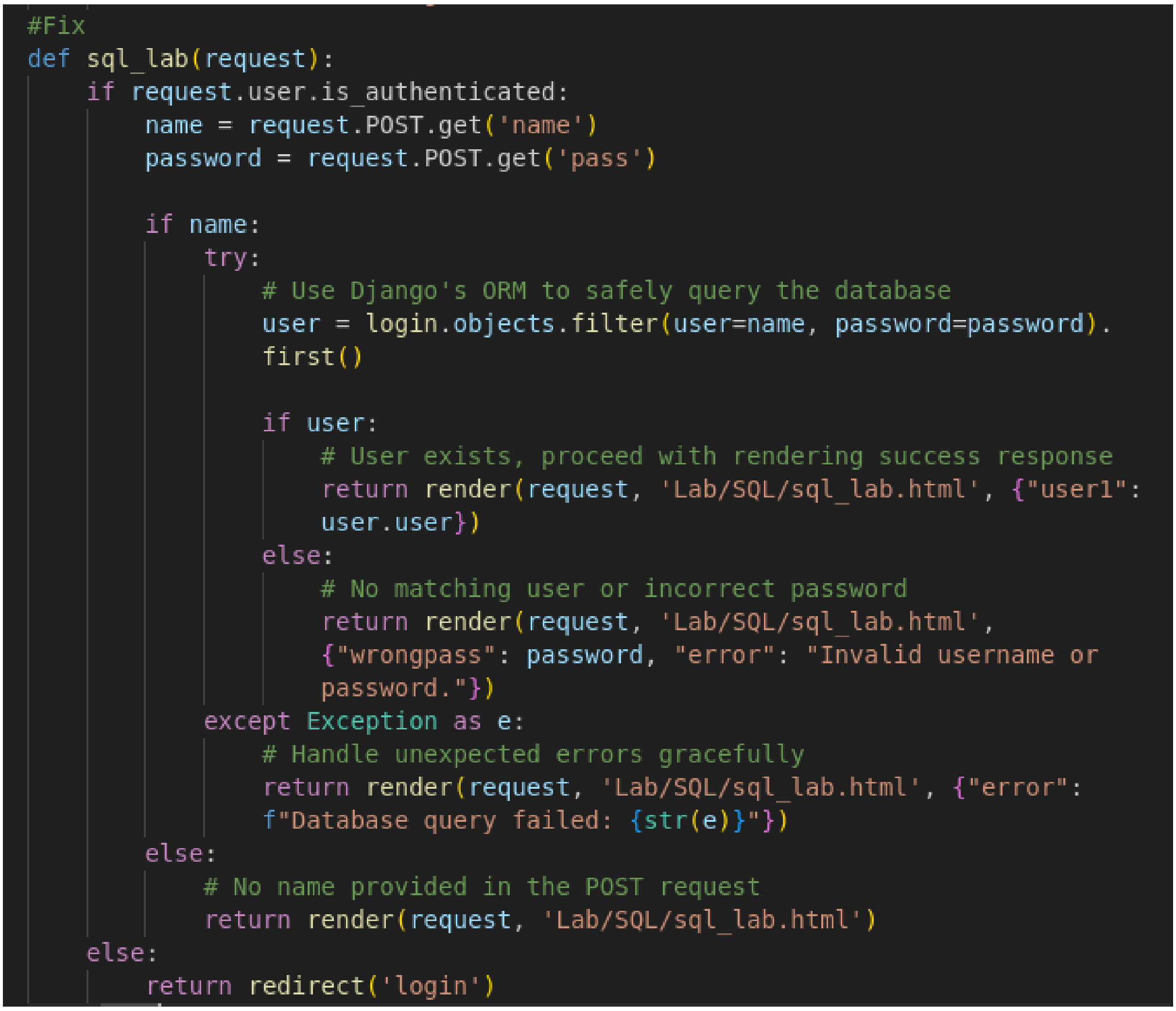
The vulnerable code was comprise of the following function:



```
#Old code
'''
def sql_lab(request):
    if request.user.is_authenticated:

        name=request.POST.get('name')

        password=request.POST.get('pass')

        if name:

            if login.objects.filter(user=name):

                sql_query = "SELECT * FROM introduction_login WHERE user='"
                +name+"'AND password='"+password+"'"
                print(sql_query)
                try:
                    print("\nin try\n")
                    val=login.objects.raw(sql_query)
                except:
                    print("\nin except\n")
                    return render(
                        request,
                        'Lab/SQL/sql_lab.html',
                        {
                            "wrongpass":password,
                            "sql_error":sql_query
                        })

                if val:
                    user=val[0].user
                    return render(request, 'Lab/SQL/sql_lab.html',
                    {"user1":user})
                else:
                    return render(
                        request,
                        'Lab/SQL/sql_lab.html',
                        {
                            "wrongpass":password,
                            "sql_error":sql_query
                        })
            else:
                return render(request, 'Lab/SQL/sql_lab.html',{"no": "User
                not found"})
        else:
            return render(request, 'Lab/SQL/sql_lab.html')
    else:
        return redirect('login')
```

Key Improvements:

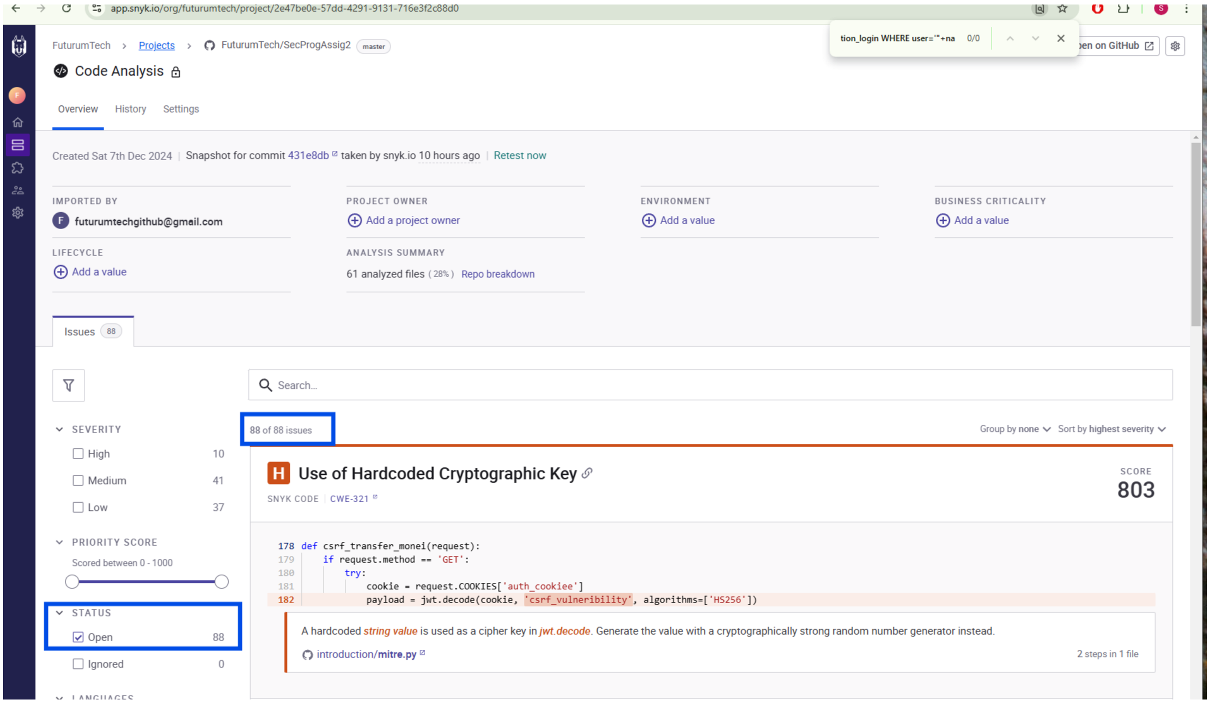
- Raw SQL (SELECT * FROM introduction_login WHERE user='{name}' AND password='{password}') is replaced with Django ORM's filter method.
- Query: login.objects.filter(user=name, password=password) ensures parameterized queries, protecting against SQL injection.
- Used .first() to fetch the first result or None safely, avoiding list indexing issues.
- Added a try-except block to catch database-related exceptions gracefully.
- The variable sql_query is no longer needed because the query construction is handled by the ORM.
- Cleaned up the logic to handle cases when no name is provided or when the user is not found.



```
#Fix
def sql_lab(request):
    if request.user.is_authenticated:
        name = request.POST.get('name')
        password = request.POST.get('pass')

        if name:
            try:
                # Use Django's ORM to safely query the database
                user = login.objects.filter(user=name, password=password).
                first()

                if user:
                    # User exists, proceed with rendering success response
                    return render(request, 'Lab/SQL/sql_lab.html', {"user1":
                    user.user})
                else:
                    # No matching user or incorrect password
                    return render(request, 'Lab/SQL/sql_lab.html',
                    {"wrongpass": password, "error": "Invalid username or
                    password."})
            except Exception as e:
                # Handle unexpected errors gracefully
                return render(request, 'Lab/SQL/sql_lab.html', {"error":
                f"Database query failed: {str(e)}"})
        else:
            # No name provided in the POST request
            return render(request, 'Lab/SQL/sql_lab.html')
    else:
        return redirect('login')
```

Scan results confirming that the fix decreased the number of vulnerabilities by 1:

Snyk:



app.snyk.io/org/futurumtech/project/2e47be0e-57dd-4291-9131-716e3f2c88d0

tion_login WHERE user='"+na 0/0

FuturumTech › Projects › FuturumTech/SecProgAssig2 master

pen on GitHub

Code Analysis

Overview History Settings

Created Sat 7th Dec 2024 | Snapshot for commit 431e8db taken by snyk.io 10 hours ago | Retest now

IMPORTED BY
futurumtechgithub@gmail.com

PROJECT OWNER
Add a project owner

ENVIRONMENT
Add a value

BUSINESS CRITICALITY
Add a value

LIFECYCLE
Add a value

ANALYSIS SUMMARY
61 analyzed files (28%) Repo breakdown

Issues 88

Search...

SEVERITY
- □ High 10
- □ Medium 41
- □ Low 37

PRIORITY SCORE
Scored between 0 - 1000

STATUS
- ☑ Open 88
- □ Ignored 0

LANGUAGES

88 of 88 issues

Group by none Sort by highest severity

H Use of Hardcoded Cryptographic Key

SNYK CODE | CWE-321

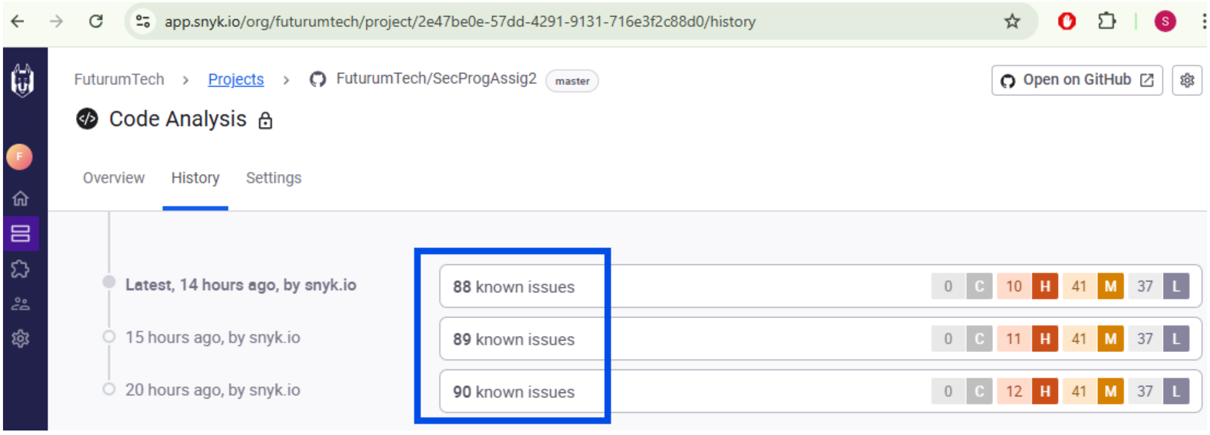
SCORE 803

```
def csrf_transfer_monei(request):
    if request.method == 'GET':
        try:
            cookie = request.COOKIES['auth_cookiee']
            payload = jwt.decode(cookie, 'csrf_vulneribility', algorithms=['HS256'])
```

A hardcoded *string value* is used as a cipher key in *jwt.decode*. Generate the value with a cryptographically strong random number generator instead.

introduction/mitre.py

2 steps in 1 file



app.snyk.io/org/futurumtech/project/2e47be0e-57dd-4291-9131-716e3f2c88d0/history

FuturumTech › Projects › FuturumTech/SecProgAssig2 master

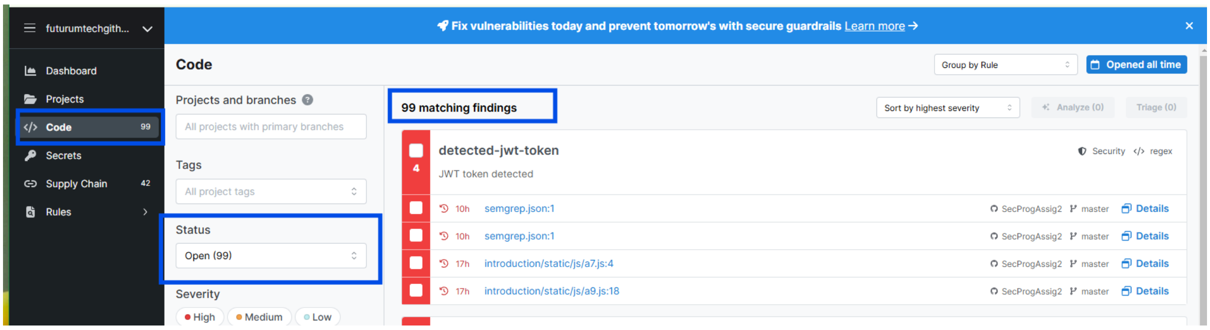
Open on GitHub

Code Analysis

Overview History Settings

		C	H	M	L
Latest, 14 hours ago, by snyk.io	88 known issues	0	10	41	37
15 hours ago, by snyk.io	89 known issues	0	11	41	37
20 hours ago, by snyk.io	90 known issues	0	12	41	37

Semgrep:



Fix vulnerabilities today and prevent tomorrow's with secure guardrails Learn more →

futurumtechgith...

- Dashboard
- Projects
- Code 99
- Secrets
- Supply Chain 42
- Rules

Code

Group by Rule | Opened all time

Projects and branches
All projects with primary branches

Tags
All project tags

Status
Open (99)

Severity
● High ● Medium ● Low

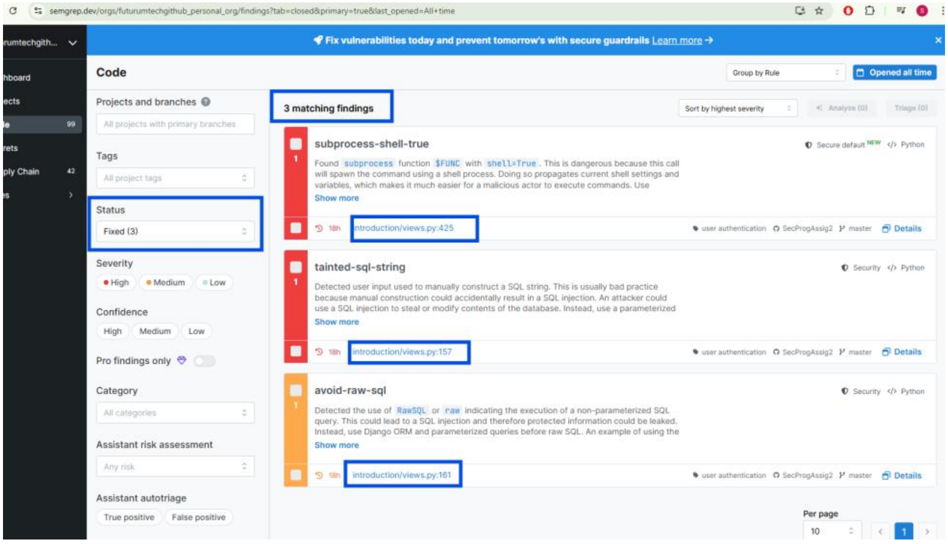
99 matching findings

Sort by highest severity | Analyze (0) | Triage (0)

detected-jwt-token 4
JWT token detected
Security </> regex

10h	semgrep.json:1	SecProgAssig2	master	Details
10h	semgrep.json:1	SecProgAssig2	master	Details
17h	introduction/static/js/a7.js:4	SecProgAssig2	master	Details
17h	introduction/static/js/a9.js:18	SecProgAssig2	master	Details

Semgrep correctly identified fixes of two SQL injections:



semgrep.dev/orgs/futurumtechgithub_personal_org/findings?tab=closed&primary=true&last_opened=All+time

Fix vulnerabilities today and prevent tomorrow's with secure guardrails Learn more →

Code

Group by Rule

Opened all time

Projects and branches: All projects with primary branches

Tags: All project tags

Status: Fixed (3)

Severity: ● High ● Medium ● Low

Confidence: High Medium Low

Pro findings only

Category: All categories

Assistant risk assessment: Any risk

Assistant autotriage: True positive False positive

3 matching findings

Sort by highest severity

subprocess-shell-true — Secure default NEW · Python

Found `subprocess` function `$FUNC` with `shell=True`. This is dangerous because this call will spawn the command using a shell process. Doing so propagates current shell settings and variables, which makes it much easier for a malicious actor to execute commands. Use

Show more

18h introduction/views.py:425 · user authentication · SecProgAssig2 · master · Details

tainted-sql-string — Security · Python

Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized

Show more

18h introduction/views.py:157 · user authentication · SecProgAssig2 · master · Details

avoid-raw-sql — Security · Python

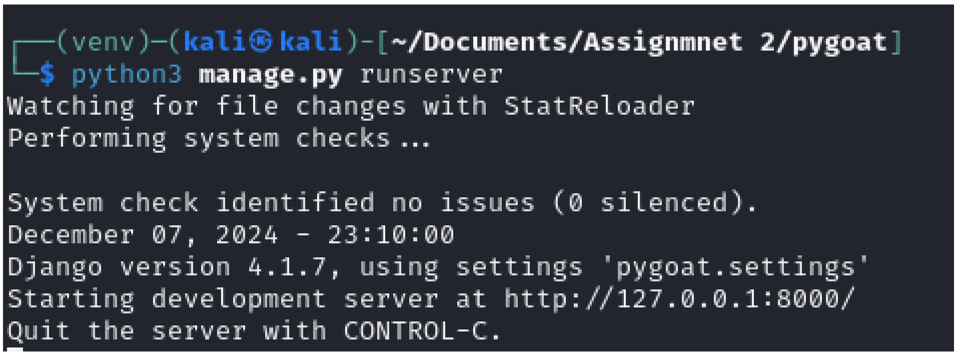
Detected the use of `RawSQL` or `raw` indicating the execution of a non-parameterized SQL query. This could lead to a SQL injection and therefore protected information could be leaked. Instead, use Django ORM and parameterized queries before raw SQL. An example of using the

Show more

18h introduction/views.py:161 · user authentication · SecProgAssig2 · master · Details

Per page: 10

The application run after fixes without any issues:



```
┌──(venv)─(kali㉿kali)-[~/Documents/Assignmnet 2/pygoat]
└─$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
December 07, 2024 - 23:10:00
Django version 4.1.7, using settings 'pygoat.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```