Migration-Relevant Software Idioms under Capability Systems

Fatih Durmaz, *Sven Bugiel*

MOTIVATION

- Legacy **software idioms** become **migration blockers** when **capability primitives** invalidate hidden assumptions.

```
uint64_t raw = (uint64_t)p;
void*q = (void*) raw;
```

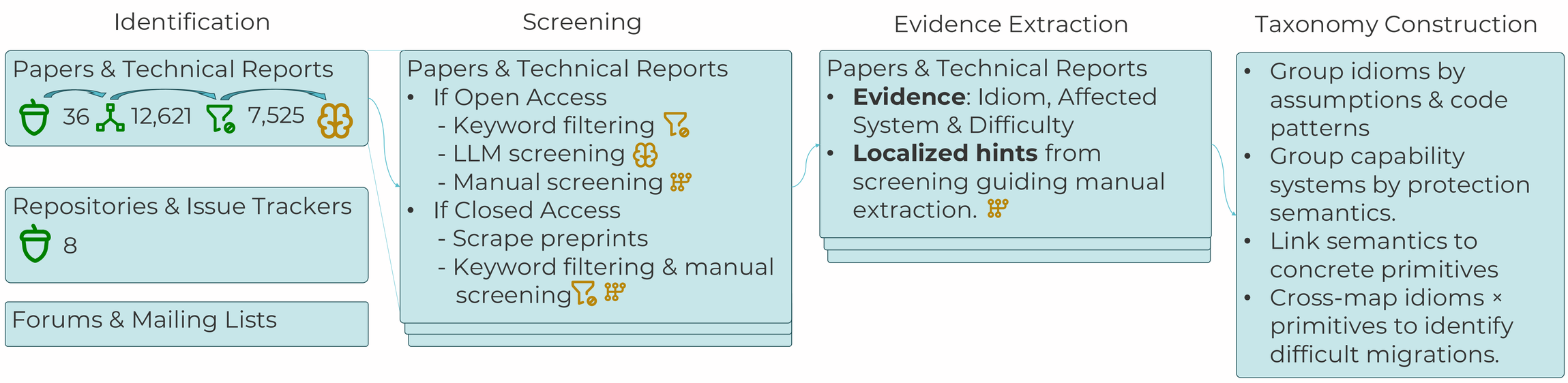
- Software compartmentalization is still **not widely adopted**
- **Lack of automation** and high **migration effort** is big reason.
- Certain software **idioms** make automated compartmentalization **difficult.**

RESEARCH QUESTIONS

RQ1: Which idioms across software stacks recur as obstacles during migration to capability systems?

RQ2: How do capability-system primitives shape these obstacles, and in which software-capability-system combinations is automation realistic?

METHODOLOGY



SELECTED IDIOM PATTERNS

Idiom Pattern	Hidden assumption	Exposing primitive type	Migration Symptom	Example
Dynamic loading/ Lazy initialization	Code can acquire authority later at runtime	Sandbox entry + explicit delegation of filesystem / loader authority.	dlopen() or late initialization fails inside compartment	SOAAP[1]
Enclosing-object recovery	A pointer to a field can recover the full enclosing object using layout arithmetic	Strict capability bounds, provenance, and validity tag	Field-bounded capability may not authorize access to the enclosing object	CHERI[2]
Type-unsafe zero initialization	A zero-initialized pointer field is safely readable as NULL.	On-load pointer authentication for in-memory pointers	NULL checks over calloc-initialized pointer fields may evaluate incorrectly.	Capacity[3]

IDIOM DIFFICULTY CRITERIA

Criterion	What it measures
Patch scope	How large and widespread the required change is
Semantic risk	Risk of changing intended program behaviour
Automation difficulty	Whether tools can recognize and transform the idiom reliably
Capability conflict	How directly the idiom violates capability system assumptions

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

1-Gudka, K., et al. “Clean Application Compartmentalization with SOAAP.” ACM CCS, 2015.
2-Watson, R. N. M., et al. “CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization.” IEEE Symposium on Security and Privacy, 2015.
3-Duy, K. D., Cho, K., Noh, T., and Lee, H. “Capacity: Cryptographically-Enforced In-Process Capabilities for Modern ARM Architectures.” arXiv:2309.11151, 2023.