Dynamic Security Analysis Report — Implementation 06

1. Overview

This dynamic analysis was conducted inside a sandboxed Docker container under full isolation:
- Network disabled (--network=none)
- Non-root user: auditor
- Restricted privileges: (--cap-drop=ALL, --security-opt=no-new-privileges)
- Temporary filesystem: (--tmpfs /tmp:rw)

The target system, labeled SOCP, is a Python 3.11 chat application with multiple submodules and command options (e.g., gen, server, client). The analysis aimed to identify any malicious behavior, unauthorized network communication, or system manipulation during runtime.

Observation duration: approximately 60 seconds.

Primary tools:
- strace — to capture all system calls involving network, file, and process activity
- inotifywait — to monitor file modifications in real time
- ps and ss — to log process and socket information.

2. Runtime Behavior Summary

2.1 Process Activity
The program src/main.py was executed but immediately reported:
usage: main.py [-h] {gen,server,client} ...
main.py: error: the following arguments are required: cmd
This indicates the script uses Python's argparse and requires a cmd argument (such as server or client) to start properly. Because no argument was provided, the application terminated before initializing network components or threads.

Result: No suspicious subprocesses, shell invocations, or privilege escalations were detected. The system stayed idle after command-line argument validation failed.

2.2 File Access Behavior
Analysis of strace_log.14 revealed:
- Normal Python module imports (os, sys, argparse, logging)
- Access to internal project directories like /home/auditor/app/src
- No read/write activity involving sensitive files such as /etc/passwd, /root/, or user credentials
- No evidence of external file writing beyond the container workspace

Result: All file access operations remained within legitimate project directories; no unauthorized access attempts were found.

3. Network Activity Analysis

From ss.txt:
No TCP or UDP sockets were created during the observation window. The absence of connect() and bind() system calls in the strace log further confirms that no networking activity occurred.

Result: The program did not attempt to establish or listen on any network sockets, showing no backdoor or unauthorized connections.

4. System Call and API Observations

Inspection of traced syscalls in strace_log.14 revealed:
- Only standard I/O operations (open/read/write for Python libraries)
- No process control functions (execve, fork, clone)
- No system modifications (chmod, unlink, rename)
- No signs of privilege alteration (setuid, setgid)

Result: System call usage was minimal and strictly limited to interpreter startup, argument parsing, and error reporting.

5. Security Assessment

Category	Observation	Status	Remarks
Process spawning	None	✅ Safe	No subprocesses or shell execution
Network activity	None	✅ Safe	No sockets, no connections
File access	Internal only	✅ Safe	Restricted to /home/auditor/app/src
Privilege use	None	✅ Safe	No escalation or modification
Code execution	Requires argument	⚠️ Info	Program exited before main logic
Backdoor patterns	None observed	✅ Safe	No hidden processes or ports

6. Recommendations

1. Provide valid startup parameters (e.g., python3 src/main.py server) to observe full runtime behavior in future tests.
2. Continue using sandboxed Docker environments for future analyses to isolate potential malicious traffic.
3. Consider static scanning (bandit, pylint --enable=security) for additional code-level checks.
4. Maintain detailed runtime logging for reproducibility in later reviews.

7. Conclusion

The SOCP Implementation 06 sample exhibited no evidence of malicious or suspicious activity under dynamic analysis. The process executed argument validation logic only, without initiating any I/O or network operations.

Result: PASS — No security anomalies detected. The program behavior aligns with normal Python application initialization and safe argument handling routines.