

# (1) Overly Permissive File Permissions

- **Overly permissive file permissions** occur when a file or directory is created with access rights that are too broad (e.g., anyone can read/write/execute it).
- This allows **unauthorized users or processes** to modify, delete, or steal sensitive data.
- In Linux, using permissions like `chmod 777` or Python's `os.chmod(path, 0o777)` is a common example.
- Attackers can exploit this to replace config files, inject malicious code, or escalate privileges.
- Secure practice: grant the **minimum necessary access** (e.g., `600` or `700`) and avoid world-writable directories unless explicitly required.

Filename: [main.py](#)

Line No: 112

The screenshot shows a GitHub Code scanning alert for a file named `main.py`. The alert is titled "Overly permissive file permissions" and was triggered 19 minutes ago. It has a severity of "High". The alert details a specific line of code (line 112) where a file is being chmoded to 0777, which is described as giving write permissions to everyone. A tooltip explains that this is an overly permissive mask. The alert also notes a potential security vulnerability related to file creation permissions. There are sections for "Development" (with a note to create a new branch), "Tags" (security), and "Weaknesses" (CWE-732). The bottom of the alert shows a task to implement examples of common Python vulnerabilities, which is marked as verified. The footer of the page includes standard GitHub navigation links like Terms, Privacy, Security, Status, Community, Docs, Contact, and Manage cookies.

Screenshot: Overly Permissive File Permissions

## (2) Insecure Temporary File

- An **insecure temporary file** is created in a way that allows another program or attacker to access or modify it before the legitimate program uses it.
- If a temp file is created with predictable names (like `/tmp/data.txt`), an attacker can create or replace it beforehand.
- This can cause data leaks, unauthorized file modification, or even **code execution** if the program later runs data from that file.
- Temp directories like `/tmp` are shared, so files placed there need secure random names and proper permissions.
- Safe practice: use secure temp functions (e.g., `tempfile.NamedTemporaryFile()` in Python) to avoid guessing and tampering.

Filename: [main.py](#)

Line No: 103

The screenshot shows a GitHub Code Scanning alert for a file named `main.py`. The alert is titled "Insecure temporary file" and was first detected 22 minutes ago. It is categorized as "High" severity. The code snippet at line 103 shows a call to `tempfile.mktemp` with a predictable name, which is flagged as deprecated and potentially dangerous. The alert includes a "Generate fix" button and a "Dismiss alert" button. Below the alert, there is a detailed description of the vulnerability, tool information (CodeQL), and associated GitHub actions. At the bottom, there are links to GitHub's Terms of Service, Privacy Policy, and other community resources.

Screenshot: Insecure Temporary File

# (3) Request without Certificate Validation

- **Request without certificate validation** means a program makes an HTTPS request but does **not verify the server's SSL certificate**.
- This skips the security check that ensures the server is **authentic** and not an attacker pretending to be the server.
- Without validation, the connection can be intercepted using a **man-in-the-middle attack**.
- Attackers can steal sensitive data (passwords, API keys) or inject malicious responses.
- It is often seen in code like `verify=False` in Python `requests`, and should be avoided except in controlled testing environments.

Filename: [main.py](#)

Line No: 64

The screenshot shows a GitHub Code Scanning alert for a repository named 'fraxhost / code-scanning-codeql'. The alert is titled 'Request without certificate validation' and is categorized under 'Code scanning alerts / #4'. It was opened 13 minutes ago. The severity is marked as 'High'. The affected file is 'main.py:64' with the following code snippet:

```
61
62 # 8) Disabling SSL certificate verification
63 def disable_ssl_verification_example(url: str) -> requests.Response:
64     resp = requests.get(url, verify=False) # disabled TLS verification
This request may run without certificate validation because it is disabled.
CodeQL
65     return resp
66
67 # 9) Logging sensitive data
```

The alert message states: 'Encryption is key to the security of most, if not all, online communication. Using Transport Layer Security (TLS) can ensure that communication cannot be interrupted by an interloper. For this reason, it is unwise to disable the verification that TLS provides. Functions in the `requests` module provide verification by default, and it is only when explicitly turned off using `verify=False` that no verification occurs.' There is a 'Show more' link below this message.

Below the code snippet, there are sections for 'Tool', 'Rule ID', 'Query', and 'View source'. The 'Tool' is 'CodeQL', 'Rule ID' is 'py/request-without-cert-validation', 'Query' is 'View source'. The 'Severity' is 'High'. Under 'Assignees', it says 'No one - Assign yourself'. Under 'Affected branches', it lists 'main / default' with a note 'First detected 41 minutes ago'. Under 'Development', there is a link to 'Create a new branch'. Under 'Tags', it has 'security'. Under 'Weaknesses', it lists 'CWE-295'. At the bottom, there are links for 'First detected in commit 40 minutes ago', 'Add vulnerable examples for CodeQL testing ...', 'main.py:64 on branch main', and 'Appeared in branch main 40 minutes ago'.

Screenshot: Request without Certificate Validation

# (4) Use of a Broken or Weak Cryptographic Hashing Algorithm on Sensitive Data

- **Weak cryptographic hashes** like MD5 or SHA-1 are no longer secure for protecting sensitive data such as passwords or tokens.
- They are vulnerable to **collision attacks**, where two different inputs produce the same hash, compromising data integrity.
- Using them on passwords allows attackers to **precompute hash tables (rainbow tables)** to reverse them easily.
- Sensitive data hashed with weak algorithms can be **easily cracked or tampered**, leading to breaches or impersonation.
- Secure practice: use modern, strong hashing algorithms like **SHA-256** with salts or **password-specific hashing functions** like **bcrypt**, **scrypt**, or **Argon2**.

Filename: [main.py](#)

Line No: 39

The screenshot shows a GitHub Code Scanning alert for a file named main.py. The alert is titled "Use of a broken or weak cryptographic hashing algorithm on sensitive data". It highlights a line of code at line 39: "h = hashlib.md5() # weak hash". A tooltip explains that using MD5 is insecure for password hashing because it is not computationally expensive. The alert is categorized as "High" severity. It includes sections for "Affected branches" (main), "Development" (with a note to create a new branch), and "Weaknesses" (listing CVE-327, CVE-328, and CVE-916). The alert was first detected 41 minutes ago and has been verified. There are also links to "Copilot Autofix for CodeQL" and "Generate fix".

Screenshot: Use of a Broken or Weak Cryptographic Hashing Algorithm on Sensitive Data

# (5) Clear-Text Logging of Sensitive Information

- **Clear-text logging** occurs when sensitive information like passwords, API keys, or personal data is written directly to logs.
- Anyone with access to the logs (developers, administrators, or attackers) can **read confidential data** easily.
- Logs may be stored long-term or sent to external systems, increasing the risk of **data leaks**.
- Logging sensitive data can lead to **compliance violations** with standards like GDPR, HIPAA, or PCI-DSS.
- Secure practice: **mask, redact, or avoid logging** sensitive information, and log only non-sensitive identifiers or anonymized data.

Filename: [main.py](#)

Line No: 69

The screenshot shows a GitHub security alert for a code scanning issue. The alert is titled "Clear-text logging of sensitive information" and is associated with the file "main.py" at line 69. The code snippet at line 69 is:

```
67 # 9) Logging sensitive data
68 def sensitive_logging_example(username: str, password: str) -> None:
69     logging.warning(f"User login attempt: user={username} password={password}")
This expression logs sensitive data (password) as clear text.
```

The alert has a severity of "High". It includes a "Generate fix" button and a "Copilot Autofix for CodeQL" link. The "Affected branches" section shows the main branch was first detected 42 minutes ago. The "Tags" section includes "security". The "Weaknesses" section lists three CVEs: CVE-312, CVE-359, and CVE-532. The footer of the page includes standard GitHub links like Terms, Privacy, Security, Status, Community, Docs, Contact, Manage cookies, and a note about not sharing personal information.

Screenshot: Clear-Text Logging of Sensitive Information