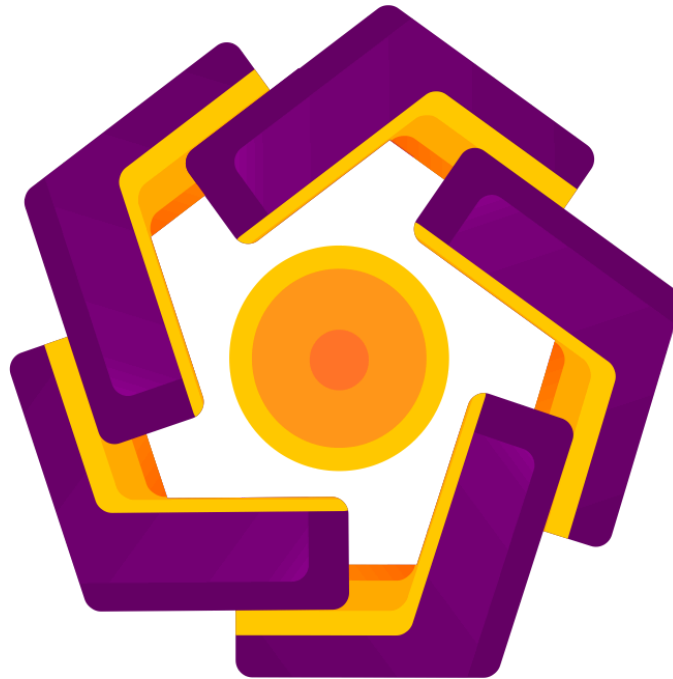


LAPORAN PRAKTIKUM DEVELOPMENT OPERATION



Nama : Frumentios David Ivan Satria

NIM : 23.01.5085

Kelas : D3 Teknik Informatika 03

Waktu : 18 Juni 2025

Pengampu :

Hastari Utama, S.Kom., M.Cs.

**UNIVERSITAS AMIKOM YOGYAKARTA
FAKULTAS ILMU KOMPUTER
SLEMAN 2025**

A. PERSIAPAN AWAL

Melakukan pengecekan tools yang akan digunakan

```
david ~ master ~ docker --version
Docker version 28.3.2, build 578ccf6
david ~ master ~ git --version
git version 2.43.0
```

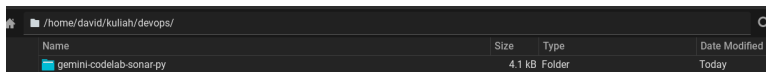
Semua SourceCode saya ada di link GitHub saya berikut :

https://github.com/c0llosseum/devops_laprak11_5085.git

B. MENYIAPKAN PROYEK LATIHAN

1) Membuat folder dengan nama *gemini-codelab-sonar-py* kemudian masuk ke dalam folder tersebut dengan *cd gemini-codelab-sonar-py*

```
david ~ master ~ kuliah > devops > mkdir gemini-codelab-sonar-py
david ~ master ~ kuliah > devops > cd gemini-codelab-sonar-py
david ~ master ~ kuliah > devops > gemini-codelab-sonar-py
```



2) Buat file *app.py* kemudian isilah dengan script seperti dibawah ini

```
david ~ master ~ kuliah > devops > gemini-codelab-sonar-py > nano app.py
```

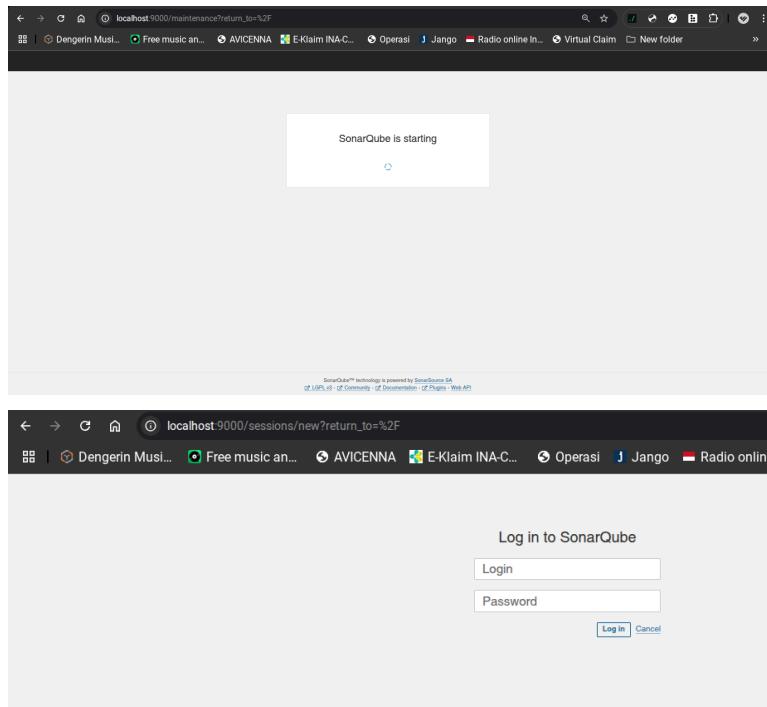
```
app.py ~
1 import sys
2 import hashlib
3 import sqlite3
4
5 API_KEY = "kunci_rahasia_super_penting_12345"
6 DB_PASSWORD = "SuperSecretPassword123"
7
8 def connect_to_services():
9     """Fungsi yang menggunakan kredensial yang ditulis langsung."""
10    print(f"Mencoba menggunakan API Key: {API_KEY}")
11    print(f"Mencoba terhubung ke DB dengan password: {DB_PASSWORD}")
12
13 def create_weak_hash(password_text):
14    """Fungsi ini secara sengaja menggunakan MD5 yang tidak aman."""
15    print(f"Membuat hash MD5 untuk: {password_text}")
16
17    weak_hash = hashlib.md5(password_text.encode()).hexdigest()
18    print(f"Hasil hash MD5: {weak_hash}")
19    return weak_hash
20
21 def search_user(username_input):
22    """Fungsi ini sangat rentan terhadap serangan SQL Injection."""
23
24    conn = sqlite3.connect(':memory:')
25    cursor = conn.cursor()
26    cursor.execute("CREATE TABLE users (id INT, name TEXT)")
27    cursor.execute("INSERT INTO users VALUES (1, 'admin')")
28
29    query = f"SELECT * FROM users WHERE name = '{username_input}'"
30    print(f"Menjalankan query berbahaya: {query}")
31
32    try:
33        cursor.execute(query)
34        result = cursor.fetchall()
35        print(f"Hasil pencarian untuk '{username_input}': {result}")
36    except sqlite3.Error as e:
37        print(f"Terjadi error SQL: {e}")
38
39    conn.close()
40
41
42 def main():
43    """Fungsi utama untuk memanggil semua fungsi yang memiliki kerentanan."""
44    print("--- Menjalankan Aplikasi Latihan Keamanan SonarQube (Versi Agresif) ---")
45
46    connect_to_services()
47    print("-" * 20)
48
49    create_weak_hash("password123")
50    print("-" * 20)
51
52    search_user("admin")
53    print("-" * 20)
54
55    print("--- Aplikasi Selesai ---")
56
57
58 if __name__ == "__main__":
59    main()
```

C. MENJALANKAN SERVER SONARQUBE DENGAN DOCKER

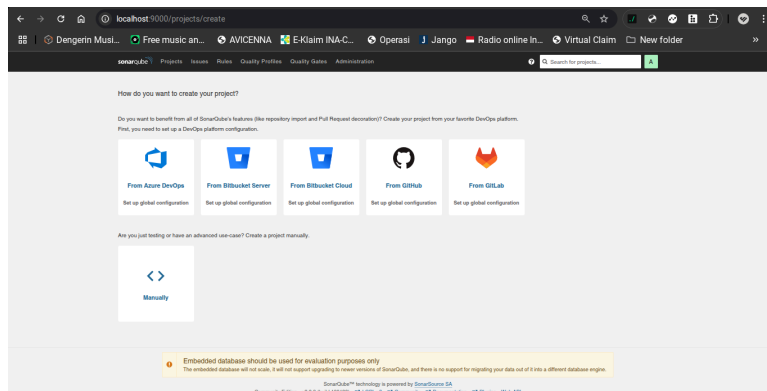
1) Ketikkan perintah berikut pada terminal docker `docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube:lts-community`

```
david@master: ~ > kuliaah > devops > gemini-codelab-sonar-py > docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube:lts-community a4aac9e17643c5fc15122002a04da4be30501540e1b438b244421e3cfd45ce5c
```

2) Buka SonarQube dengan link <http://localhost:9000> pada browser kemudian melakukan login menggunakan Username & Password default : `admin`

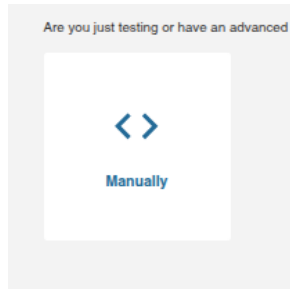


3) Jika berhasil login akan muncul halaman seperti dibawah



D. MEMBUAT PROYEK DAN TOKEN DI SONARQUBE

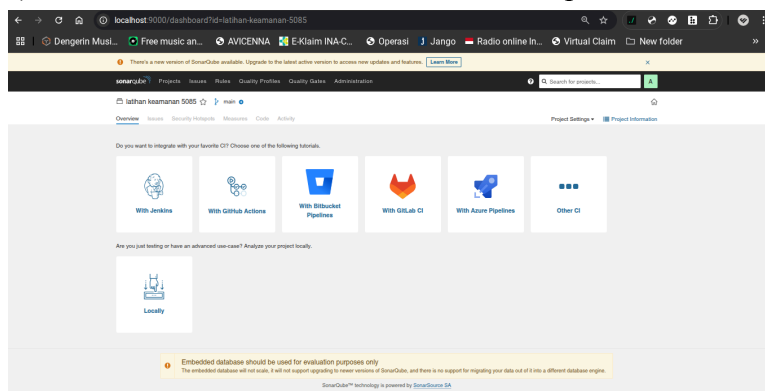
1) Melakukan create new project dengan metode manually



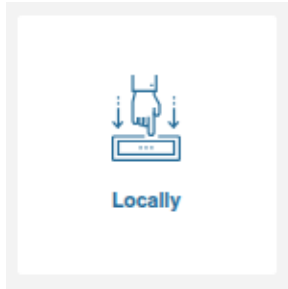
2) Isilah formulirnya sesuai keinginanmu kemudian *Set Up*

A screenshot of the "Create a project" form in SonarQube. The form has a light gray background and a white border. At the top, it says "Create a project". Below that, a note says "All fields marked with * are required". There are three input fields: "Project display name *" with the value "latihan keamanan 5085" and a green checkmark; "Project key *" with the value "latihan-keamanan-5085" and a green checkmark; and "Main branch name *" with the value "main". Below the last field is a link "Learn More". At the bottom left is a blue "Set Up" button. Text below the fields provides details: "Up to 255 characters. Some scanners might override the value you provide." for the display name, and "The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit." for the project key.

3) Setelah berhasil akan muncul halaman seperti dibawah



4) Klik *Locally*



5) Konfigurasi seperti dibawah ini kemudian klik *Generate*

latihan keamanan 5085 ☆ main

Overview Issues Security Hotspots Measures Code Activity

Analyze your project

We initialized your project on SonarQube, now it's up to you to launch analyses!

1 Provide a token

Generate a project token

Token name @ Expires in

scanner-token 30 days Generate

Please note that this token will only allow you to analyze the current project. If you want to use the same token to analyze multiple projects, you need to generate a global token in your user account. See the [documentation](#) for more information.

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#).

2 Run analysis on your project

6) Salin & Copy token yang muncul karena hanya ditampilkan 1x kemudian *Continue*
`sqp_308b0edf606ad98c3ce11e26bcbf77dd975c2b5c`

latihan keamanan 5085 ☆ main

Overview Issues Security Hotspots Measures Code Activity

Analyze your project

We initialized your project on SonarQube, now it's up to you to launch analyses!

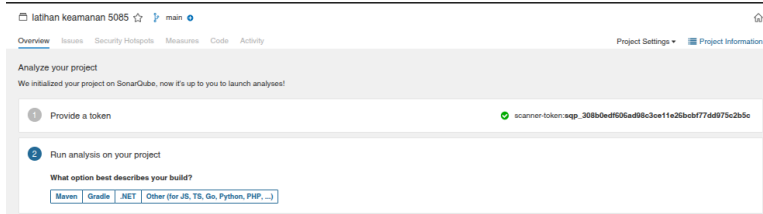
1 Provide a token

scanner-token: **sqp_308b0edf606ad98c3ce11e26bcbf77dd975c2b5c** 🗑

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#).

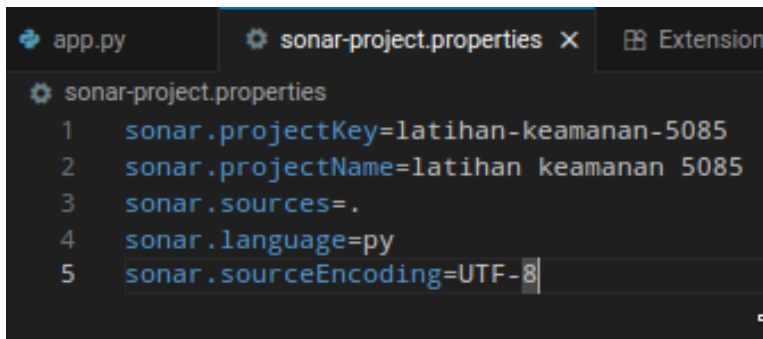
Continue

2 Run analysis on your project



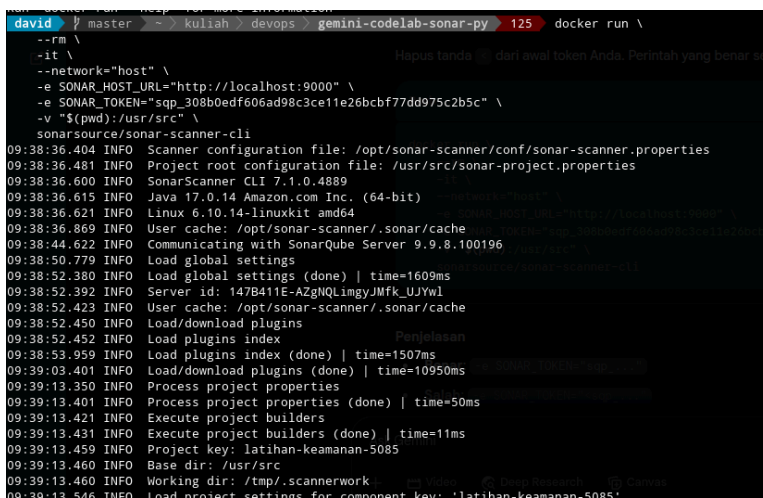
MENJALANKAN ANALISIS DENGAN SONARSCANNER

1) Buat file baru dengan nama *sonar-project.properties* kemudian isi file tersebut dengan script dibawah dan sesuaikan dkonfigurasinya dengan milikmu sendiri.



2) Jalankan scanner dengan script dibawah ini, pada bagian *TOKEN_ANDA* isilah dengan token yang sudah kamu generate terakhir kali.

```
docker run \
  --rm \
  -it \
  --network="host" \
  -e SONAR_HOST_URL="http://localhost:9000" \
  -e SONAR_TOKEN="TOKEN_ANDA" \
  -v "$(pwd):/usr/src" \
  sonarsource/sonar-scanner-cli
```



3) Jika running berhasil akan muncul status *EXECUTION SUCCESS* seperti dibawah

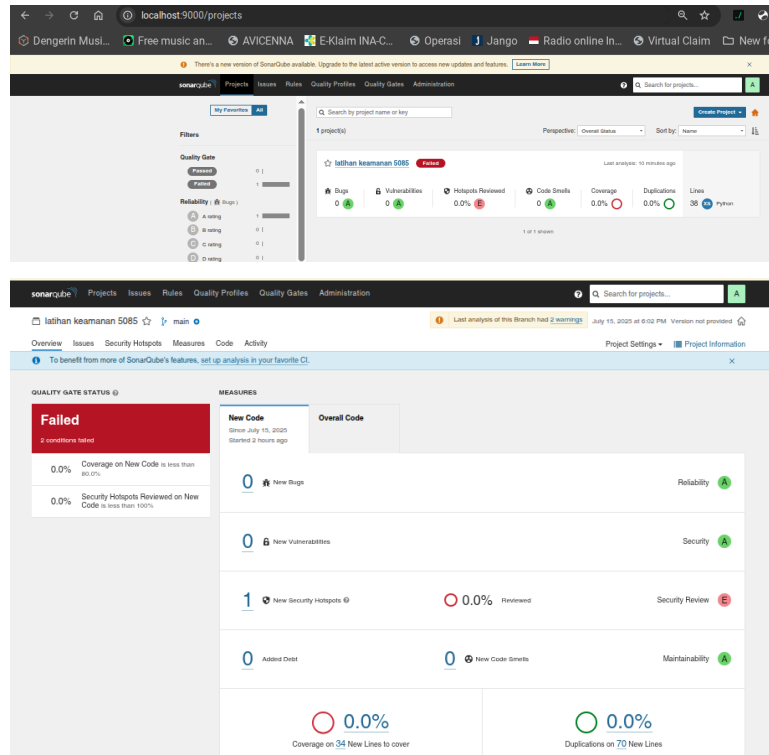
```

09:40:18.432 INFO More about the report processing at http://local
09:40:18.562 INFO Analysis total time: 1:13.676 s
09:40:18.592 INFO EXECUTION SUCCESS
09:40:18.596 INFO Total time: 1:42.290s
david master ~ > kuliah > devops > gemini-codelab-sonar-py

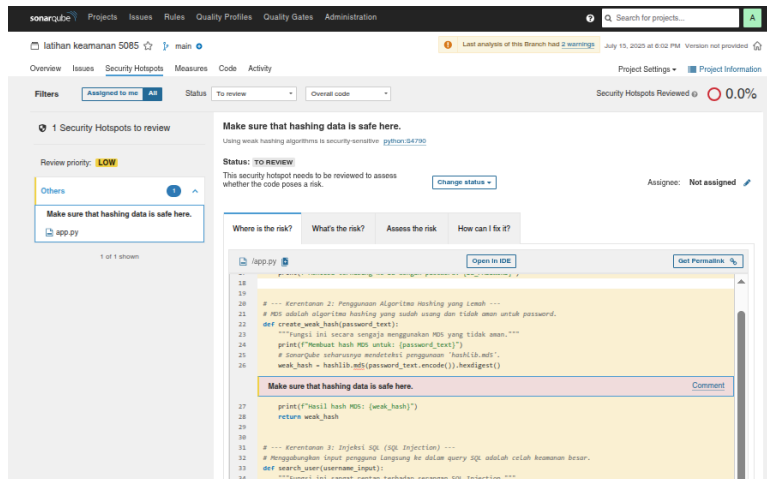
```

E. MELIHAT DAN MENYELESAIKAN HASIL ANALISIS

1) Buka & Refreh kembali SonarQube pada browser maka akan muncul tampilan seperti dibawah ini



2) Lihat pada tab *Security Dashboard* saya menemukan kerentanan "*Make sure that hashing data is safe here*" dan sorotan pada baris kode `weak_hash = hashlib.md5(...)` secara spesifik menunjukkan bahwa SonarQube berhasil mengidentifikasi penggunaan algoritma hashing MD5 yang lemah.



3) Kemudian saya akan membenarkannya dengan script app.py dibawah ini

```

app.py > main
1 import sys
2 import os
3 import bcrypt
4 import sqlite3
5
6 API_KEY = os.getenv("API_KEY", "default_api_key_for_dev")
7 DB_PASSWORD = os.getenv("DB_PASS", "default_pass_for_dev")
8
9 def connect_to_services():
10     """Fungsi ini sekarang menggunakan kredensial dengan cara yang lebih aman."""
11     print(f"Mencoba menggunakan API Key dari environment.")
12     print(f"Mencoba terhubung ke DB dengan password dari environment.")
13
14 def create_strong_hash(password_text):
15     """Fungsi ini menggunakan bcrypt untuk hashing password yang aman."""
16     print(f"Membuat hash bcrypt yang kuat untuk: {password_text}")
17     password_bytes = password_text.encode('utf-8')
18     strong_hash = bcrypt.hashpw(password_bytes, bcrypt.gensalt())
19     print(f"Hasil hash bcrypt: {strong_hash.decode()}")
20     return strong_hash
21
22
23 def search_user_securely(username_input):
24     """Fungsi ini sekarang aman dari serangan SQL Injection."""
25     conn = sqlite3.connect(':memory:')
26     cursor = conn.cursor()
27     cursor.execute("CREATE TABLE users (id INT, name TEXT)")
28     cursor.execute("INSERT INTO users VALUES (1, 'admin')")
29
30     query = "SELECT * FROM users WHERE name = ?"
31     print(f"Menjalankan query yang aman untuk mencari: {username_input}")
32
33     cursor.execute(query, (username_input,))
34     result = cursor.fetchall()
35     print(f"Hasil pencarian untuk '{username_input}': {result}")
36
37     conn.close()
38
39 def main():
40     """Fungsi utama untuk memanggil semua fungsi yang sudah diperbaiki."""
41     print("--- Menjalankan Aplikasi Latihan Keamanan SonarQube (Versi Perbaikan) ---")
42
43     connect_to_services()
44     print("-" * 20)
45
46     create_strong_hash("password123")
47     print("-" * 20)
48
49     search_user_securely("admin")
50     search_user_securely("' OR '1'='1")
51     print("-" * 20)
52
53     print("--- Aplikasi Selesai ---")
54
55 if __name__ == "__main__":
56     main()

```

4) Saya akan menjalankan ulang perintah
*docker run *


```

--rm \
-it \
--network="host" \
-e SONAR_HOST_URL="http://localhost:9000" \
-e SONAR_TOKEN="sqp_308b0edf606ad98c3ce11e26bcbf77dd975c2b5c" \
-v "$(pwd):/usr/src" \
sonarsource/sonar-scanner-cli

```

5) Dapat dilihat output SonarQube sudah tidak muncul threat lagi

The screenshot displays the SonarQube web interface for a project named "latihan keamanan 5085". The top status bar indicates the analysis is "Failed" and occurred 7 minutes ago. A summary row shows the following metrics:

Metric	Value	Quality Gate
Bugs	0	Pass (Green A)
Vulnerabilities	0	Pass (Green A)
Hotspots Reviewed	0	Pass (Green A)
Code Smells	0	Pass (Green A)
Coverage	0.0%	Fail (Red)
Duplications	0.0%	Pass (Green)
Lines	38	Pass (Blue X)

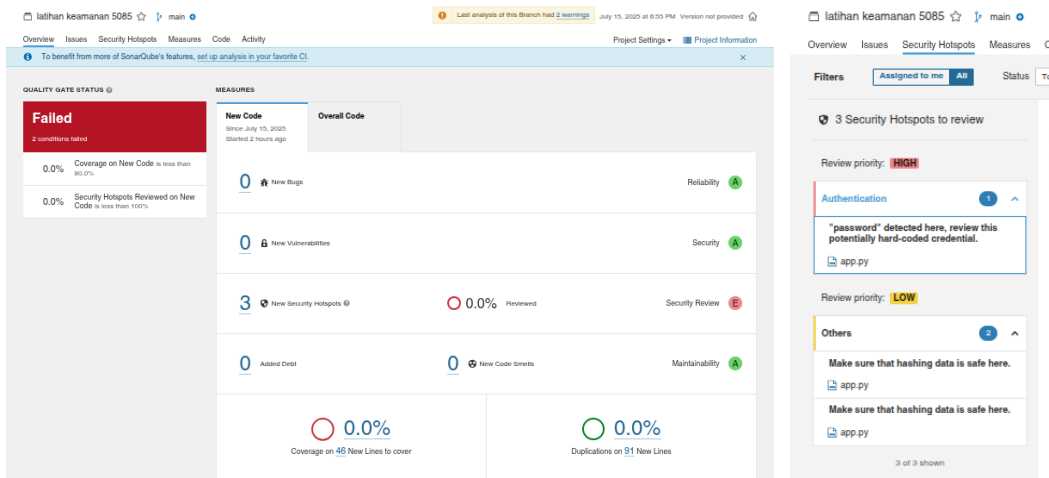
The main content area shows the "Security Hotspots" tab. It displays a message: "There are no Security Hotspots to review. Next time you analyze a piece of code that contains a potential security risk, it will show up here." Below this message is a link to "Learn more about Security Hotspots".

F. TUGAS

1) Kali ini saya akan membuat banyak kerentanan dengan script dibawah

```
app.py ~-
1 # VERSI BENCANA KEAMANAN
2 # Skrip ini sengaja diisi dengan berbagai jenis kerentanan parah
3 # untuk demonstrasi kemampuan deteksi SonarQube secara maksimal.
4
5 import sys
6 import os
7 import hashlib
8 import sqlite3
9 import subprocess # Untuk Command Injection
10 import pickle      # Untuk Deserialisasi yang tidak aman
11
12 # --- Kerentanan 1: Kredensial di mana-mana (Hard-coded Credentials) ---
13 # Tidak hanya satu, tapi beberapa kredensial penting ditulis langsung.
14 DB_PASSWORD_PROD = "PasswordSuperRahasia123!"
15 DB_PASSWORD_DEV = "admin123"
16 API_SECRET_KEY = "kunci-rahasia-milik-perusahaan-jangan-disebar"
17 FTP_USER = "ftp_user"
18 FTP_PASS = "ftp_password_123"
19
20 # --- Kerentanan 2: Penggunaan Algoritma Hashing yang Sangat Lemah ---
21 # Menggunakan MD5 dan SHA1 yang sudah tidak boleh dipakai untuk keamanan.
22 def create_very_weak_hashes(password_text):
23     """Fungsi ini menggunakan MD5 dan SHA1, keduanya tidak aman."""
24     print(f"Membuat hash MD5 & SHA1 untuk: {password_text}")
25
26     # SonarQube akan menandai keduanya sebagai kerentanan.
27     md5_hash = hashlib.md5(password_text.encode()).hexdigest()
28     sha1_hash = hashlib.sha1(password_text.encode()).hexdigest()
29
30     print(f"Hasil MD5: {md5_hash}")
31     print(f"Hasil SHA1: {sha1_hash}")
32     return md5_hash, sha1_hash
33
34 # --- Kerentanan 3: Injeksi SQL (SQL Injection) ---
35 # Celah keamanan klasik di mana input pengguna bisa mengubah query database.
36 def search_product_dangerously(product_name):
37     """Fungsi ini sangat rentan terhadap serangan SQL Injection."""
38     conn = sqlite3.connect(':memory:')
39     cursor = conn.cursor()
40     cursor.execute("CREATE TABLE products (id INT, name TEXT, price INT)")
41
42     # BAGIAN SANGAT BERBAHAYA: Input digabung langsung ke query.
43     query = f"SELECT * FROM products WHERE name = '{product_name}'"
44     print(f"Menjalankan query berbahaya: {query}")
45     cursor.execute(query)
46     print(f"Hasil: {cursor.fetchall()}")
47     conn.close()
48
49 # --- Kerentanan 4: Injeksi Perintah Sistem (Command Injection) ---
50 # Celah fatal di mana input pengguna bisa menjalankan perintah di server.
51 def check_file_status(filename):
52     """Fungsi ini bisa disalahgunakan untuk menjalankan perintah OS apa pun."""
53     # BAGIAN FATAL: Input pengguna menjadi bagian dari perintah sistem.
54     # Penyerang bisa memasukkan "; ls -la" untuk melihat semua file.
55     command = "ls -l" + " " + filename
56     print(f"Menjalankan perintah sistem: {command}")
57     # Penggunaan shell=True dengan input dari luar sangat berbahaya.
58     subprocess.run(command, shell=True, check=True)
59
60
61 # --- Kerentanan 5: Penggunaan 'eval' yang Berbahaya ---
62 # 'eval' mengeksekusi string sebagai kode Python, ini sangat tidak aman.
63 def calculate_expression(expression_string):
64     """Mengevaluasi ekspresi matematika dari string menggunakan eval()."""
65     print(f"Mengevaluasi: {expression_string}")
66     # Penyerang bisa memasukkan '__import__(\"os\").system(\"rm -rf /\")'
67     result = eval(expression_string)
68     print(f"Hasil eval: {result}")
69     return result
70
71 # --- Fungsi Utama untuk Menjalankan Semua Bencana ---
72 def main():
73     print("--- MENJALANKAN SKRIP VERSI BENCANA KEAMANAN ---")
74
75     create_very_weak_hashes("kataSandilemah")
76     print("-" * 30)
77
78     # Penggunaan normal
79     search_product_dangerously("Buku")
80     # Serangan SQL Injection
81     search_product_dangerously("' OR '1'='1'")
82     print("-" * 30)
83
84     # Penggunaan normal
85     check_file_status("app.py")
86     # Serangan Command Injection (akan menampilkan semua file di direktori)
87     check_file_status("app.py; ls -la")
88     print("-" * 30)
89
90     # Penggunaan normal
91     calculate_expression("2 + 2 * 10")
92     # Serangan menggunakan eval (akan menjalankan perintah 'ls' di terminal)
93     calculate_expression("__import__(\"os\").system('ls')")
94     print("-" * 30)
95
96     print("--- SEMUA KERENTANAN TELAH DIJALANKAN ---")
97
98 if __name__ == "__main__":
99     main()
```

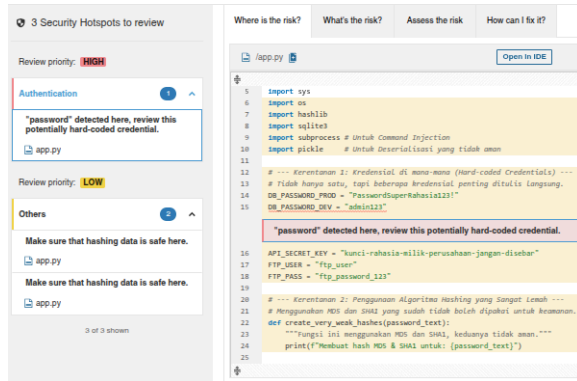
2) Output pada SonarQube mendeteksi 3 kerentanan :



a) Pertama dengan kerentanan priority HIGH dengan indikasi Kredensial yang Ditulis Langsung di Kode (Hard-coded Credential)

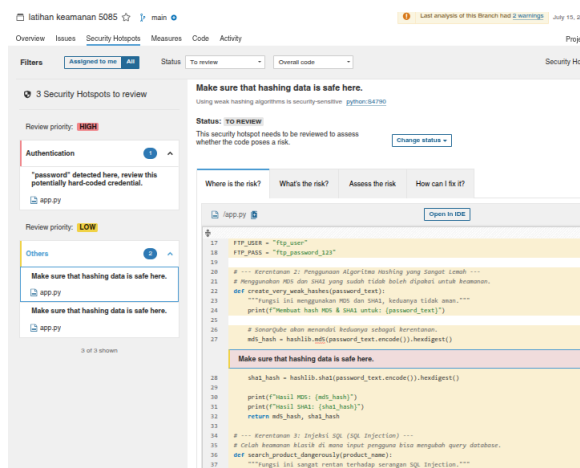
Ini adalah praktik menulis informasi sensitif seperti password, kunci API, atau token langsung di dalam kode sumber (.py, .java, dll.). SonarQube mendeteksinya

dengan mencari kata kunci seperti "password", "secret", "key" pada variabel yang menyimpan nilai teks. Ini berbahaya karena jika kode sumber saya bocor. Misalnya tidak sengaja diunggah ke repositori publik seperti GitHub maka siapa pun bisa langsung melihat dan menyalahgunakan kredensial tersebut untuk mengakses sistem, database, atau layananmu. Seperti menempelkan kunci rumah di pintu depan.



b) Kedua kerentanan dengan priority LOW dengan indikasi Penggunaan Algoritma Hashing yang Lemah (Weak Hashing Algorithm).

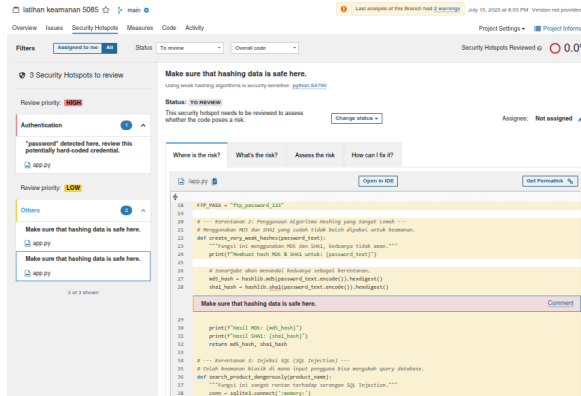
Ini adalah penggunaan algoritma kriptografi yang sudah usang dan terbukti tidak aman untuk tujuan sensitif seperti menyimpan password. Contoh yang paling umum adalah MD5 dan SHA1. Ini berbahaya karena algoritma seperti MD5 sangat cepat dihitung. Ini memungkinkan peretas untuk mencoba miliaran kombinasi password dalam waktu singkat (brute-force attack) atau menggunakan rainbow table (database berisi miliaran hash yang sudah jadi) untuk menemukan password asli dari hash-nya dalam hitungan detik.



c) Ketiga kerentanan dengan priority LOW juga dengan indikasi Injeksi Perintah Sistem (Command Injection)

Ini terjadi karena aplikasi memungkinkan input dari pengguna untuk dieksekusi sebagai perintah di sistem operasi server. Threat ini merupakan salah satu kerentanan paling berbahaya. Penyerang bisa menjalankan perintah apa pun di server Anda,

seperti mencuri semua data (cat /etc/passwd), menghapus semua file (rm -rf /), atau menginstal malware.



3) Untuk perbaikan dapat dilakukan menggunakan script dibawah ini

```
1 # app.py
2 # VERSI PERBAIKAN KEAMANAN LENGKAP
3 # Skrip ini berisi perbaikan untuk semua kerentanan yang ada sebelumnya.
4
5 import sys
6 import os
7 import sqlite3
8 import subprocess
9 # Library yang aman untuk hashing password, perlu di-install: pip install bcrypt
10 import bcrypt
11 # Library yang aman untuk evaluasi ekspresi matematika, perlu di-install: pip install asteval
12 from asteval import Interpreter
13
14 # --- PERBAIKAN 1: Kredensial diambil dari Environment Variables ---
15 # Praktik terbaik adalah tidak menulis kredensial di kode.
16 # Kita mengambilnya dari lingkungan server, dengan nilai default jika tidak ditemukan.
17 DB_PASSWORD_PROD = os.getenv("DB_PASSWORD_PROD", "default_prod_pass")
18 DB_PASSWORD_DEV = os.getenv("DB_PASSWORD_DEV", "default_dev_pass")
19 API_SECRET_KEY = os.getenv("API_SECRET_KEY", "default_api_key")
20 FTP_USER = os.getenv("FTP_USER", "default_ftp_user")
21 FTP_PASS = os.getenv("FTP_PASS", "default_ftp_pass")
22
23 # --- PERBAIKAN 2: Menggunakan Algoritma Hashing yang Kuat (bcrypt) ---
24 # MD5 dan SHA1 diganti dengan bcrypt yang jauh lebih aman dan modern.
25 def create_strong_hash(password_text):
26     """Fungsi ini menggunakan bcrypt untuk hashing password yang aman."""
27     print(f"Membuat hash bcrypt yang kuat untuk: {password_text}")
28     password_bytes = password_text.encode('utf-8')
29     # bcrypt.gensalt() secara otomatis membuat proses hashing sangat aman.
30     strong_hash = bcrypt.hashpw(password_bytes, bcrypt.gensalt())
31     print(f"Hasil hash bcrypt: {strong_hash.decode()}")
32     return strong_hash
33
34 # --- PERBAIKAN 3: Menggunakan Parameterized Queries untuk Mencegah SQL Injection ---
35 # Input pengguna tidak lagi digabung langsung ke dalam query.
36 def search_product_securely(product_name):
37     """Fungsi ini menggunakan aman dari serangan SQL Injection."""
38     conn = sqlite3.connect(':memory:')
39     cursor = conn.cursor()
40     cursor.execute("CREATE TABLE products (id INT, name TEXT, price INT)")
41
42     # BAGIAN AMAN: Input pengguna dilewatkan secara terpisah menggunakan placeholder '?'.
43     # Database akan memperlakukan input sebagai data murni, bukan perintah.
44     query = "SELECT * FROM products WHERE name = ?"
45     print(f"Menjalankan query yang aman untuk mencari: {product_name}")
46
47     # Input dilewatkan sebagai tuple di argumen kedua.
48     cursor.execute(query, (product_name,))
49     print(f"Hasil: {cursor.fetchall()}")
50     conn.close()
51
52 # --- PERBAIKAN 4: Mencegah Command Injection dengan Argumen Terpisah ---
53 # Perintah dan argumennya dipisahkan dalam sebuah list untuk mencegah eksekusi kode arbitrer
54 def check_file_status_securely(filename):
55     """Fungsi ini sekarang lebih aman dari Command Injection."""
56     # Memvalidasi input untuk memastikan tidak ada karakter berbahaya (opsional tapi bagus)
57     if not filename.isalnum() and '.' not in filename:
58         print(f"Error: Nama file tidak valid: {filename}")
59         return
60
61     print(f"Menjalankan perintah sistem secara aman untuk: {filename}")
62     # Perintah dan argumen dipisah dalam list, shell=False adalah default dan lebih aman.
63     try:
64         subprocess.run(["ls", "-l", filename], check=True)
65     except FileNotFoundError:
66         print(f"File tidak ditemukan: {filename}")
67     except subprocess.CalledProcessError as e:
68         print(f"Error saat menjalankan perintah: {e}")
69
70 # --- PERBAIKAN 5: Menggunakan Interpreter Aman sebagai Pengganti 'eval' ---
71 # Menggunakan library 'asteval' yang dirancang untuk mengevaluasi ekspresi matematika dengan aman.
72 def calculate_expression_safely(expression_string):
73     """Mengevaluasi ekspresi matematika dengan aman menggunakan asteval."""
74     print(f"Mengevaluasi dengan aman: {expression_string}")
75     aeval = Interpreter() # Membuat interpreter yang aman
76     # aeval.eval() hanya akan mengeksekusi operasi matematika, bukan perintah sistem.
77     result = aeval.eval(expression_string)
78     print(f"Hasil aman: {result}")
79     return result
80
81 # --- Fungsi Utama untuk Menjalankan Semua Fungsi yang Sudah Diperbaiki ---
82 def main():
83     print("--- MENJALANKAN SKRIP VERSI PERBAIKAN KEAMANAN ---")
84
85     create_strong_hash("kataSandiKuatSekali123")
86     print("-" * 30)
87
88     # Penggunaan normal
89     search_product_securely("Buku")
90     # Serangan SQL Injection (sekarang akan gagal dan tidak menemukan apa-apa)
91     search_product_securely("' OR '1'='1'")
92     print("-" * 30)
93
94     # Penggunaan normal
95     check_file_status_securely("app.py")
96     # Serangan Command Injection (sekarang akan gagal karena input tidak valid)
97     check_file_status_securely("app.py; ls -la")
98     print("-" * 30)
99
100     # Penggunaan normal
101     calculate_expression_safely("2 + 2 * 10")
102     # Serangan menggunakan eval (sekarang akan gagal karena asteval tidak mengemali perintah)
103     try:
104         calculate_expression_safely("__import__('os').system('ls')")
105     except Exception as e:
106         print(f"Gagal mengevaluasi ekspresi berbahaya: {e}")
107     print("-" * 30)
108
109     print("--- SEMUA FUNGSI AMAN TELAH DIJALANKAN ---")
110
111 if __name__ == "__main__":
112     main()
113
```

Penjelasan:

a) Perbaikan 1 dengan menyimpan kredensial di Environment Variables (variabel lingkungan) di server, bukan di dalam kode. Kode kemudian akan membaca variabel tersebut saat berjalan.

b) Perbaikan 2 dengan menggunakan algoritma modern yang dirancang khusus untuk password, seperti bcrypt atau Argon2. Algoritma ini secara inheren lambat dan menyertakan "salt" (data acak) untuk membuat serangan jauh lebih sulit.

c) Perbaikan 3 yaitu jangan pernah menggabungkan input pengguna langsung ke dalam string perintah. Gunakan daftar argumen yang aman, di mana input pengguna selalu

diperlakukan sebagai data, bukan sebagai bagian dari perintah.

4) Berikut ini adalah output SonarQube setelah diperbaiki

The screenshot displays the SonarQube web interface for a project named "latihan keamanan 5085". The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar is present on the right. The left sidebar contains filters for Quality Gate (Failed) and Reliability (A rating). The main content area shows a summary of the project's status, including a "Failed" quality gate and various metrics: Bugs (0), Vulnerabilities (0), Hotspots Reviewed (1), Code Smells (0), Coverage (0.0%), Duplications (0.0%), and Lines (62). Below this, a detailed view of the "QUALITY GATE STATUS" is shown, indicating a "Failed" status with a "0.0%" coverage on new code. The "MEASURES" section provides a breakdown of new code metrics: New Bugs (0), New Vulnerabilities (0), New Security Hotspots (0), Added Debt (0), and New Code Smells (0). The bottom section, "Security Hotspots", shows a message stating "There are no Security Hotspots to review." and provides a link to learn more about Security Hotspots.

QUALITY GATE STATUS

Failed
1 condition failed
Coverage on New Code is less than 80.0%
0.0%

MEASURES

New Code
Since July 15, 2020
Started 9 hours ago

Overall Code

0 New Bugs Reliability

0 New Vulnerabilities Security

0 New Security Hotspots Reviewed Security Review

0 Added Debt 0 New Code Smells Maintainability

Coverage on 58 New Lines to cover 0.0%

Duplications on 105 New Lines 0.0%

Security Hotspots

There are no Security Hotspots to review.
Next time you analyze a piece of code that contains a potential security risk, it will show up here.
[Learn more about Security Hotspots](#)

```
PS D:\23.01.5085> mkdir gemini-codelab-sonar-py

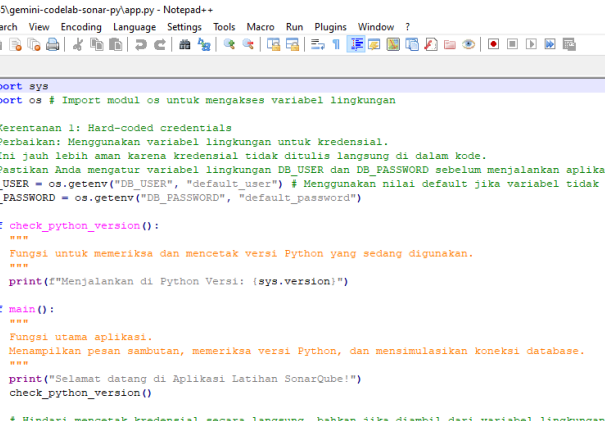
Directory: D:\23.01.5085

Mode                LastWriteTime         Length Name
----                -
d-----            6/18/2025    7:30 AM          gemini-codelab-sonar-py

PS D:\23.01.5085> cd gemini-codelab-sonar-py
PS D:\23.01.5085\gemini-codelab-sonar-py> notepad app.py
PS D:\23.01.5085\gemini-codelab-sonar-py> ls

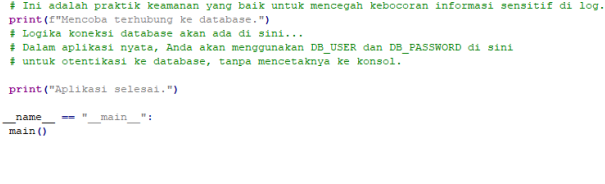
Directory: D:\23.01.5085\gemini-codelab-sonar-py

Mode                LastWriteTime         Length Name
----                -
-a-----            6/18/2025    7:33 AM          1472 app.py
```



```
1 import sys
2 import os # Import modul os untuk mengakses variabel lingkungan
3
4 # Kerentanan 1: Hard-coded credentials
5 # Perbaikan: Menggunakan variabel lingkungan untuk kredensial.
6 # Ini jauh lebih aman karena kredensial tidak ditulis langsung di dalam kode.
7 # Pastikan Anda mengatur variabel lingkungan DB_USER dan DB_PASSWORD sebelum menjalankan aplikasi.
8 DB_USER = os.getenv("DB_USER", "default_user") # Menggunakan nilai default jika variabel tidak ditemukan
9 DB_PASSWORD = os.getenv("DB_PASSWORD", "default_password")
10
11 def check_python_version():
12     """
13     Fungsi untuk memeriksa dan mencetak versi Python yang sedang digunakan.
14     """
15     print(f"Menjalankan di Python Versi: {sys.version}")
16
17 def main():
18     """
19     Fungsi utama aplikasi.
20     Menampilkan pesan sambutan, memeriksa versi Python, dan mensimulasikan koneksi database.
21     """
22     print("Selamat datang di Aplikasi Latihan SonarQube!")
23     check_python_version()
24
25     # Hindari mencetak kredensial secara langsung, bahkan jika diambil dari variabel lingkungan.
26     # Ini adalah praktik keamanan yang baik untuk mencegah kebocoran informasi sensitif di log.
27     print(f"Mencoba terhubung ke database.")
28     # Logika koneksi database akan ada di sini...
29     # Dalam aplikasi nyata, Anda akan menggunakan DB_USER dan DB_PASSWORD di sini
30     # untuk otentikasi ke database, tanpa mencetaknya ke konsol.
31
32     print("Aplikasi selesai.")
33
34 if __name__ == "__main__":
35     main()
```

Search results for "sonarqube"



```
PS D:\23.01.5085\gemini-codelab-sonar-py> docker pull sonarqube
Using default tag: latest
latest: Pulling from library/sonarqube
d9d352c11bbd: Already exists
35394ce74242: Already exists
a156ac8fc59e: Already exists
b4d578382bab: Already exists
b894632d8872: Pull complete
9b3ac574fd21: Downloading [>
a63bd2282757: Download complete
4f4fb726e6f54: Download complete
```

```
PS D:\23.01.5885\gemini-code\lab-sonar-py> docker run -d --name sonarqube -p 9998:9998 -p 9992:9992 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
89dc0e9ae2: Downloading [=====] 14.62MB/29.53MB
31436812ac5b: Downloading [=====] 14.74MB/16.15MB
2d16eb7e762: Downloading [=====] 10.93MB/46.96MB
ac81863d97cb: Waiting
2d16dfcecc1b: Waiting
a7343c0b8fb1: Waiting
7c8e698ec954: Waiting
4f4fb780ef54: Waiting
```

