Rethinking the OS for Isolation
Flexibility with FlexOS

Hugo Lefeuvre
The University of Manchester
Current OS Designs

OS security/isolation strategies are fixed at design time!
Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)
Current OS Designs

OS security/isolation strategies are **fixed** at design time!

Isolation granularity, underlying mechanisms, data sharing strategies (copy/share)

- Security
  - Micro/Separation kernels
  - Monolithic kernels
  - Single Address Space / Dataplane OSes

- Performance
  - Slower
  - Faster

- Removing user-kernel separation?
- Using hardware capabilities?
- Memory Protection Keys?
Decouple security/isolation decisions from the OS design

Achieve a range of trade-offs instead of a single point in the design space

Support a range of isolation mechanisms and granularities
Other Use-Cases for Flexible Isolation

Deployment to heterogeneous hardware
Make optimal use of each machine/architecture's safety mechanisms with the same code

Quickly isolate vulnerable libraries
React easily and quickly to newly published vulnerabilities while waiting for a full patch

Incremental verification of code-bases
Mix and match verified and non-verified code-bases while preserving guarantees
FlexOS 101: Approach in 4 points

1. Focus on **single-purpose appliances** such as cloud microservices

...the more applications run together, the least specialization you can achieve
FlexOS 101: Approach in 4 points

1. Focus on **single-purpose appliances** such as cloud microservices

   **Full-system (OS+app) understanding of compartmentalization**

2. Not "only application" or "only kernel": consider everything and **specialize**

   Embrace the **library OS philosophy**: everything is a library... network stack, nginx, libopenssl, sound driver, etc.
FlexOS 101: Approach in 4 points

1. Focus on **single-purpose appliances** such as cloud microservices

2. **Full-system** *(OS+app)* understanding of compartmentalization

3. **Abstract away** the technical details of isolation mechanisms

   Page table, MPK, CHERI, TEEs? Not the same guarantees, but a **similar interface can be achieved**.
FlexOS 101: Approach in 4 points

1. Focus on **single-purpose appliances** such as cloud microservices

   **Full-system** (*OS*+*app*) understanding of compartmentalization

2. Abstract away the technical details of isolation mechanisms

3. Flexibility must not **get into the way of performance**
FlexOS 101: Overview

Happy users: config.yaml

① Input config

FlexOS Toolchain

config.yaml

```yaml
compartments:
- comp1:
  mechanism: intel-mpk
  default: True
- comp2:
  mechanism: intel-mpk
  hardening: [cfi, asan]
libraries:
- libredis: comp1
- libopenjpg: comp2
- lwip: comp2
```

"Redis image with two compartments, isolate libopenjpeg and lwip together"
FlexOS 101: Overview

1. Input config
2. Select isolation mechanism ("Backend")
3. Select libraries (kernel and app), rewrite, and statically put in compartments
4. Generate image with appropriate isolation properties

Isolation Backends
- MPK
- EPT
- ...

Core Libraries
- boot
- sched
- mm
- ramfs
- vfs
- ...

Possible Image 1
- Possible Image 2

Happy users

FlexOS Toolchain

Input config: config.yaml
FlexOS 101: Mechanism Abstraction

Based on a **highly modular LibOS design** (Unikraft)

Such libOSes are composed of **fine-granular, independent** libraries.

Reuse libraries as finest granularity of compartmentalization.

"Pre-compartmentalize" them.

Cross-library **calls and shared data** are replaced by an **abstract construct** (**gates**, data sharing primitives).

Define them as part of the **FlexOS API**.

At build time, these abstract constructs are replaced with a particular implementation by the toolchain. These implementations are defined by the **backends**.

![Diagram showing Core Libraries and Kernel & User Libs](image)
### FlexOS 101: Compartmentalization API

<table>
<thead>
<tr>
<th>int rc, connfd;</th>
<th>Add gate placeholders</th>
</tr>
</thead>
<tbody>
<tr>
<td>char buf[512];</td>
<td></td>
</tr>
<tr>
<td>/* ... */</td>
<td></td>
</tr>
<tr>
<td>rc = recv(connfd, buf, 512, 0);</td>
<td>Annotate shared data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int rc, connfd;</th>
<th>Porting</th>
</tr>
</thead>
<tbody>
<tr>
<td>char buf[512] <strong>attribute</strong>((flexos_share));</td>
<td></td>
</tr>
<tr>
<td>/* ... */</td>
<td></td>
</tr>
<tr>
<td>rc = flexos_gate(lblwip, recv, connfd, buf, 512, 0);</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int rc, connfd;</th>
<th>Automatic gate instantiation at build time</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *buf[512] = shared_malloc(512);</td>
<td></td>
</tr>
<tr>
<td>/* ... */</td>
<td></td>
</tr>
<tr>
<td>rc = mpk_gate(0, 1, recv, connfd, buf, 512, 0);</td>
<td>Replace with MPK gate</td>
</tr>
</tbody>
</table>

Replace with shared heap allocation

Porting

Coccinelle

Lwip

App

1

0
### FlexOS 101: Compartmentalization API

```c
int rc, connfd;
char buf[512];
/* ... */
rc = recv(connfd, buf, 512, 0);  // Add gate placeholders
```

**Annotate shared data**

```c
int rc, connfd;
char buf[512] __attribute__((flexos_share));
/* ... */
rc = flexos_gate(liblwip, recv, connfd, buf, 512, 0);
```

**Automatic gate instantiation at build time**

```c
int rc, connfd;
char *buf[512] = shared_malloc(512);
/* ... */
rc = mpk_gate(0, 1, recv, connfd, buf, 512, 0);
```

**Replace with normal stack allocation**

**Replace with MPK gate**

**Replace with shared heap allocation**

**Porting**

**Replace with function call**

**Coccinelle**
Prototype

Implementation on top of Unikraft

Backend implementations for Intel MPK and VMs (EPT)

Port of libraries: network stack, scheduler, filesystem, time subsystem

Port of applications: Redis, Nginx, SQLite, iPerf server

This talk: focus on demonstrating flexibility and performance

more results in our paper 😊
Flexibility

Runtime performance with Redis in requests/s

FlexOS libraries used in the Redis image
(only a subset for readability):

- Redis application
- C standard library (newlib)
- FlexOS scheduler (uksched)
- Network stack (lwip)
Flexibility

Runtime performance with Redis in requests/s

One configuration and its associated performance (80 configurations in total)

FlexOS libraries used in the Redis image (only a subset for readability):
- Redis application
- C standard library (newlib)
- FlexOS scheduler (uksched)
- Network stack (lwip)

The color of boxes indicates the compartment:
- Compartment 1
- Compartment 2
- Compartment 3

The dot whether hardening (ASan, Safestack, etc.) is enabled:
- Hardening on
- Hardening off
Flexibility

1. Large safety / performance space! (4x)

2. Smooth slope, performance degrades gracefully

- Hardening on
- Hardening off
- Compartment 1
- Compartment 2
- Compartment 3
Flexibility

2 crossings

lwip

uksched

Performance-wise:

= 

2 crossings

lwip

uksched

You can get some safety for free by exploring intelligently

Similar performance, very different properties!

need to reason about communication patterns, fast paths
Performance

Time to perform 10K SQLite INSERT queries in seconds

Number of compartments and mechanism (e.g., PT2 = 2 compartments with the page table)

VMM/environment
Performance

① No overhead when disabling isolation – you only pay for what you get
The MPK backend compares very positively to competing solutions.

- Tricky comparison with CubicleOS - they're using linuxu, a Linux userland debug platform of Unikraft.
Performance

③ The EPT backend too compares positively to competing solutions
In a Nutshell

There is a **need for isolation flexibility**
- OS Specialization, hardware heterogeneity
- or quickly react to vulnerabilities!

Current approaches: **one isolation approach at design time**

Decouple isolation from the OS design:
- Make isolation decisions at **build time**
- Explore **performance v.s. security trade-offs**
Interested?

Webpage: https://project-flexos.github.io/
Pre-print of our ASPLOS'22 paper: https://arxiv.org/abs/2112.06566
By e-mail: hugo.lefeuvre@manchester.ac.uk

License: 3-Clause BSD License 😊

Vlad-Andrei Bădoiu, Alexander Jung, Stefan Teodorescu, Sebastian Rauch, Felipe Huici, Costin Raiciu, Pierre Olivier