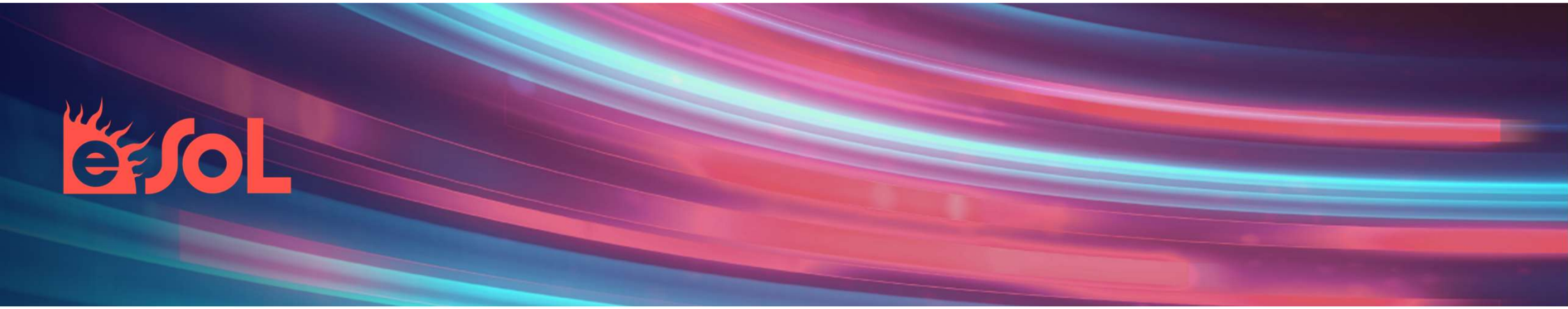
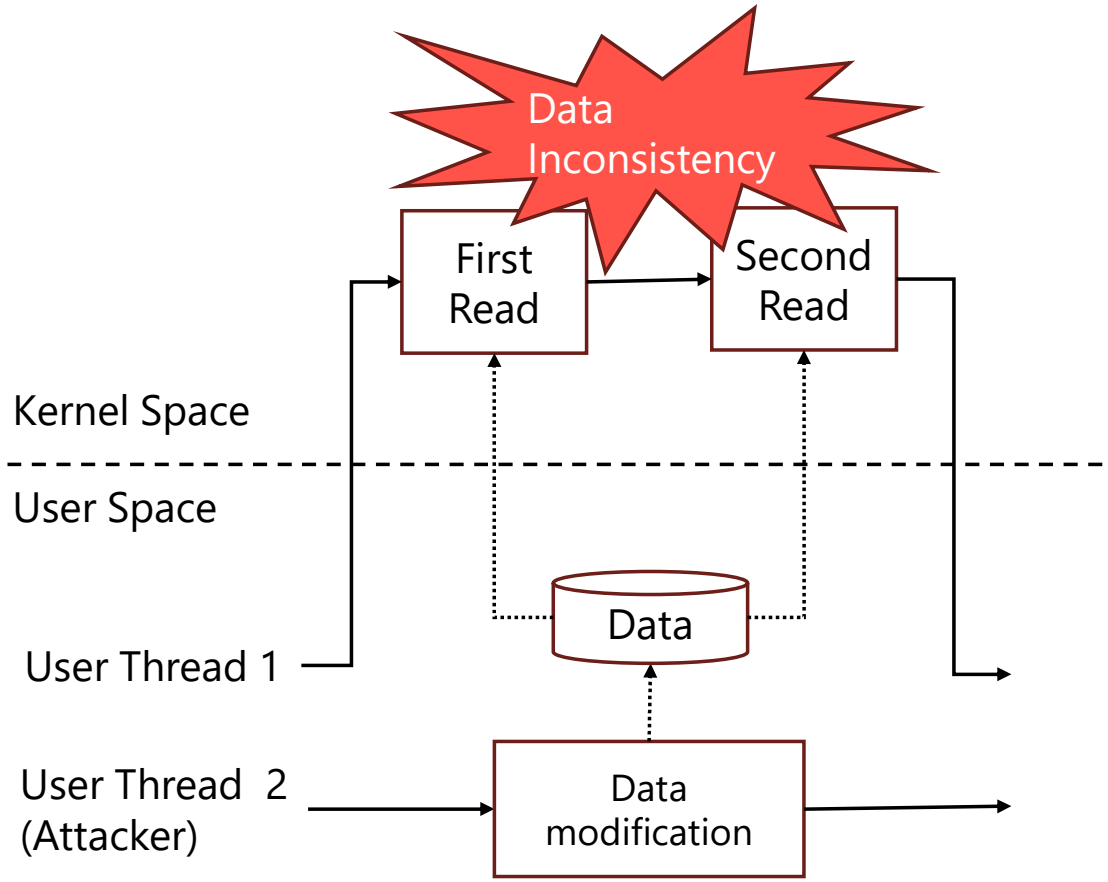


Toward a Practical Double-Fetch Checker for Clang Static Analyzer: Early Results and Future Directions for OS Security

Fumiya Shigemitsu
f-shigemitsu@esol.co.jp

Why Clang Static Analyzer, Why Double-Fetch?

- Great commercial tools ≠ always care about what we need
  → Some checks we want are missing, unsupported, or unsuitable for our use-case
- Clang/LLVM = open, hackable, and already a part of our tool-chain
  → Clang Static Analyzer allows us to write custom rules to meet our needs
- We develop Real-Time Operating Systems (RTOS) – a kind of OS
  → Requires specialized validation
- Pick one serious OS bug → Double-Fetch
  → Serious, OS-specific, and ideal for learning custom checker design



Checker Overview

Memory Access Provenance

- Record where each load comes from.
- Tag each access with a unique ID.

Double Fetch Detection

- Detect when data with mismatched provenance tags is consumed.

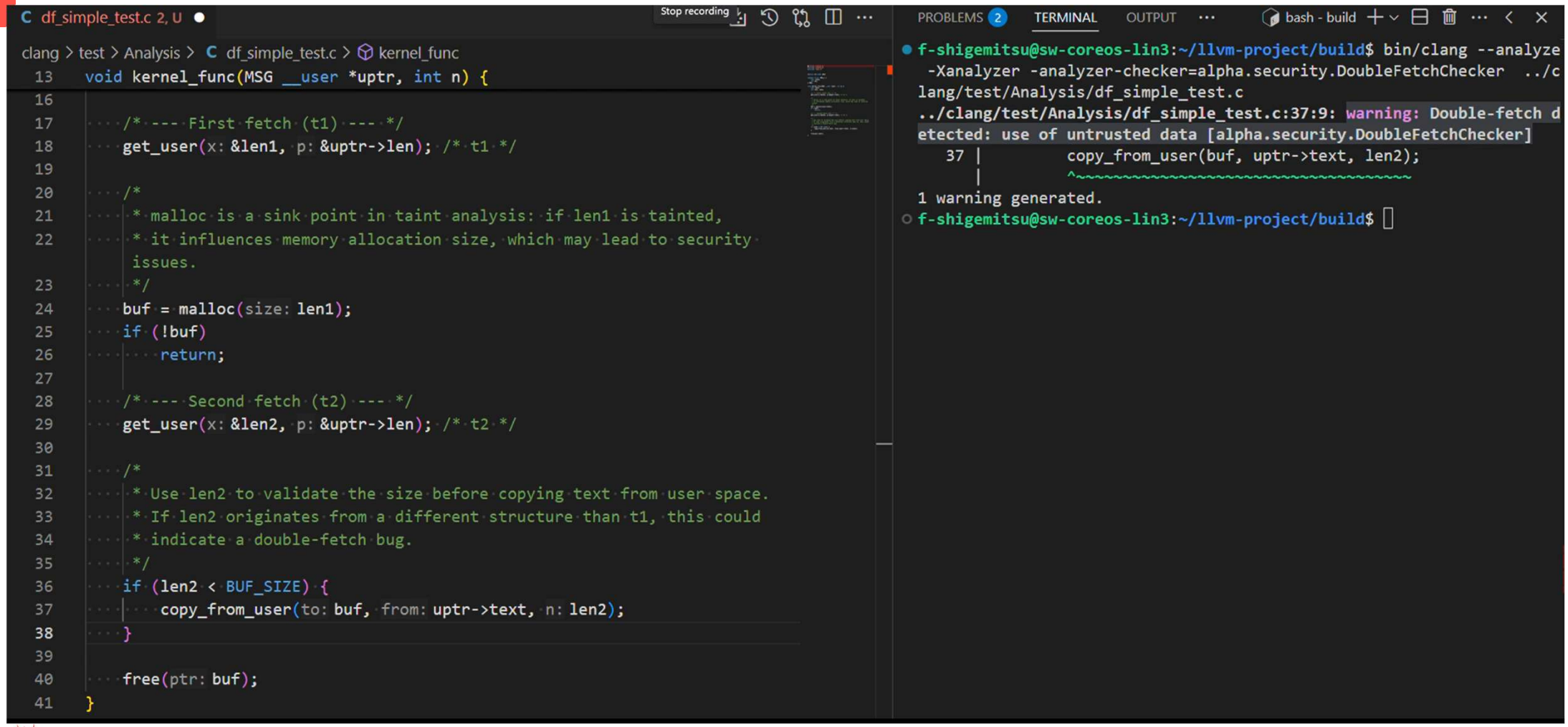
Alias relationships

- Track tags through all symbol aliases.
- Ensure that any alias retains its original provenance tag.

Control-flow analysis

- Propagate tags into if/else paths.
- Make a branch “tagged” whenever its condition uses tagged data.

Demo



How The Checker Reduce False Positives

- Pointer changing
- Use of the double-fetched data

Reference:

- P. Wang, K. Lu, G. Li, and X. Zhou, "A survey of the double-fetch vulnerabilities," *Concurrency and Computation*, vol. 30, no. 6, p. e4345, Mar. 2018, doi: 10.1002/cpe.4345.

Next Ideas

Future Ideas for RTOS/Hypervisor

- Product-specific API contract validation
- Interrupt Mask Duration policy checks
- Priority-Inversion detection
- Hypervisor Call validation

Additional Improvements for the Current Double-Fetch Checker:

- Application to large codebases
- Integration with the existing Clang Static Analyzer TOCTOU framework

Challenge with Passion