Dynamic Security Analysis Report — Implementation 05

1. Overview

This dynamic analysis was conducted in a sandboxed Docker container under strict isolation:
- Network disabled (--network=none)
- Non-root user (auditor)
- Minimal privileges (--cap-drop=ALL, --security-opt=no-new-privileges)

The target system is the Team-33 CodingGeeks Chat Application, written in Python 3.11. The goal of this analysis was to detect potential backdoors, unauthorized network communications, or unexpected system behavior during runtime.

Observation duration: approximately 60 seconds.

Tools used:
- strace: monitored all file, process, and network syscalls
- inotifywait: tracked file creation and modification
- ps, ss: collected process and socket information.

2. Runtime Behavior Summary

2.1 Process Behavior
During runtime, the primary script server.py attempted to start but failed immediately due to a Python syntax issue:
IndentationError: expected an indented block after function definition on line 10.
This indicates a malformed indentation within the source code (likely intentional as part of the assignment's testing scenarios). Despite the crash, no suspicious subprocesses or shell invocations (/bin/sh, bash, etc.) were observed.

Result: The application remained self-contained and did not attempt to spawn or control any external processes.

2.2 File Access Behavior
From the strace_log.14:
- The application accessed local project files under /home/auditor/app/, specifically server.py, standard libraries, and temporary Python cache files.
- No file access attempts were made to sensitive system directories such as /etc/passwd, /root/, or /home/auditor/.ssh/.
- No file creation or write activity outside the working directory (e.g., /tmp, /var) was detected.

Result: File operations are safe, localized, and limited to legitimate paths.

3. Network Activity Analysis

From ss.txt:
No active or listening sockets were reported. Additionally, strace_log.14 revealed no connect() or bind() system calls, confirming that no network activity was initiated.

Result: No external communications or port exposure detected.

4. System Call and API Observations

System call tracing indicates:
- Normal Python interpreter initialization (open, read, write, fstat, futex).
- No execution-related calls like execve, fork, or clone.
- No file or network descriptors persisted beyond interpreter startup.
- No suspicious attempts at privilege modification (setuid, setgid).

Result: System calls correspond only to Python's standard library initialization and error reporting.

5. Security Assessment

Category	Observation	Status	Remarks
Process Spawning	None	✅ Safe	No subprocesses or shell activity
Network Connections	None	✅ Safe	No sockets or external communications
File Access	Local only	✅ Safe	No sensitive paths accessed
Privilege Use	None	✅ Safe	No privilege escalation
Code Integrity	IndentationError crash	⚠️ Info	Code halted early, not a runtime security issue
Backdoor Indicators	None observed	✅ Safe	No dynamic links, sockets, or file drops

6. Recommendations

1. Fix the syntax issue (IndentationError) in server.py to allow full runtime evaluation.
2. Enable full program run after syntax correction to capture message routing or chat logic behaviors.
3. Maintain sandbox isolation for future dynamic tests to prevent unintended network

exposure.
4. Add structured error logging to capture runtime failures for debugging and verification.

7. Conclusion

The Team-33 CodingGeeks (impl05) implementation exhibited no signs of malicious intent or hidden functionality during dynamic analysis. The application crashed immediately due to a code indentation error, but no unsafe behavior was detected before termination.

Result: PASS — No security anomalies or backdoors found. The system demonstrates low runtime risk, consistent with a benign Python application under sandbox conditions.