TOOLS

- Bandit – detects common security problems (bandit -r .)
- SEMGREP - detects common security problems (semgrep scan --config=p/python)
- Pip-audit – scans for libraries and packages with vulnerabilities (pip-audit)
- ChatGPT – manual code review by AI agent

Here the two backdoors have already been provided (Impact if it is exploited, Recommendations)

Weak keys – poc_weak_key_register.py

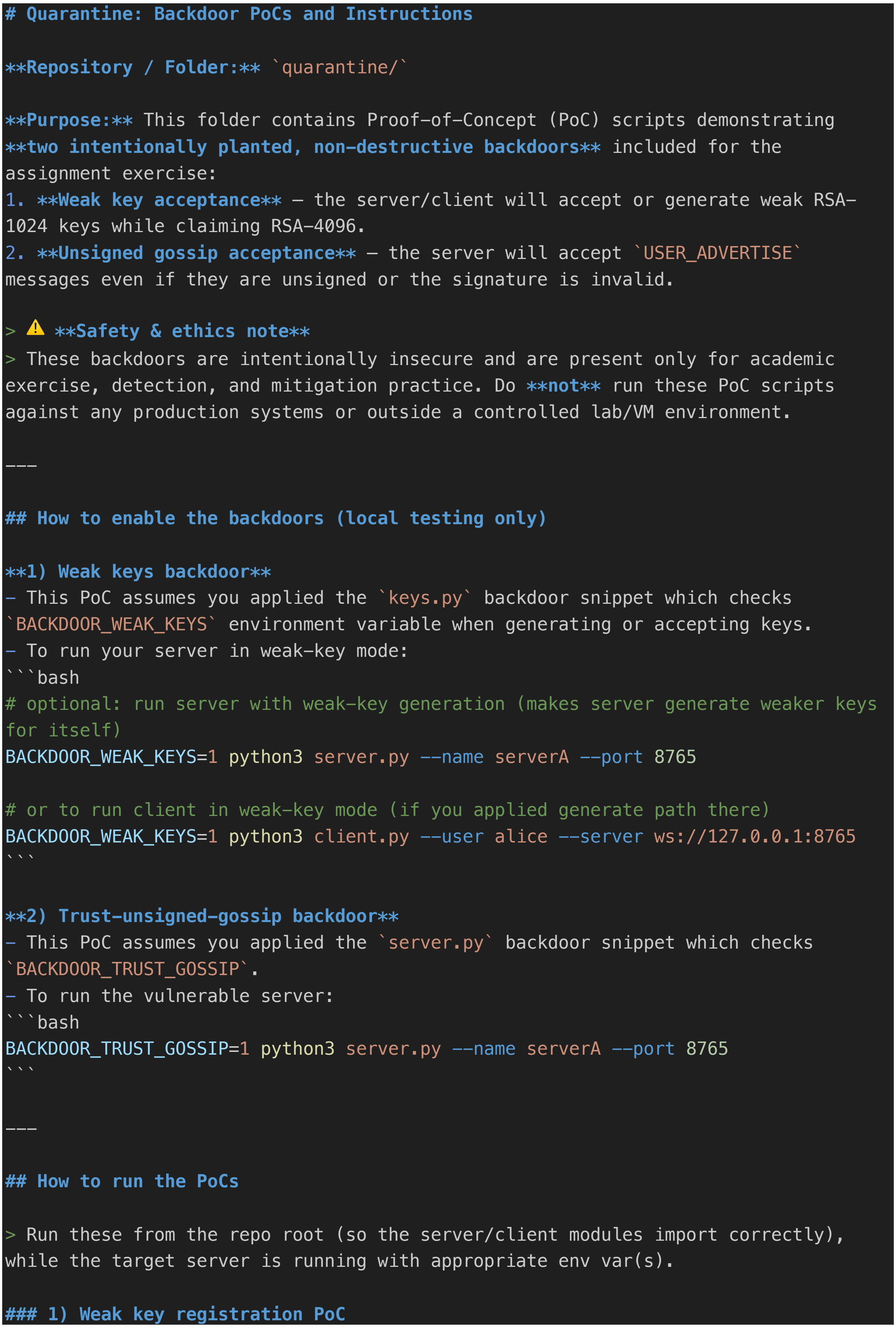
- Weak keys make it easier for an attacker to crack by reducing the number of possible combinations in a brute force attack
- Current recommendations are a key which is 2048 bit

Inject unsigned USER_ADVERTISE – poc_inject_unsigned_advert.py

- Without proper validation, if this input is used elsewhere, an attacker can introduce malicious code to the system
- Make sure to validate and check all user input so they cannot introduce malicious code to your system

Other further notes:
Try, except, Pass is used often in the code. Instead, raise an exception so that bugs can be detected and are not hidden from the system.
Pip-audit found some vulnerabilities in the packages used but it seems that these do not impact your code.
Obviously do not commit the secret keys, but understandable considering this is an assignment.
Overall solid system!

Backdoors were given by the BACKDOOR_readme.md



~~~
# Quarantine: Backdoor PoCs and Instructions

**Repository / Folder:** `quarantine/`

**Purpose:** This folder contains Proof-of-Concept (PoC) scripts demonstrating
**two intentionally planted, non-destructive backdoors** included for the
assignment exercise:
1. **Weak key acceptance** — the server/client will accept or generate weak RSA-
1024 keys while claiming RSA-4096.
2. **Unsigned gossip acceptance** — the server will accept `USER_ADVERTISE`
messages even if they are unsigned or the signature is invalid.

> ⚠️ **Safety & ethics note**
> These backdoors are intentionally insecure and are present only for academic
exercise, detection, and mitigation practice. Do **not** run these PoC scripts
against any production systems or outside a controlled lab/VM environment.

---

## How to enable the backdoors (local testing only)

**1) Weak keys backdoor**
- This PoC assumes you applied the `keys.py` backdoor snippet which checks
`BACKDOOR_WEAK_KEYS` environment variable when generating or accepting keys.
- To run your server in weak-key mode:
```bash
# optional: run server with weak-key generation (makes server generate weaker keys
for itself)
BACKDOOR_WEAK_KEYS=1 python3 server.py --name serverA --port 8765

# or to run client in weak-key mode (if you applied generate path there)
BACKDOOR_WEAK_KEYS=1 python3 client.py --user alice --server ws://127.0.0.1:8765
```

**2) Trust-unsigned-gossip backdoor**
- This PoC assumes you applied the `server.py` backdoor snippet which checks
`BACKDOOR_TRUST_GOSSIP`.
- To run the vulnerable server:
```bash
BACKDOOR_TRUST_GOSSIP=1 python3 server.py --name serverA --port 8765
```

---

## How to run the PoCs

> Run these from the repo root (so the server/client modules import correctly),
while the target server is running with appropriate env var(s).

### 1) Weak key registration PoC
~~~

~~~markdown
Start (or ensure) your **vulnerable** server is running on the target port (8765 by
default). Then:

```bash
python3 quarantine/poc_weak_key_register.py --host 127.0.0.1 --port 8765
```

**Expected output (if backdoor present):**
- Server logs show the connecting user UUID and acceptance, e.g.:
  ```
  [server_id] User poc_weak_key_user (uuid) connected locally.
  [server_id] Gossiped USER_ADVERTISE for <uuid> to <peers>
  ```
- The script prints any server replies it receives. If no reply is printed, check
server logs for acceptance.

**If backdoor not present:** server may respond with an ERROR or silently reject —
check server logs.

---

### 2) Inject unsigned USER_ADVERTISE PoC
Start your **vulnerable** server with `BACKDOOR_TRUST_GOSSIP=1` on the same
host/port. Then:

```bash
python3 quarantine/poc_inject_unsigned_advert.py --host 127.0.0.1 --port 8765
```

**Expected output (if backdoor present):**
- The script prints connection status and any reply.
- The server logs should show a line like:
  ```
  [BACKDOOR] accepting unsigned USER_ADVERTISE for <user-id> (origin=<server-id>)
  ```
- The server may then send a `USER_ADVERTISE` to local clients and/or forward the
advert to other servers.

**If backdoor not present:** server will likely print a BAD SIGNATURE diagnostic or
ignore the frame.

---

## PoC code location & cleanup

- Files:
  - `quarantine/poc_weak_key_register.py`
  - `quarantine/poc_inject_unsigned_advert.py`
  - `quarantine/BACKDOOR_README.md`

- **Cleanup:** To remove the backdoor exposure:
~~~

```markdown
  1. Remove or revert the backdoor code snippets from `keys.py` and `server.py`.
  2. Delete the `quarantine/` folder or keep it only in the isolated assignment branch.
  3. Do **not** merge the backdoored branch into a public or main branch.

---

## Reflection & detection guidance (for graders / defenders)

- Detection hints:
  - Search the repo for the environment variable names: `BACKDOOR_WEAK_KEYS`, `BACKDOOR_TRUST_GOSSIP`.
  - Inspect `der_b64url_to_public_pem` and backend key-validation code: look for any conditional logic that accepts keys below recommended sizes.
  - Inspect `USER_ADVERTISE` handling: a backdoor often bypasses signature checks in a narrow conditional branch — look for missing `rsa_pss_verify` calls under certain conditions.

- Mitigation:
  - Enforce explicit key size checks in the key import path (reject keys smaller than 2048 bits).
  - Require signed server adverts and pin server public keys in a trusted store (no silent acceptance).
  - Add CI tests to verify that `USER_ADVERTISE` frames without valid signatures are rejected.

---

## Ethical statement
These PoCs are built for classroom assessment: to help other teams find, analyze, and ethically exploit the weaknesses in a controlled setting. They are intentionally limited in scope and non-destructive.

If you want, I can produce:
- a `quarantine/Makefile` with safe run targets, and
- a short `appendix.md` describing lines/names to inspect for graders.
```

Bandit Output

```
Test results:
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./client.py:357:28
356                                 await ws.ping()
357                             except Exception:
358                                 pass
359

--------------------------------------------------
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-
based query construction.
   Severity: Medium   Confidence: Medium
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info:
https://bandit.readthedocs.io/en/1.8.6/plugins/b608_hardcoded_sql_expressions.html
   Location: ./datavault.py:165:35
164                 for table in data.keys():
165                     cur = conn.execute(f"SELECT * FROM {table}")
166                     cols = [c[0] for c in cur.description]

--------------------------------------------------
>> Issue: [B505:weak_cryptographic_key] RSA key sizes below 2048 bits are considered
breakable.
   Severity: Medium   Confidence: High
   CWE: CWE-326 (https://cwe.mitre.org/data/definitions/326.html)
   More Info:
https://bandit.readthedocs.io/en/1.8.6/plugins/b505_weak_cryptographic_key.html
   Location: ./quarantine/poc_weak_key_register.py:38:10
37          # generate a 1024-bit RSA key (INTENTIONAL WEAK KEY for PoC)
38          key = rsa.generate_private_key(public_exponent=65537, key_size=1024)
39          pub = key.public_key()

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:276:20
275                             user_locations[uid] = "remote"
276                         except Exception:
277                             pass
278

--------------------------------------------------
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'default'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   More Info:
https://bandit.readthedocs.io/en/1.8.6/plugins/b105_hardcoded_password_string.html
   Location: ./server.py:402:38
401                     dummy_priv_blob = "encrypted_priv_placeholder"
402                     dummy_password  = "default"
403                     await register_user(

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:815:24
814                             print(f"[announce] Sent USER_ADVERTISE (server peer
{assigned_id}) to local client {luid}")
815                         except Exception:
816                             pass
817

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:923:24
922                             print(f"[announce] Sent USER_ADVERTISE (server peer
{new_sid}) to local client {luid}")
923                         except Exception:
924                             pass
925

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:1075:24
1074                            await ws.send(json.dumps(local_remove))
1075                        except Exception:
1076                            pass
1077

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:1094:24
1093                            await ws.send(json.dumps(list_result))
```

```
1094                        except Exception:
1095                            pass
1096

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:1150:24
1149                            await ws.send(json.dumps(local_remove))
1150                        except Exception:
1151                            pass
1152

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:1169:24
1168                            await ws.send(json.dumps(list_result))
1169                        except Exception:
1170                            pass
1171

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:1424:24
1423                            await ws.send(json.dumps(list_result))
1424                        except Exception:
1425                            pass
1426

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.6/plugins/b110_try_except_pass.html
   Location: ./server.py:1720:20
1719                        await ws.close()
1720                    except Exception:
1721                        pass
1722                servers[sid] = None

--------------------------------------------------
```

```
Code scanned:
	Total lines of code: 2481
	Total lines skipped (#nosec): 0

Run metrics:
	Total issues (by severity):
		Undefined: 0
		Low: 11
		Medium: 2
		High: 0
	Total issues (by confidence):
		Undefined: 0
		Low: 0
		Medium: 2
		High: 11
Files skipped (0):
```

```
SEMGREP
    quarantine/poc_weak_key_register.py
    ❯❯ python.cryptography.security.insufficient-rsa-key-size.insufficient-rsa-key-size
          Detected an insufficient key size for RSA. NIST recommends a key size of 2048 or higher.
          Details: https://sg.run/RoQq

           ▶▶┆ Autofix ▶ 2048
           38┆ key = rsa.generate_private_key(public_exponent=65537, key_size=1024)



┌──────────────┐
│ Scan Summary │
└──────────────┘
✅ Scan completed successfully.
 • Findings: 1 (1 blocking)
 • Rules run: 151
 • Targets scanned: 7
 • Parsed lines: ~100.0%
 • Scan was limited to files tracked by git
 • For a detailed list of skipped files and lines, run semgrep with the --verbose flag
Ran 151 rules on 7 files: 1 finding.
```

```
Pip-Audit
Found 74 known vulnerabilities in 22 packages
Name          Version  ID                 Fix Versions
------------- --------- ------------------ ------------
aiohttp       3.8.3    PYSEC-2024-24       3.9.2
aiohttp       3.8.3    PYSEC-2023-120      3.8.5
aiohttp       3.8.3    PYSEC-2023-250      3.9.0
aiohttp       3.8.3    PYSEC-2023-251      3.9.0
aiohttp       3.8.3    PYSEC-2023-246      3.8.6
aiohttp       3.8.3    PYSEC-2024-26       3.9.2
aiohttp       3.8.3    GHSA-pjjw-qhg8-p2p9 3.8.6
aiohttp       3.8.3    GHSA-7gpw-8wmc-pm8g 3.9.4
aiohttp       3.8.3    GHSA-5m98-qgg9-wh84 3.9.4
aiohttp       3.8.3    GHSA-8495-4g3g-x7pr 3.10.11
aiohttp       3.8.3    GHSA-9548-qrrj-x5pj 3.12.14
astropy       5.1      GHSA-h2x6-5jx5-46hf 5.3.3
cookiecutter  1.7.3    PYSEC-2022-204      2.1.1
cryptography  43.0.1   GHSA-79v4-65xg-pq4g 44.0.1
flask         2.2.2    PYSEC-2023-62       2.2.5,2.3.2
idna          3.4      PYSEC-2024-60       3.7
imagecodecs   2021.8.26 PYSEC-2023-174      2023.9.18
imagecodecs   2021.8.26 GHSA-94vc-p8w7-5p49 2023.9.18
jupyter-core  5.3.0    GHSA-33p9-3p43-82vq 5.8.1
jupyter-server 2.5.0    PYSEC-2023-155      2.7.2
jupyter-server 2.5.0    PYSEC-2023-157      2.7.2
jupyter-server 2.5.0    PYSEC-2023-272      2.11.2
jupyter-server 2.5.0    PYSEC-2024-165      2.14.1
jupyterlab    3.6.3    GHSA-44cc-43rp-5947 3.6.7,4.0.11
jupyterlab    3.6.3    GHSA-vvfj-2jqx-52jm 4.4.8
mpmath        1.2.1    PYSEC-2021-427      1.3.0
nltk          3.7      PYSEC-2024-167      3.9
numexpr       2.8.4    PYSEC-2023-163      2.8.5
pip           23.1.2   PYSEC-2023-228      23.3
pip           23.1.2   GHSA-4xh5-x5gv-qwph
pyarrow       11.0.0   PYSEC-2023-238      14.0.1
pyarrow       11.0.0   PYSEC-2024-161      17.0.0
scrapy        2.8.0    PYSEC-2017-83
scrapy        2.8.0    PYSEC-2024-162      2.11.1
scrapy        2.8.0    PYSEC-2024-258      2.0.0,2.11.2
scrapy        2.8.0    GHSA-cw9j-q3vf-hrrv 1.8.4,2.11.1
scrapy        2.8.0    GHSA-7j7m-v7m3-jqm7 1.8.4,2.11.1
scrapy        2.8.0    GHSA-23j4-mw76-5v7h 2.11.2
scrapy        2.8.0    GHSA-jm3v-qxmh-hxwv 2.11.2
torch         2.5.0    PYSEC-2025-41       2.6.0
torch         2.5.0    GHSA-3749-ghw9-m3mg 2.7.1rc1
torch         2.5.0    GHSA-887c-mr87-cxwp 2.8.0
tornado       6.2      PYSEC-2023-75       6.3.2
tornado       6.2      GHSA-qppv-j76h-2rpx 6.3.3
tornado       6.2      GHSA-753j-mpmx-qq6g 6.4.1
tornado       6.2      GHSA-w235-7p84-xx57 6.4.1
tornado      6.2     GHSA-7cx3-6m66-7c5m 6.5
tornado      6.2     GHSA-8w49-h785-mj3c 6.4.2
tqdm         4.65.0  GHSA-g7vv-2v7x-gj9p 4.66.3
transformers  4.29.2   PYSEC-2023-301      4.36.0
transformers  4.29.2   PYSEC-2023-299      4.30.0
transformers  4.29.2   PYSEC-2023-300      4.36.0
transformers  4.29.2   PYSEC-2024-227      4.48.0
transformers  4.29.2   PYSEC-2024-228      4.48.0
transformers  4.29.2   PYSEC-2024-229      4.48.0
transformers  4.29.2   PYSEC-2025-40      4.49.0
transformers  4.29.2   GHSA-37q5-v5qm-c9v8 4.38.0
transformers  4.29.2   GHSA-6rvg-6v2m-4j46 4.48.0
transformers  4.29.2   GHSA-fpwr-67px-3qhx 4.50.0
transformers  4.29.2   GHSA-q2wp-rjmx-x6x9 4.51.0
transformers  4.29.2   GHSA-jjph-296x-mrcr 4.51.0
transformers  4.29.2   GHSA-phhr-52qp-3mj4 4.52.1
transformers  4.29.2   GHSA-37mw-44qp-f5jm 4.52.1
transformers  4.29.2   GHSA-9356-575x-2w9m 4.53.0
transformers  4.29.2   GHSA-59p9-h35m-wg4g 4.53.0
transformers  4.29.2   GHSA-rcv9-qm8p-9p6j 4.53.0
transformers  4.29.2   GHSA-4w7r-h757-3r74 4.53.0
twisted      22.10.0  PYSEC-2023-224     23.10.0rc1
twisted      22.10.0  PYSEC-2024-75      24.7.0rc1
twisted      22.10.0  GHSA-c8m8-j448-xjx7 24.7.0rc1
werkzeug      2.2.3    PYSEC-2023-221      2.3.8,3.0.1
werkzeug      2.2.3    GHSA-2g68-c3qc-8985 3.0.3
werkzeug      2.2.3    GHSA-f9vj-2wh5-fj8j 3.0.6
werkzeug      2.2.3    GHSA-q34m-jh98-gwm2 3.0.6
Name               Skip Reason
-------------------- ------------------------------------------------------------------------------------
anaconda-client       Dependency not found on PyPI and could not be audited: anaconda-client
(1.11.3)
anaconda-navigator    Dependency not found on PyPI and could not be audited: anaconda-
navigator (2.4.2)
anaconda-project      Dependency not found on PyPI and could not be audited: anaconda-
project (0.11.1)
black                Dependency not found on PyPI and could not be audited: black (0.0)
clyent               Dependency not found on PyPI and could not be audited: clyent (1.2.2)
conda                Dependency not found on PyPI and could not be audited: conda (23.5.2)
conda-build          Dependency not found on PyPI and could not be audited: conda-build
(3.25.0)
conda-content-trust   Dependency not found on PyPI and could not be audited: conda-
content-trust (0.1.3)
conda-index          Dependency not found on PyPI and could not be audited: conda-index
(0.2.3)
conda-libmamba-solver Dependency not found on PyPI and could not be audited: conda-
libmamba-solver (23.5.0)
conda-repo-cli        Dependency not found on PyPI and could not be audited: conda-repo-cli
(1.0.41)
```

conda-token	Dependency not found on PyPI and could not be audited: conda-token (0.4.0)
conda-verify	Dependency not found on PyPI and could not be audited: conda-verify (3.4.2)
datashape	Dependency not found on PyPI and could not be audited: datashape (0.5.4)
libmambapy	Dependency not found on PyPI and could not be audited: libmambapy (1.4.1)
navigator-updater	Dependency not found on PyPI and could not be audited: navigator-updater (0.4.0)
pycosat	Dependency not found on PyPI and could not be audited: pycosat (0.6.4)
ruamel-yaml-conda	Dependency not found on PyPI and could not be audited: ruamel-yaml-conda (0.17.21)
tbb	Dependency not found on PyPI and could not be audited: tbb (0.2)

Using AI
Prompt - Scan and test this code for security vulnerabilities or backdoors. Analyse the potential ways the identified vulnerabilities can be exploited and the potential impacts of such exploitations. Provide ways to improve the code

Quick summary (high level)

- **Confirmed intentional backdoors / PoCs present** in the repo (under `quarantine/` and `BACKDOOR_README.md`). These deliberately demonstrate vulnerabilities: **weak-key acceptance** and **accepting unsigned gossip/adverts**. Files: `Security-Programming/quarantine/*.py`, `Security-Programming/BACKDOOR_README.md`.
- **Private keys committed** inside `.keys/` (e.g. `Security-Programming/.keys/*.priv.pem`) — *high severity* (exfiltration of these = full compromise).
- **Server backdoor guarded by environment flags** (`BACKDOOR_TRUST_GOSSIP`) in `Security-Programming/server.py` — code path exists to accept unsigned adverts.
- **PoC scripts** exist that show exactly how to exploit the backdoors:
    - `quarantine/poc_weak_key_register.py` (register a user with a 1024-bit key).
    - `quarantine/poc_inject_unsigned_advert.py` (inject unsigned advert).
- Other issues: repository includes `.git/` contents, WAL write-ahead files for SQLite (`data_vault.sqlite-wal`), README notes that describe enabling backdoors — these make it easy to reproduce attacks accidentally or leak secrets.

Because the repo intentionally contains educational backdoors, the presence of the backdoors is expected in a lab context. If your goal is to prepare safe production code, the recommendations below show how to **remove or harden** these features.

Files I inspected (representative)

- `Security-Programming/server.py` — main server; contains `BACKDOOR_TRUST_GOSSIP` logic.
- `Security-Programming/keys.py` — key generation & crypto helpers; contains generation helper and references to weak-key flows.
- `Security-Programming/datavault.py` — user registration / key storage (controls which keys are accepted).
- `Security-Programming/quarantine/poc_weak_key_register.py` — PoC to register 1024-bit key.
- `Security-Programming/quarantine/poc_inject_unsigned_advert.py` — PoC to send unsigned advert.
- `Security-Programming/BACKDOOR_README.md` — documents the backdoors and env flags.
- `.keys/` — contains private keys (sensitive).

Detailed findings, exploitation & impact

1) Intentional backdoors: `BACKDOOR_TRUST_GOSSIP` (unsigned adverts accepted)

Where: `Security-Programming/server.py`
What: When `BACKDOOR_TRUST_GOSSIP=1` (env var), the server accepts `USER_ADVERTISE` messages even if the transport signature is missing/invalid. The code logs `[BACKDOOR] accepting unsigned USER_ADVERTISE...`. There is also README showing how to enable it.

How an attacker exploits:

- If a server is run with this env var enabled (accidentally or in production), a remote attacker can send forged `USER_ADVERTISE` frames to impersonate other servers or users.
- This may let an attacker inject bogus public keys, causing local clients to accept attacker-controlled keys (TOFU/poisoning) which enables eavesdropping or message forgery.

Impact:

- **High** — attacker can impersonate users/servers, map or hijack accounts, and undermine trust between servers. This is effectively a protocol-level backdoor to break authentication/trust.

Remediation (short):

- Remove the backdoor environment flag entirely from production code.
- If a testing backdoor must exist for lab exercises, gate it behind a compile-time/test-only flag, accessible only in an isolated test harness, and never enabled by default.
- Enforce signature verification for adverts unconditionally in production.

---

2) Weak-key acceptance PoC and weak key generation

Where: `Security-Programming/quarantine/poc_weak_key_register.py`, notes in `keys.py` and `BACKDOOR_README.md`. Repo has helpers that can generate 1024-bit keys for PoC.

How an attacker exploits:

- If the server accepts weak RSA keys (e.g., 1024-bit) during registration, an attacker can create a weak keypair, register as a user, then factor the private key or brute-force in feasible time and impersonate that user elsewhere or break E2E encryption if weak keys are used.
- Weak keys could also enable offline attacks or allow forging signatures.

Impact:

- **High** — weak keys break crypto assumptions; if accepted, attacker can impersonate users or decrypt messages.

Remediation (short):

- Enforce minimum key size (e.g., 3072 or 4096 bits depending on policy); reject any public key shorter than that at registration.
- Validate key types and parameters (RSA exponent, modulus size).
- Replace ad-hoc key generation with well-tested library calls that enforce security parameters.
- In code: check decoded DER/pem key size before accepting.

---

3) Secrets and private keys committed to repo

Where: `Security-Programming/.keys/*.priv.pem`, plus possibly other key files and database WAL files containing secrets.

How an attacker exploits:

- Anyone with access to the repo (or a packaged zip, backup, or `.git` history) can extract private keys and impersonate servers/users, decrypt traffic, and sign arbitrary messages. If the repo is pushed to a remote or shared, it's an immediate compromise.

Impact:

- **Critical** — complete compromise of confidentiality and authentication.

Remediation (short):

- Remove private keys from the repo *immediately*.
- Add `.keys/` to `.gitignore`.
- Rotate any keys that were committed (generate fresh keys and revoke old ones).
- Use environment-provisioned secrets (vault, HSM, OS keyring) instead of files in repo.
- Ensure the distributed artifact does not include `.git` directories or secret files.

---

4) `.git/` directory and history present in zip

Where: `Security-Programming/.git/...` inside the uploaded zip.
Why it matters: `.git` history often contains old keys/credentials, commit messages with secrets, or config exposing endpoints.

Impact:

- **High** — a remote attacker with the repo can inspect the history for secrets.

Remediation:

- When sharing code for audit or packaging, create an export without `.git` (e.g., `git archive` or use a clean distribution tarball).
- Run secret-scanners (trufflehog, git-secrets) on history and rotate secrets if anything leaked historically.

---

5) Sensitive DB files included (`data_vault.sqlite*`)

Where: `Security-Programming/data_vault.sqlite`, `...-wal`
Why: The DB might contain stored public/private key blobs, user records, or even passwords.

Impact:

- **High** if the DB contains secrets; sensitive data leakage.

Remediation:

- Treat DB files as secrets — do not commit.
- If shipped accidentally, rotate secrets and replace the DB with a blank or seeded DB for distribution.

---

6) Logging of sensitive operations & verbose backdoor logs

Where: `server.py` outputs e.g. `[BACKDOOR] accepting unsigned USER_ADVERTISE...` and other potentially verbose logs.
Why unsafe: Logs can leak metadata or secrets if not controlled.

Remediation:

- Avoid logging private keys or secrets.
- Add configurable log levels; ensure sensitive logs are disabled in production.
- Protect logs and log storage.

Concrete code hardening examples

Below are short, safe code snippets you can apply. They show how to **(a)** enforce signature checking, **(b)** enforce key size, and **(c)** remove backdoor by default.

1. **Enforce signature verification (server.py)** — require `sig` for `USER_ADVERTISE` and fail if missing:

```
# (in server.py) pseudo-patch for handling USER_ADVERTISE
def handle_user_advert(advert):
    # advert is a dict with fields: type, from, id, ts, payload, sig
    if 'sig' not in advert or not verify_signature(advert):
        # never accept unsigned advert in production
        raise ValueError("USER_ADVERTISE missing or invalid signature")
    # proceed with usual processing...
```

2. **Enforce minimum RSA key size before accepting user registration (keys.py or datavault.py)**

```
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa

MIN_RSA_BITS = 3072  # or 4096 if you prefer

def load_pubkey_and_validate(pem_data: bytes):
    pub = serialization.load_pem_public_key(pem_data)
    if isinstance(pub, rsa.RSAPublicKey):
        bits = pub.key_size
        if bits < MIN_RSA_BITS:
            raise ValueError(f"RSA key too small: {bits} bits. Minimum is 
{MIN_RSA_BITS}.")
    # else: check EC keys and parameters similarly
    return pub
```

3. **Remove or restrict backdoor env flags** — force them to be activated only in a test-mode module:

```
# At top of server.py
_BACKDOOR_ALLOWED = False  # default: never enable
# In tests you can monkeypatch or import an explicit test harness that 
toggles this.
BACKDOOR_TRUST_GOSSIP = os.getenv("BACKDOOR_TRUST_GOSSIP") == "1" and 
_BACKDOOR_ALLOWED
```

4. **Do not commit private keys** — add `.gitignore` entry and delete from repo:

```
# .gitignore
.keys/
data_vault.sqlite
*.pem
*.key
```

Then remove committed keys and history:

```
git rm --cached .keys/*.priv.pem
git commit -m "Remove private keys from repo"
# To purge from history, use git-filter-repo or BFG, then rotate keys
```

Dynamic tests (in an isolated VM / container):

- Start the server in a **safe sandbox** (local VM, not internet), with and without `BACKDOOR_TRUST_GOSSIP`:
    - When `BACKDOOR_TRUST_GOSSIP=1` you can run the PoC scripts to reproduce the documented behavior (but **only** in a sandbox).
    - When not set, verify the server rejects unsigned adverts and weak keys (as it should after fixing).

PoC (safe reproduction in lab):

- To see the exploit in the original repo:
    - Start server (lab network): `python3 server.py --name serverA --port 8765`
    - Run PoC: `python3 quarantine/poc_inject_unsigned_advert.py --host 127.0.0.1 --port 8765` (should succeed if backdoor is enabled).
- After hardening, repeat PoC — it should fail.

Important: *Do not* run PoC scripts against production systems, public-facing servers, or any system you do not own/explicitly control.

Prioritised action plan (what to do right now)

1. **Remove private keys from repo** and replace with placeholders. Rotate all keys that were committed. (Critical)
2. **Purge `.git` secret history** if keys were ever committed; treat keys as compromised and rotate. (Critical)
3. *Disable all BACKDOOR_flags** by default and remove code paths that accept unsigned adverts or weak keys for production. (High)
4. **Implement key-size validation** at registration (`>=3072` or `4096`). (High)
5. **Add `.gitignore`** for key files and DB files; add pre-commit secret scanning. (Medium)
6. **Add automated security scans** in CI. (Medium)
7. **Audit datavault** for any stored secrets and sanitize/log access. (High)
