LeakLess: Selective Data Protection Against Memory Leakage Attacks for Serverless Platforms

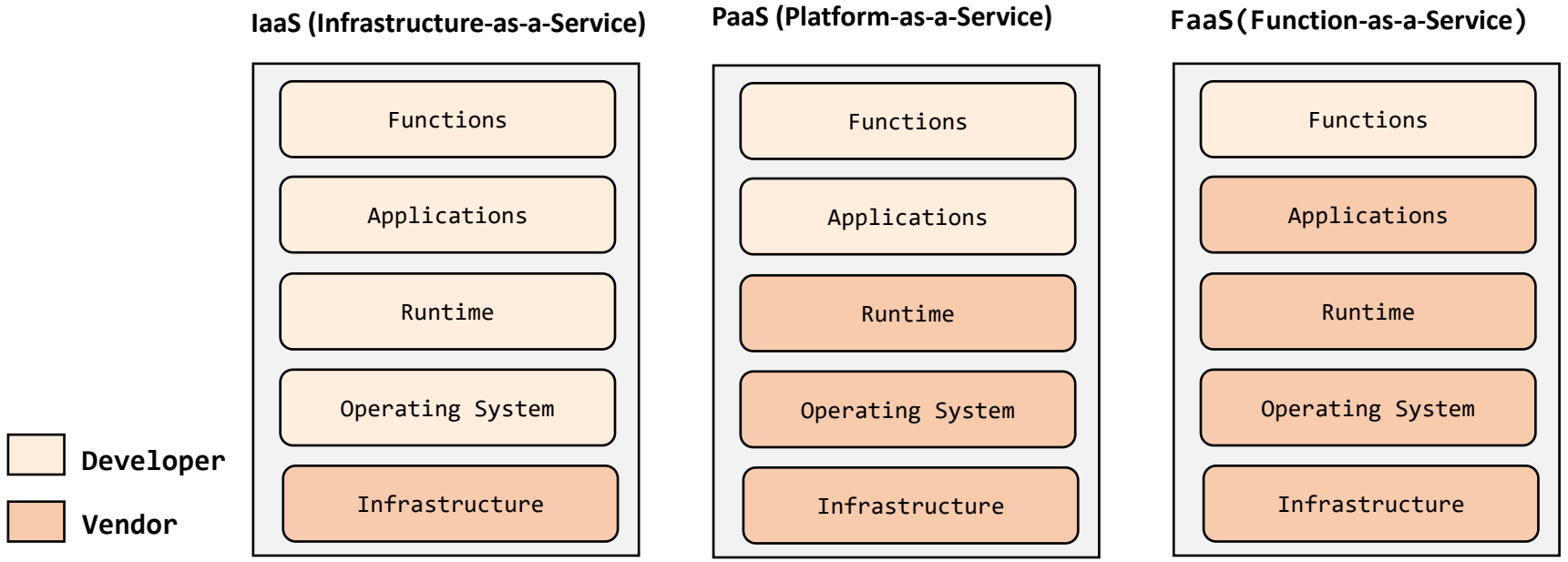
Maryam Rostamipoor, Seyedhamed Ghavamnia, Michalis Polychronakis

February 2025

Stony Brook University

Serverless Computing Platforms

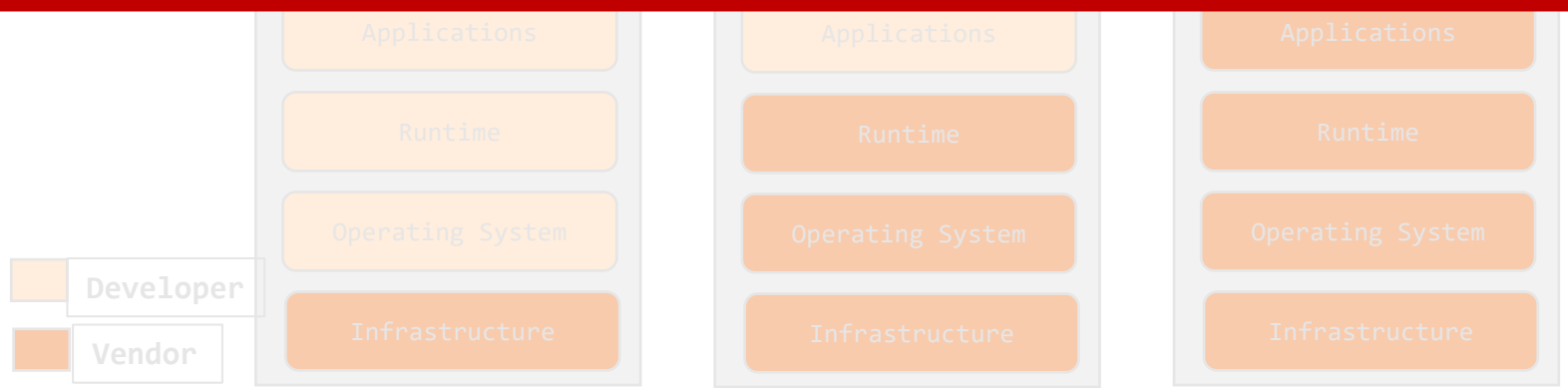
- ✓Also known as *Function-as-a-Service (**FaaS**)*
  - No need to manage servers—focus purely on code execution
  - Allow developers to run functions with on-demand scalability
  - Efficient resource allocation



Serverless Computing Platforms

- Also Known as Function-as-a-Service (**FaaS**)
  - Allows developers to run functions with on-demand scalability
  - No need to manage servers—focus purely on code execution
  - Efficient resource allocation and auto-scaling

Serverless platforms introduce **security challenges**, especially when multiple tenants share the same execution environment

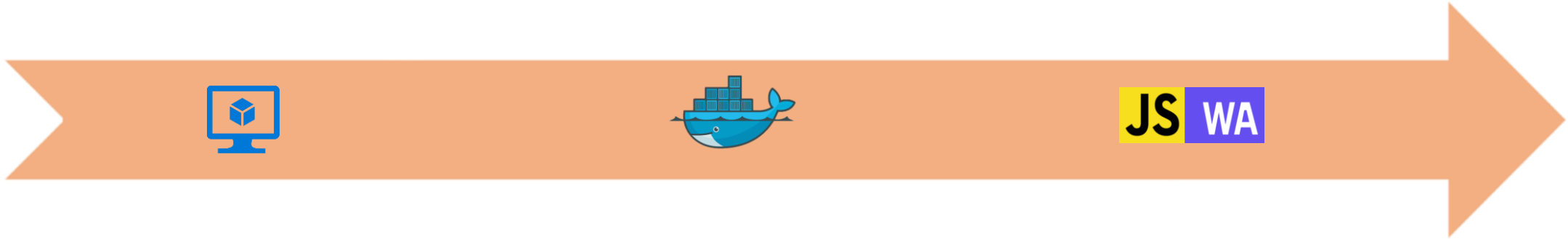


Function Isolation

Virtualization-based platforms

Container-based platforms

Language-based platforms



Low efficiency
More secure

High efficiency
Less secure

Memory Leakage Attacks

Bugs in the Language Runtime

Transient Execution Attacks

Memory Leakage Attacks

Bugs in the Language Runtime

Persistent compiler-related issues

Out-of-bounds memory access vulnerabilities

Transient Execution Attacks

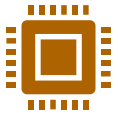
Memory Leakage Attacks

Bugs in the Language Runtime

Persistent compiler-related issues

Out-of-bounds memory access vulnerabilities

Transient Execution Attacks



Exploit speculative execution (e.g., Spectre)

Bypass memory safety, memory isolation, or enforced data flows

Effective defense remains a significant challenge

Sensitive Data in Serverless Applications

Serverless applications are a tempting target for attackers

Contain sensitive data

- API access keys
- Database connection strings

Handle sensitive data

- Session keys
- User passwords
- Credit card details

Sensitive Data in Serverless Applications

Serverless applications are a tempting target for attackers

Contain sensitive data

- API access keys
- Database connection strings

Handle sensitive data

- Session keys
- User passwords
- Credit card details

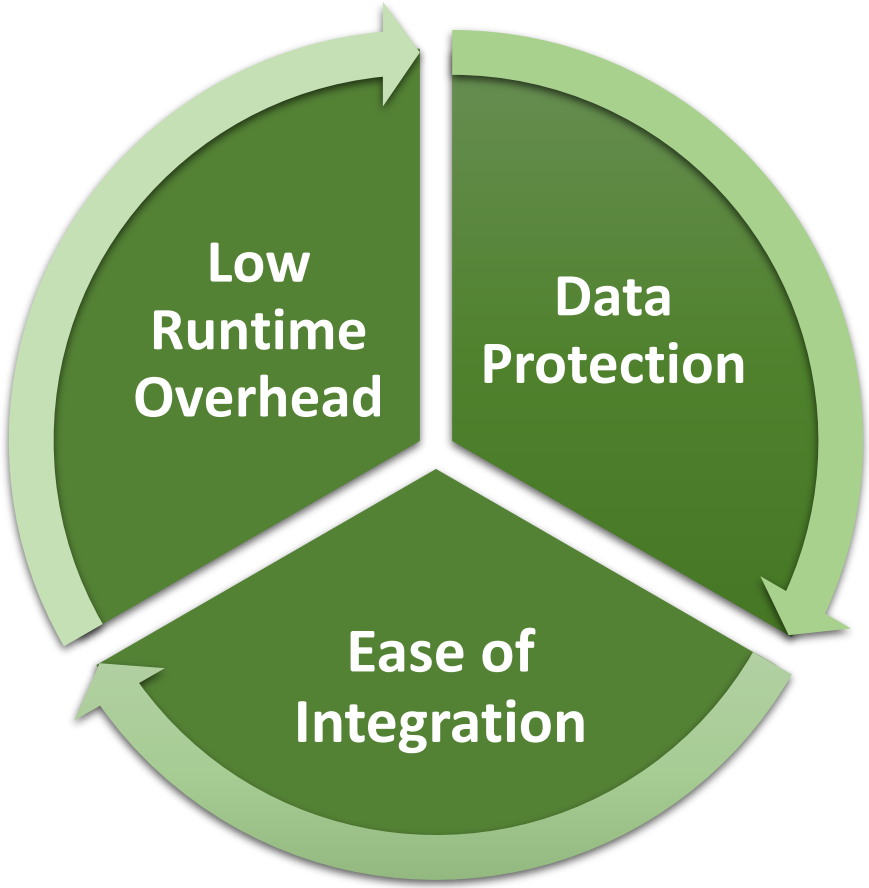
The existence of memory leakage attacks, combined with the presence of sensitive secrets, creates significant security challenges

Existing Works

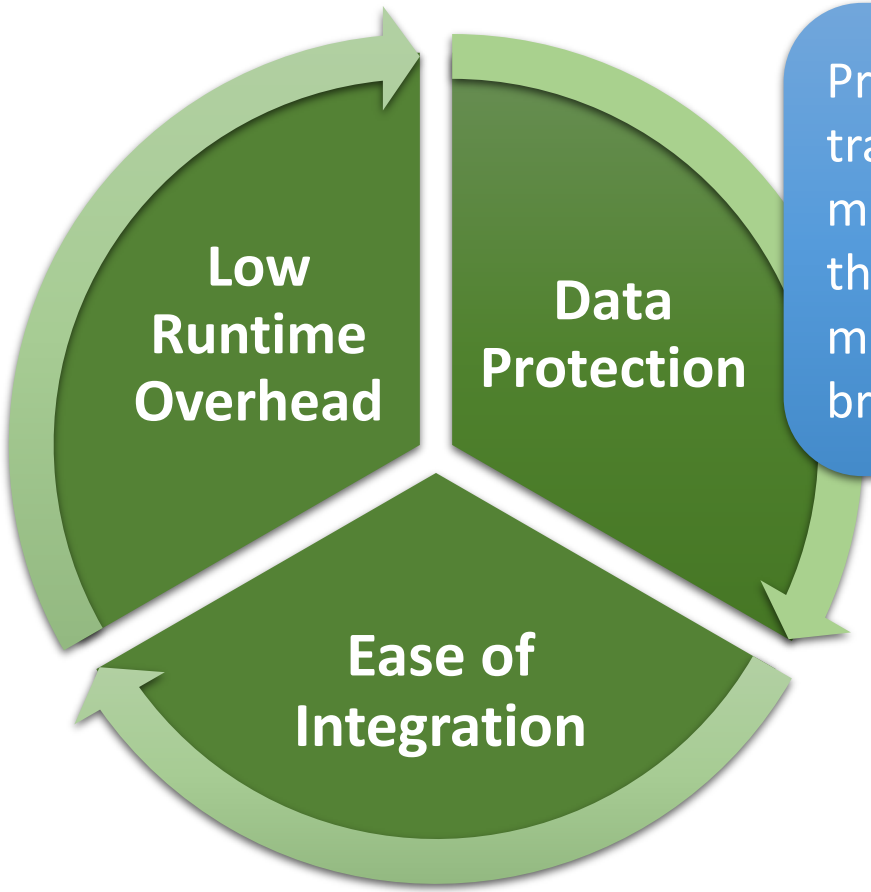
- **Ensuring memory safety** [Johnson et al., 2021; Bosamiya et al., 2022]
  - Verification-based approaches improve memory isolation in language-level sandboxes through binary verification or verified compilers
    - **Limitations**: Ensures only memory safety, difficult to adapt

- **Protecting against transient execution attacks** [Narayan et al., 2021, Schwarzl et al., 2022]
  - Defenses include compiler-based hardening and probabilistic detection techniques
    - **Limitations**: High performance overhead (240% in some cases), susceptibility to false positives/negatives, protect only against Spectre attacks

LeakLess Design Goals

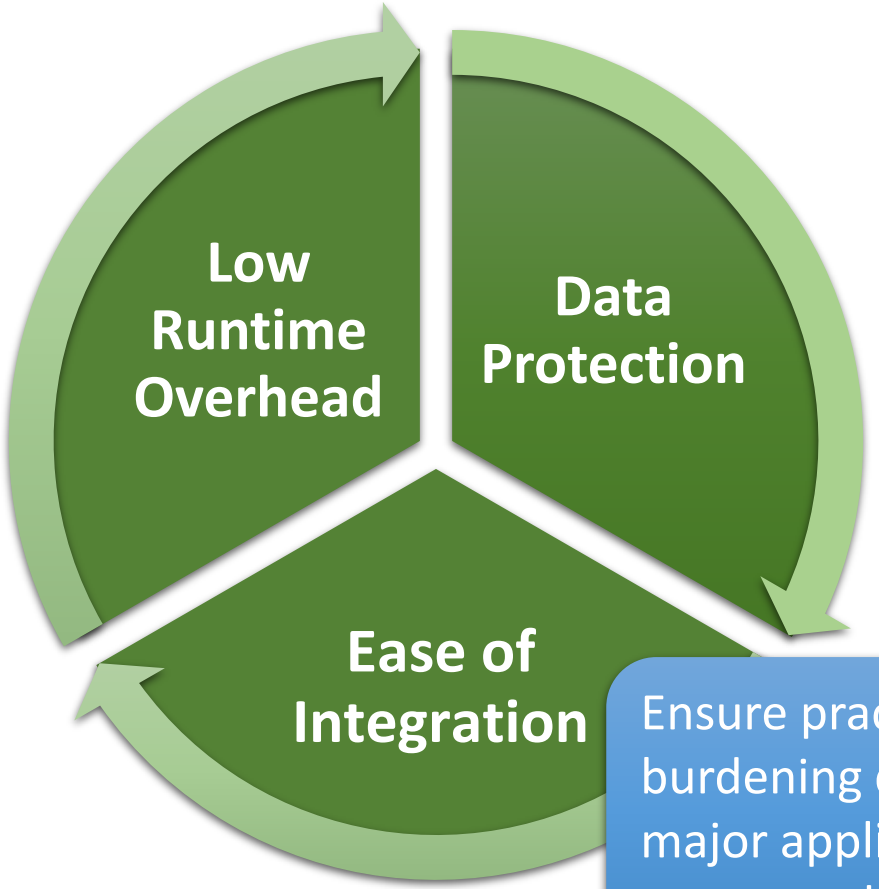


LeakLess Design Goals



Protect sensitive data from transient execution and memory disclosure attacks through a combination of in-memory encryption and I/O brokering

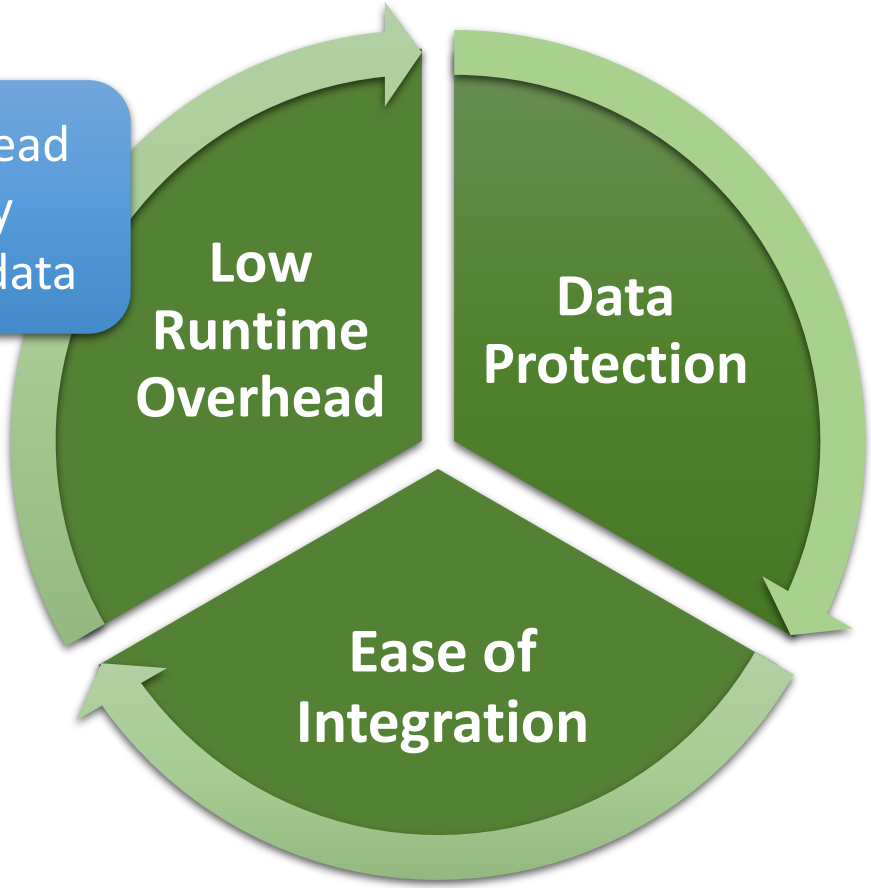
LeakLess Design Goals



Ensure practical protection without burdening developers or requiring major application changes, using simple source code data annotations

LeakLess Design Goals

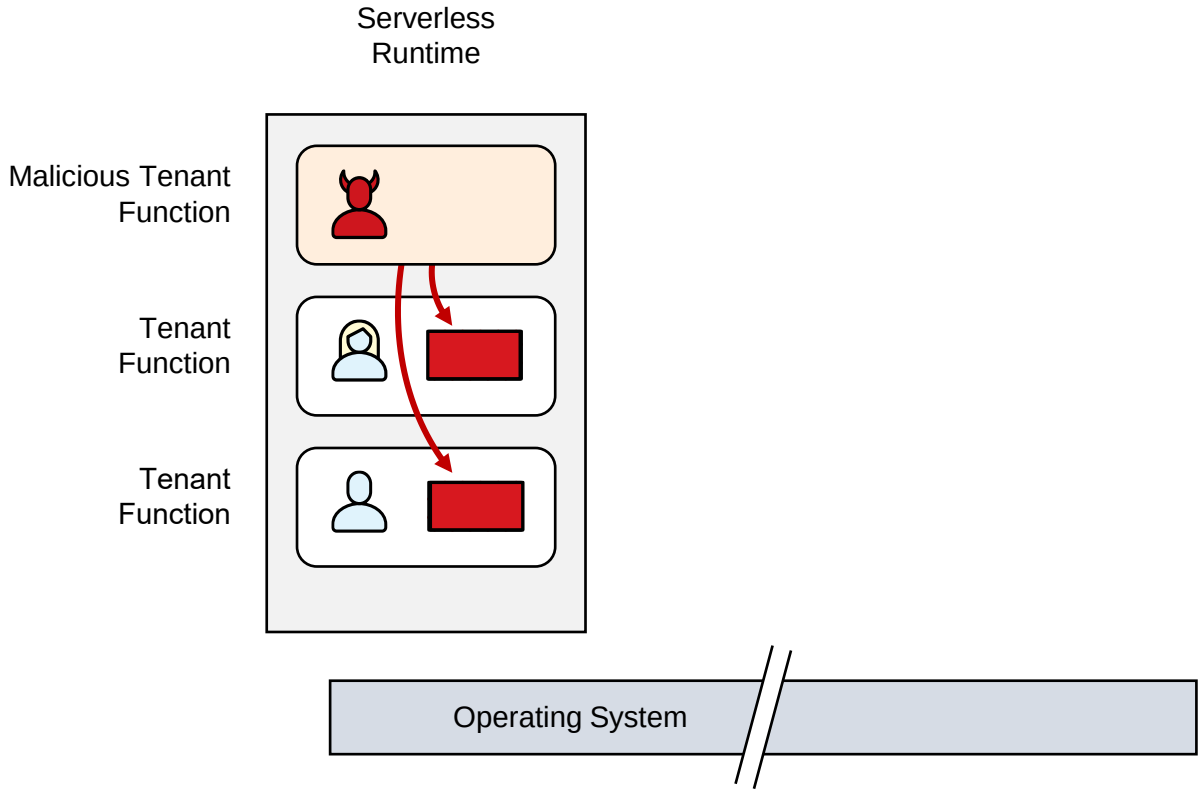
Maintain low runtime overhead by selectively protecting only sensitive data instead of all data



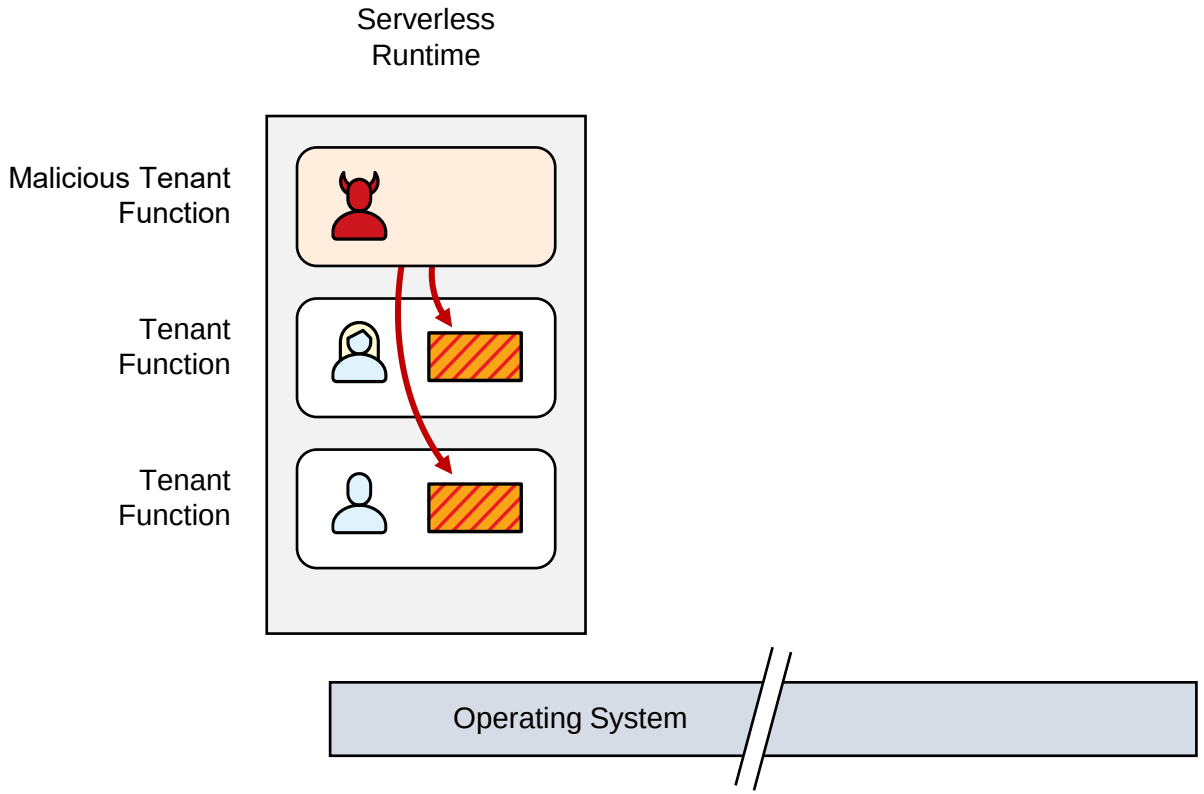
Threat Model

- Attacker may be a legitimate user of the serverless platform
- Attacker can exploit any data leakage vulnerability to read sensitive data belonging to other tenants
  - Memory disclosure vulnerabilities in the runtime
  - Spectre-style attacks and user-space Meltdown-style attacks
  - Cross-process transient execution attacks
- Attacker can not execute arbitrary code outside the runtime

LeakLess Design



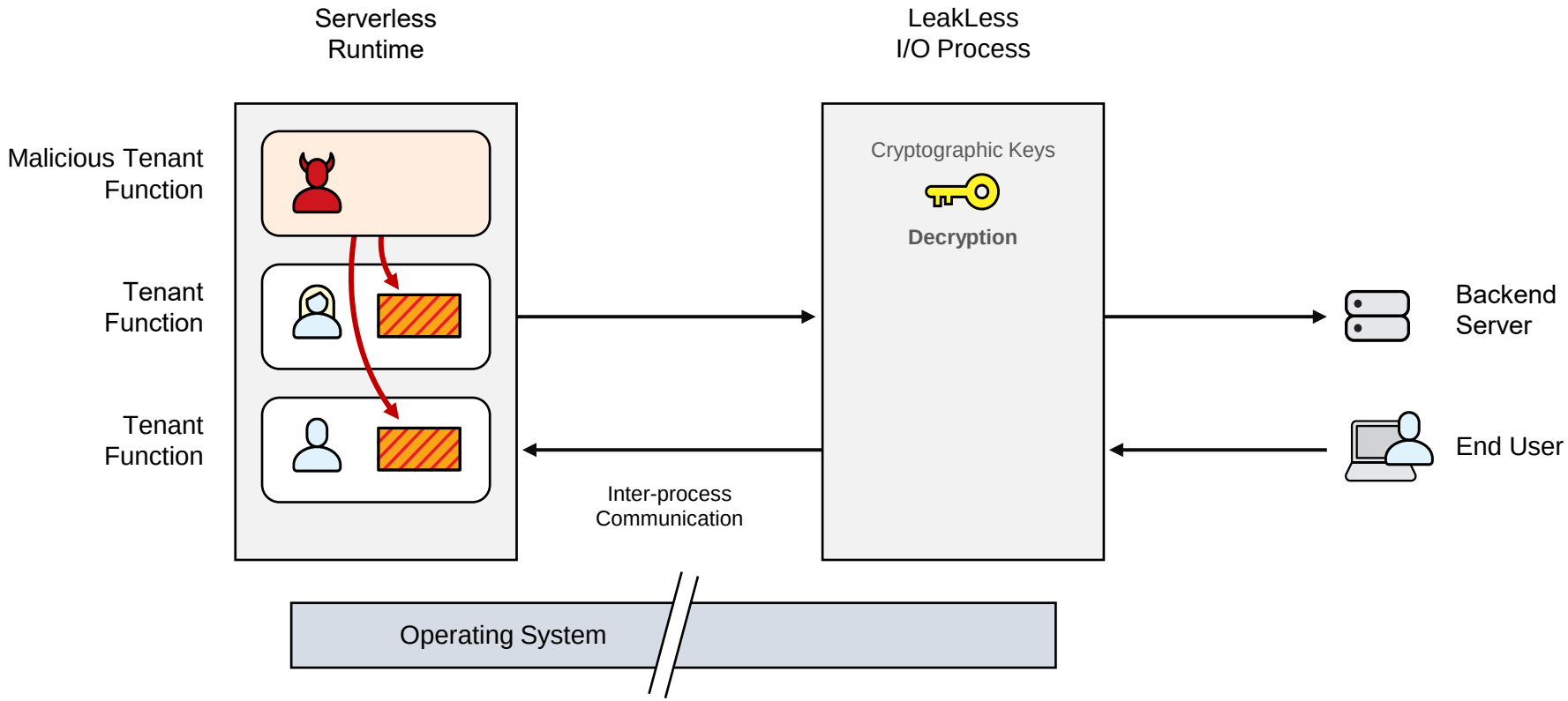
LeakLess Design



LeakLess protects sensitive data in serverless platforms by:

1. Always keeping sensitive data encrypted in memory

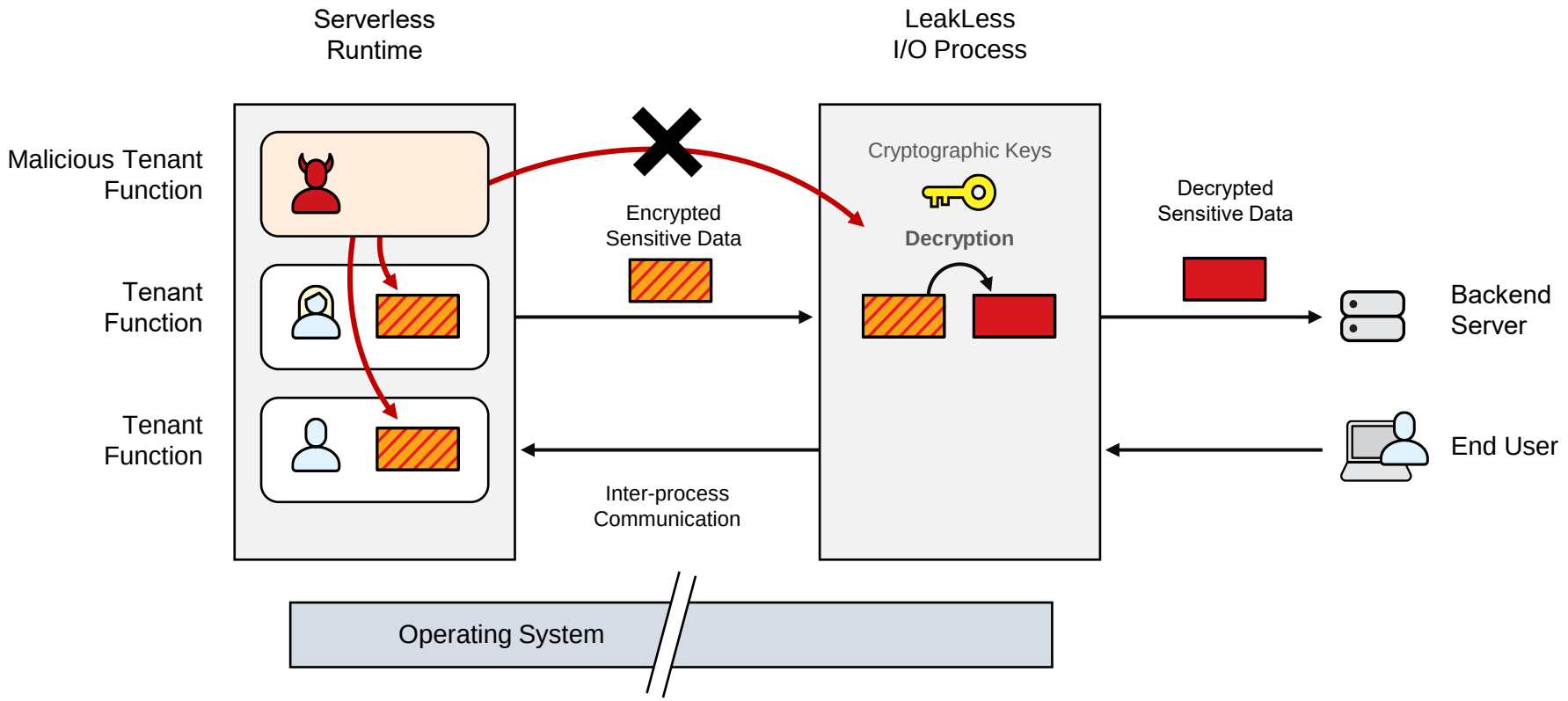
LeakLess Design



LeakLess protects sensitive data in serverless platforms by:

1. Always keeping sensitive data encrypted in memory
2. Handling cryptographic operations using a separate I/O module

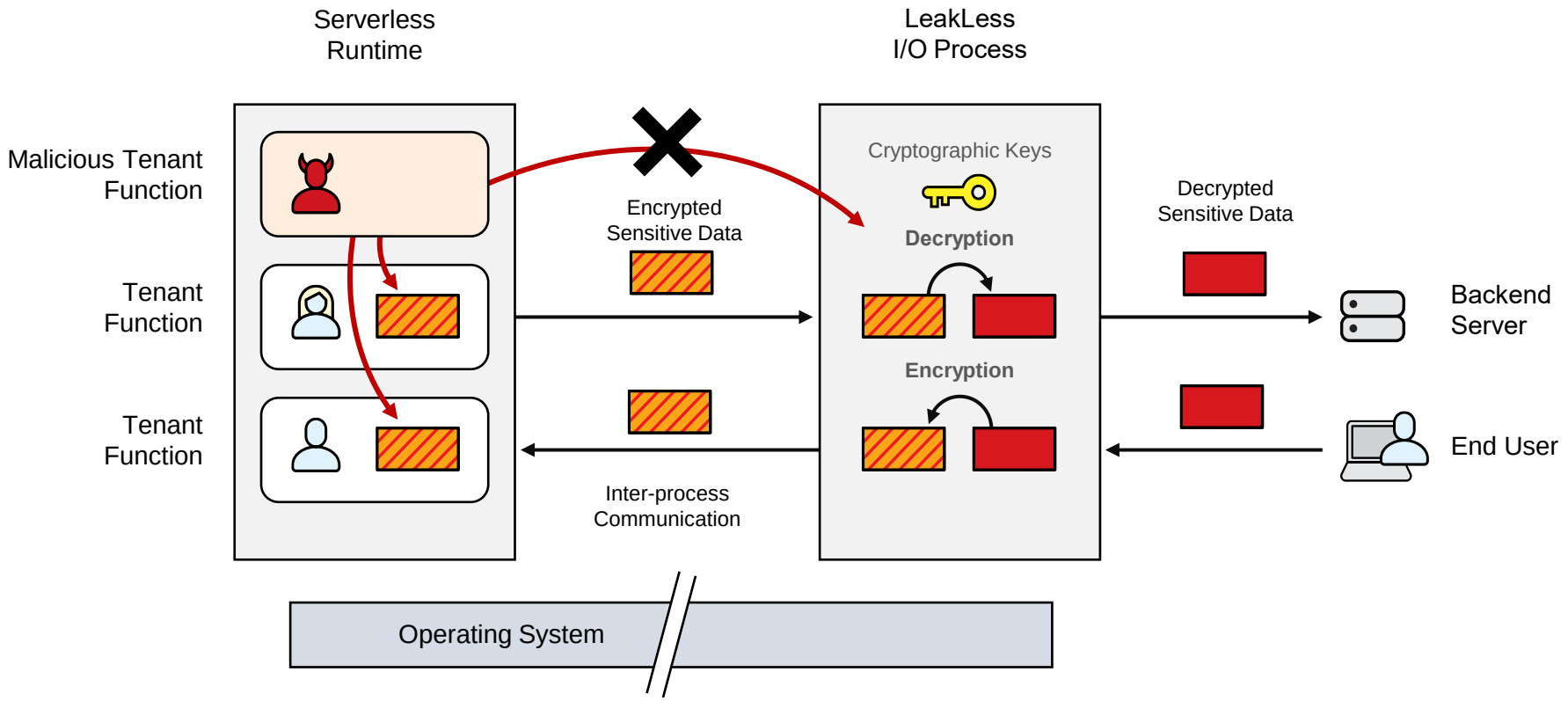
LeakLess Design



LeakLess protects sensitive data in serverless platforms by:

1. Always keeping sensitive data encrypted in memory
2. Handling cryptographic operations using a separate I/O module

LeakLess Design



LeakLess protects sensitive data in serverless platforms by:

1. Always keeping sensitive data encrypted in memory
2. Handling cryptographic operations using a separate I/O module

Sensitive Data Annotation

LeakLess Annotation

```
#[LEAKLESS_SECRET]
const AUTH_TOKEN: &str = "secret-token";

let mut res =
 spin_sdk::outbound_http::send_request(http::Request::builder()
 .method("GET")
 .header("Authorization", HeaderValue::from_bytes(AUTH_TOKEN)))
 .uri("https://backend_domain.com/image.jpg").body(None)?,)?;
```

Cross-process Flow of Sensitive data

```
GET /image.jpg HTTP/1.1
Host: backend_domain.com
Authorization: LEAKLESS_AfvUK704aQFsjnTCyfCTrA==
```

Encrypted + Marshaled Authorization Token

Compatibility Assessment

- Analyzed **1,074** serverless applications
  - Compatible with the vast majority of them
    - **449 applications** (42%) contain at least one sensitive data object
    - **LeakLess fully** supports **91% (407/449)** of these applications

Compatibility Assessment

- Analyzed **1,074** serverless applications
  - Compatible with the vast majority of them
    - **449 applications** (42%) contain at least one sensitive data object
    - **LeakLess fully** supports **91% (407/449)** of these applications
  - Broad support for sensitive data objects:
    - **966 sensitive data objects** identified across all applications
    - **LeakLess fully** protects **94% (912/966)** of sensitive data objects
    - **66%** are immutable and directly supported
    - **26%** are supported by outsourcing certain mutable operations (e.g., verification, signing) to the I/O module

Performance Evaluation

- **Real-World Testing:**
  - Evaluated on six *widely-used* serverless applications
  - Each application represents a different *type* of sensitive data and *operation*

Performance Evaluation

- **Real-World Testing:**
  - Evaluated on six *widely-used* serverless applications
  - Each application represents a different *type* of sensitive data and *operation*
- **Performance Impact:**
  - *Remote* Scenario (I/O module on a separate host):
    - Latency Increase: Up to **3.4%**
    - Throughput Decrease: Up to **2.8%**
  - *Local* Scenario (I/O module on the same host):
    - Latency Increase: Up to **9.7%**
    - Throughput Decrease: Up to **8.5%**

Summary

- **LeakLess: Efficient, Scalable, and Future-Proof Data Protection for Serverless Computing**
  - ✓ **Future-proof** against **memory leaks** and **transient execution attacks**
  - ✓ **Combines in-memory encryption** with a **dedicated I/O module** for enhanced security
  - ✓ **Easy Integration**: Uses **simple annotations**, minimizing developer effort
  - ✓ **Slight overhead** with **minimal effect on system efficiency**

https://github.com/mrostamipoor/LeakLess