Security Testing and Vulnerability Analysis of `secure_version`

Introduction:

The security testing suite for the `secure_version` project was developed to ensure that the codebase maintains a strong security posture. It combines static analysis with targeted runtime checks to detect common vulnerabilities, including weak RSA keys, unsanitized file paths, unsafe SQL operations, missing signature verification, and potential high-severity issues flagged by tools such as Bandit. These tests serve as a safeguard to prevent the accidental introduction of insecure patterns and enforce adherence to security best practices. Additionally, a proof-of-concept interoperability test was conducted with Group 43's SOCP system to validate cross-network compatibility and confirm robust handling of non-conforming peers.

Integration Testing

The `test_integration.py` suite validates the end-to-end behavior of the secure chat system. It confirms that encryption and decryption function correctly, keys persist safely across sessions, and concurrent key generation remains consistent. Real server and client processes are launched to verify encrypted on-wire communication, message delivery, and user presence updates when clients connect or disconnect. Replay detection ensures duplicate messages are rejected, while introducer configuration parsing confirms the bootstrap process operates correctly. Overall, this suite provides comprehensive systems-level verification of secure message exchange, reliability, and protocol integrity.

Server Module Testing

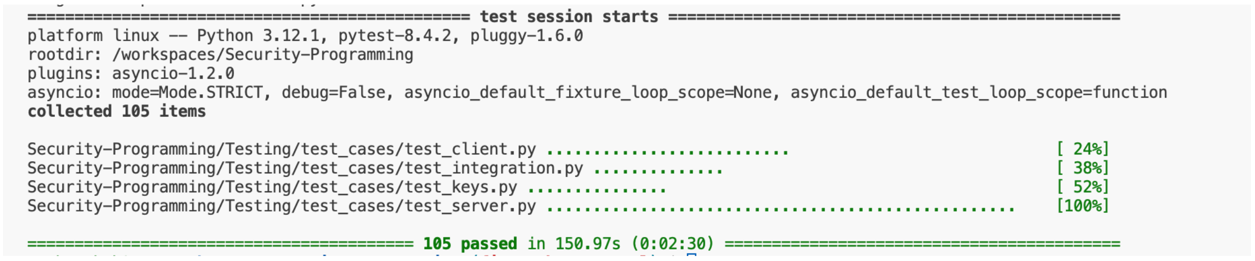
The `test_server.py` suite rigorously validates the server module's security, reliability, and protocol compliance. It tests cryptographic signing and verification of JSON envelopes, introducer loading and validation, timestamp freshness, and key conversion integrity. The tests simulate federation scenarios, verifying presence synchronization, broadcast handling, file transfer limits, and message routing through asynchronous websocket mocks. Error resilience is exercised by injecting malformed inputs, tampered payloads, and missing signatures to confirm proper rejection and exception handling. These tests ensure the server behaves securely and predictably across both local and federated operations.

Client Module Testing

The `test_client.py` suite serves as a comprehensive validation for the Secure Overlay Chat Protocol's client implementation. It tests key cryptographic and communication mechanisms, including RSA-PSS signature verification, message canonicalization, Base64 decoding, and replay attack prevention. The suite also validates proper key persistence, secure file transfer handling, and end-to-end message confidentiality. By simulating real client–server exchanges and verifying expected cryptographic behavior, these tests confirm that the client code adheres to the protocol's integrity and authenticity requirements. The controlled use of subprocesses and asynchronous networking ensures safety within the test context.

Cryptography Utilities Testing

The `test_keys.py` suite provides thorough verification of all cryptographic utilities underpinning secure server and client operations. It tests RSA-4096 key generation, OAEP encryption/decryption, and RSASSA-PSS signing/verification to ensure cryptographic correctness and resistance to tampering. Base64URL encoding, DER/PEM conversions, and UUID persistence for users and servers are also validated, confirming stable identity and storage behavior. Negative and error-handling tests for malformed keys, permission issues, and invalid inputs guarantee predictable failure responses. The suite also tests introducer key and YAML generation within controlled directories. Collectively, these tests ensure the cryptographic layer is both correct and resilient, with Figure 1 showing that all integration, server, client, and cryptography utility tests passed successfully.



```
============================================== test session starts ===============================================
platform linux -- Python 3.12.1, pytest-8.4.2, pluggy-1.6.0
rootdir: /workspaces/Security-Programming
plugins: asyncio-1.2.0
asyncio: mode=Mode.STRICT, debug=False, asyncio_default_fixture_loop_scope=None, asyncio_default_test_loop_scope=function
collected 105 items

Security-Programming/Testing/test_cases/test_client.py .........................                           [ 24%]
Security-Programming/Testing/test_cases/test_integration.py ..............                                 [ 38%]
Security-Programming/Testing/test_cases/test_keys.py ...............                                       [ 52%]
Security-Programming/Testing/test_cases/test_server.py ...................................................    [100%]

======================================== 105 passed in 150.97s (0:02:30) =========================================
```

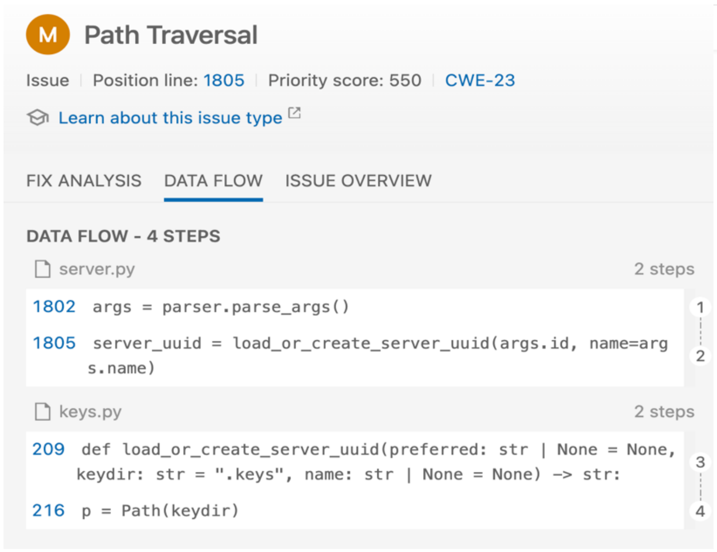
Figure 1: Test session result

Security Hardening Test

The `test_security_hardening.py` suite validates that secure_version maintains a robust security posture. It ensures no active `BACKDOOR_*` flags exist, verifies signatures for `USER_ADVERTISE` messages, enforces strong RSA key sizes (rejecting <2048-bit), and detects unsafe key generation in the code. Database operations are tested for SQL injection resilience, correct literal storage of inputs, and proper use of parameterized queries. Input handling is also checked for very long display names, while Bandit scans confirm the absence of high-severity issues. Collectively, these static and dynamic checks confirm that the code enforces strong cryptographic practices, input validation, and secure coding standards.

Snyk Path Traversal Analysis

During `Snyk` analysis, a Path Traversal vulnerability (CWE-23) was identified at line 1805, as illustrated in Figures 1 and 2. This vulnerability could have allowed untrusted input to manipulate file paths (e.g., `../../`) and access files outside the intended `.keys` directory. The issue was remediated by sanitizing all user-controlled filenames: removing path separators, rejecting `..`, enforcing a strict `[A-Za-z0-9._-]+` whitelist, and resolving paths against a safe base directory. This patch eliminates traversal and Zip-Slip risks while maintaining normal client-server functionality.



M Path Traversal

Issue | Position line: 1805 | Priority score: 550 | CWE-23

Learn about this issue type

DATA FLOW - 4 STEPS

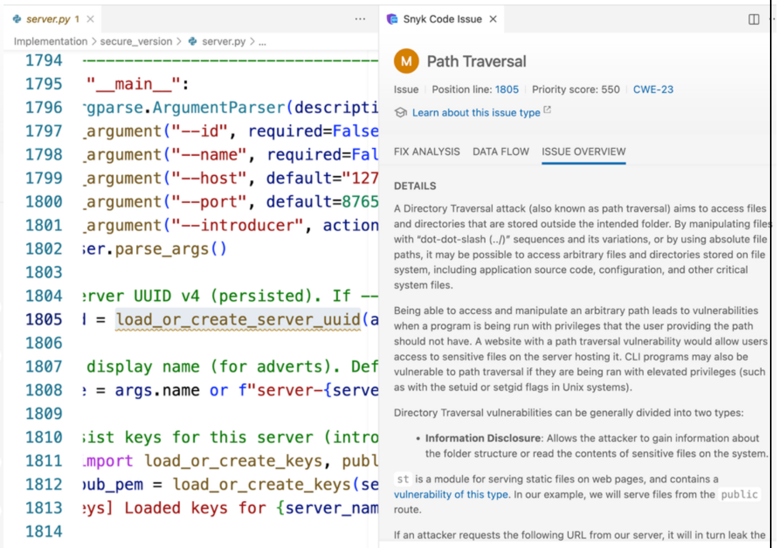
server.py — 2 steps

```
1802  args = parser.parse_args()
1805  server_uuid = load_or_create_server_uuid(args.id, name=args.name)
```

keys.py — 2 steps

```
209  def load_or_create_server_uuid(preferred: str | None = None,
     keydir: str = ".keys", name: str | None = None) -> str:
216  p = Path(keydir)
```

Figure 2: Path Traversal data flow



```
-----------------------------------
"__main__":
gparse.ArgumentParser(descripti
_argument("--id", required=False
_argument("--name", required=Fal
_argument("--host", default="127
_argument("--port", default=8765
_argument("--introducer", action
er.parse_args()

rver UUID v4 (persisted). If --
 = load_or_create_server_uuid(a

 display name (for adverts). Def
 = args.name or f"server-{serve

ist keys for this server (intro
mport load_or_create_keys, publ
ub_pem = load_or_create_keys(se
ys] Loaded keys for {server_nam
```

Snyk Code Issue

M Path Traversal

Issue | Position line: 1805 | Priority score: 550 | CWE-23

Learn about this issue type

DETAILS

A Directory Traversal attack (also known as path traversal) aims to access files and directories that are stored outside the intended folder. By manipulating files with "dot-dot-slash (../)" sequences and its variations, or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system, including application source code, configuration, and other critical system files.

Being able to access and manipulate an arbitrary path leads to vulnerabilities when a program is being run with privileges that the user providing the path should not have. A website with a path traversal vulnerability would allow users access to sensitive files on the server hosting it. CLI programs may also be vulnerable to path traversal if they are being ran with elevated privileges (such as with the setuid or setgid flags in Unix systems).

Directory Traversal vulnerabilities can be generally divided into two types:

- **Information Disclosure**: Allows the attacker to gain information about the folder structure or read the contents of sensitive files on the system.

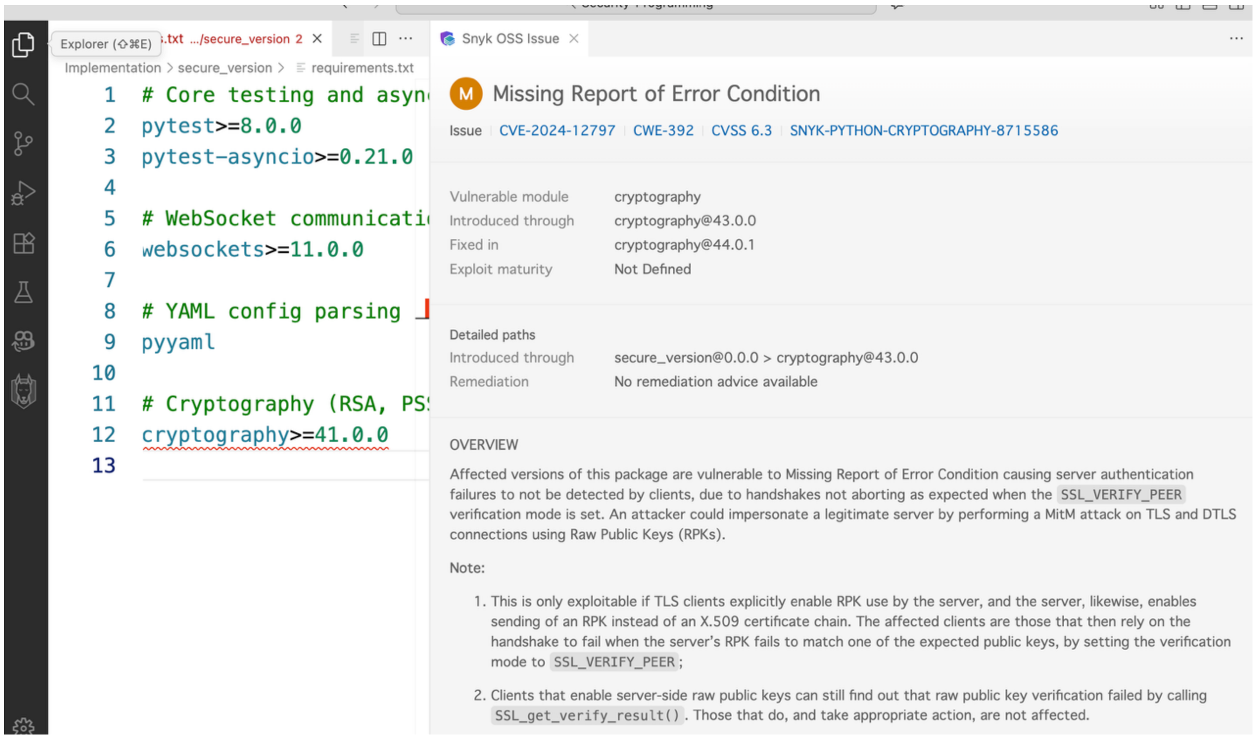
`st` is a module for serving static files on web pages, and contains a vulnerability of this type. In our example, we will serve files from the `public` route.

If an attacker requests the following URL from our server, it will in turn leak the

Figure 2: Path Traversal flag

Snyk Dependency Analysis

Snyk also flagged a Missing Report of Error Condition vulnerability (CVE-2024-12797, CWE-392) in the dependency `cryptography@43.0.0`, introduced through `secure_version@0.0.0`. Rated CVSS 6.3, this issue could cause TLS clients using Raw Public Keys (RPKs) to overlook authentication failures, potentially enabling a man-in-the-middle attack. Notably, this vulnerability was inadvertently introduced when an AI tool (Copilot) generated the project's `requirements.txt`, highlighting a limitation of current AI-assisted development in selecting secure dependency versions. As shown in Figure 3, Snyk flagged the affected module and provided details. The vulnerability was resolved by upgrading to `cryptography==46.0.0`, ensuring proper reporting of authentication failures and maintaining compatibility with existing encryption mechanisms. This fix secures TLS and DTLS communications across all environments.


Explorer (⇧⌘E)
.txt .../secure_version 2
Snyk OSS Issue
Implementation > secure_version > requirements.txt

```
# Core testing and asyn
pytest>=8.0.0
pytest-asyncio>=0.21.0

# WebSocket communicati
websockets>=11.0.0

# YAML config parsing
pyyaml

# Cryptography (RSA, PS
cryptography>=41.0.0

```

Missing Report of Error Condition

Issue | CVE-2024-12797 | CWE-392 | CVSS 6.3 | SNYK-PYTHON-CRYPTOGRAPHY-8715586

Vulnerable module	cryptography
Introduced through	cryptography@43.0.0
Fixed in	cryptography@44.0.1
Exploit maturity	Not Defined

Detailed paths

Introduced through	secure_version@0.0.0 > cryptography@43.0.0
Remediation	No remediation advice available

OVERVIEW

Affected versions of this package are vulnerable to Missing Report of Error Condition causing server authentication failures to not be detected by clients, due to handshakes not aborting as expected when the `SSL_VERIFY_PEER` verification mode is set. An attacker could impersonate a legitimate server by performing a MitM attack on TLS and DTLS connections using Raw Public Keys (RPKs).

Note:

1. This is only exploitable if TLS clients explicitly enable RPK use by the server, and the server, likewise, enables sending of an RPK instead of an X.509 certificate chain. The affected clients are those that then rely on the handshake to fail when the server's RPK fails to match one of the expected public keys, by setting the verification mode to `SSL_VERIFY_PEER`;
2. Clients that enable server-side raw public keys can still find out that raw public key verification failed by calling `SSL_get_verify_result()`. Those that do, and take appropriate action, are not affected.

Figure 3: Snyk report

Interoperability Testing

Our secure_version implementation was tested using Group 43 SOCP system to test cross-network compatibility. With our introducer and server as the shared communication base we tried to bridge their server and clients. Although both systems were based on the same cryptographic foundations, namely RSA-4096, RSA-OAEP, and RSASSA-PSS, message structures, and the handshake were also different. A single-server introducer in Group 43 only offers partial multi-server federation and their client uses an HMAC-based authentication flow unlike our signature-based authentication. Consequently, connection requests gave back an Unhandled message type and authentication mismatches. In spite of these variations, the test reaffirmed that secure_version was properly implementing message validation, timestamp freshness and signature integrity and the non-compliant peers were handled safely without compromising the SOCP v1.3 protocol vigour.

Conclusion:

All tests in the suite passed, confirming that `secure_version` enforces proper key sizes, input sanitization, signature verification, and SQL parameterization. Bandit reported no high-severity issues. The code demonstrates robust defensive practices, and the applied patches address identified vulnerabilities. Continuous monitoring, secure dependency management, and integration of automated security checks are recommended to maintain this high level of security over time. The interoperability PoC further demonstrated that secure_version preserves protocol integrity even when interfacing with systems using divergent handshake and authentication models.