Secure Coding Lab – Detailed Report

Programs 6, 9, and 10

Program 6 — Token Generator

OVERVIEW
The original implementation of Program 6 used the Python 'random' module and explicitly seeded the pseudo-random number generator with the current system timestamp using random.seed(time.time()). This creates weak, predictable randomness. Since 'random' is not designed for security purposes, any value generated from it can be predicted if the seed is known or approximated.

WHY THIS IS VULNERABLE
The time.time() function returns the current system time in seconds. Attackers can typically approximate the moment a token is generated within a small margin. Since random.seed() fully determines the output sequence, an attacker can locally reproduce the exact same tokens by initializing their PRNG with a similar timestamp.

Possible Attack Scenario:
• Attacker estimates the timestamp of token generation.
• Recreates the random number sequence locally.
• Predicts the 6-digit token, defeating authentication or verification.

SECURE FIX EXPLAINED
The 'secrets' module in Python is designed for generating cryptographically strong randomness sourced from the operating system. Unlike 'random', it is resistant to prediction because it does not rely on deterministic seeding.

Updated Approach:
• Use secrets.randbelow() to generate an unpredictable 6■digit token.
• Does not rely on seed values.
• Protects against brute■force prediction.

Program 9 — File Upload Handler

OVERVIEW
Program 9 accepts a file upload but applies insufficient validation. The main security issue is the lack of file extension validation, allowing users to upload arbitrary files. Even though Werkzeug's secure_filename() sanitizes filenames to remove path traversal sequences, it does not restrict dangerous file types.

WHY THIS IS VULNERABLE
Without filtering file types, an attacker may upload files such as:
• Executable files (.exe)
• Web shell scripts (.php, .jsp, .asp)
• Malicious JavaScript (.js)
• Overly large files for disk-filling attacks

If the upload directory is exposed via a web server, executing these files becomes possible and may lead to remote code execution, malware distribution, or defacing the system.

Possible Attack Scenario:
• Attacker uploads shell.php.
• Server responds by storing it in the uploads directory.
• If directory is accessible to HTTP requests, shell.php can be executed.
• Complete system compromise through web shell commands.

SECURE FIX EXPLAINED
The secure implementation introduces:
• A strict allowlist containing only safe file extensions.
• Continued use of secure_filename() to sanitize names.
• Controlled saving of files in a dedicated directory.

This blocks scripts, executables, and non-approved file formats, ensuring only safe uploads are accepted.

Program 10 — File Processor

OVERVIEW
Program 10 suffers from a classic TOCTOU (Time-of-Check Time-of-Use) vulnerability. The code checks if the file exists with os.path.exists() and then opens it afterward. Between these two operations, an attacker can modify or replace the file.

WHY THIS IS VULNERABLE
File race conditions occur when the security of a file-based operation depends on a check performed separately from the operation itself.

Possible Attack Scenario:
• Attacker monitors when the program checks existence of important.txt.
• Immediately swaps the file with a symbolic link to a sensitive file such as /etc/passwd.
• The program then opens the new file and reads it.
• Sensitive system content is disclosed.

In systems where symbolic links or file replacements are allowed between operations, this can lead to privilege escalation or unauthorized access.

SECURE FIX EXPLAINED
The corrected version uses atomic file opening with os.open() and the O_NOFOLLOW flag. This achieves:
• A single system call that prevents race conditions.
• Protection against symbolic link attacks by forbidding opening symlinks.
• Proper handling of dangerous conditions via exceptions.

By merging the check and open into one atomic operation, TOCTOU becomes impossible to exploit.