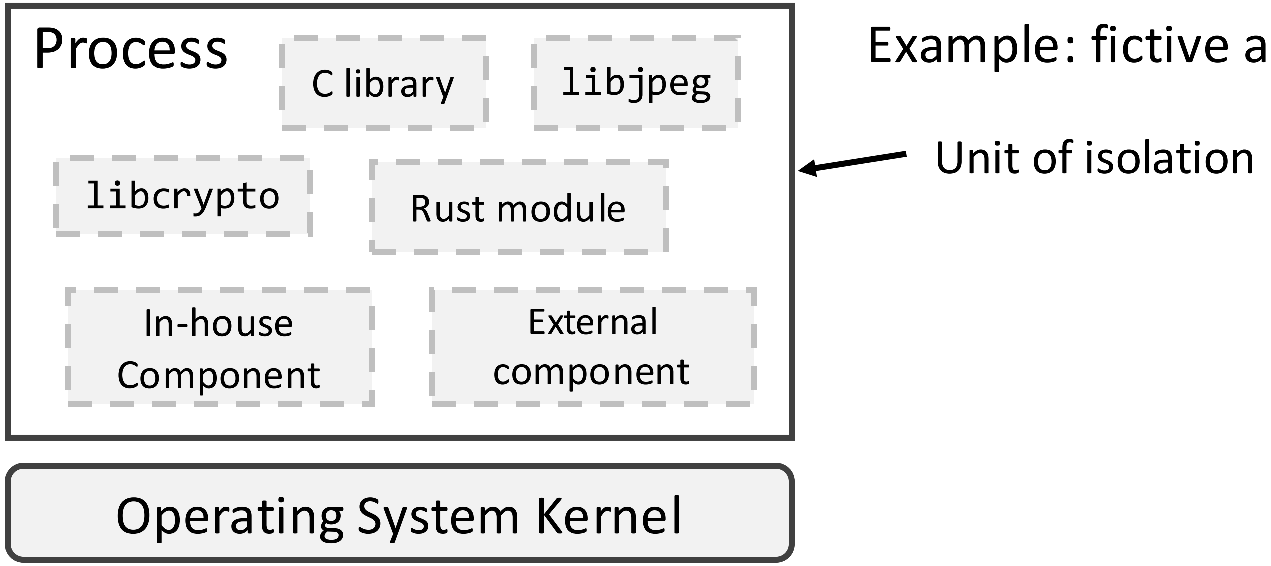
Software Compartmentalization Everywhere - *What Will it Take*?

Hugo Lefeuvre
Postdoctoral Research Fellow
University of British Columbia
Vancouver (🇨🇦)

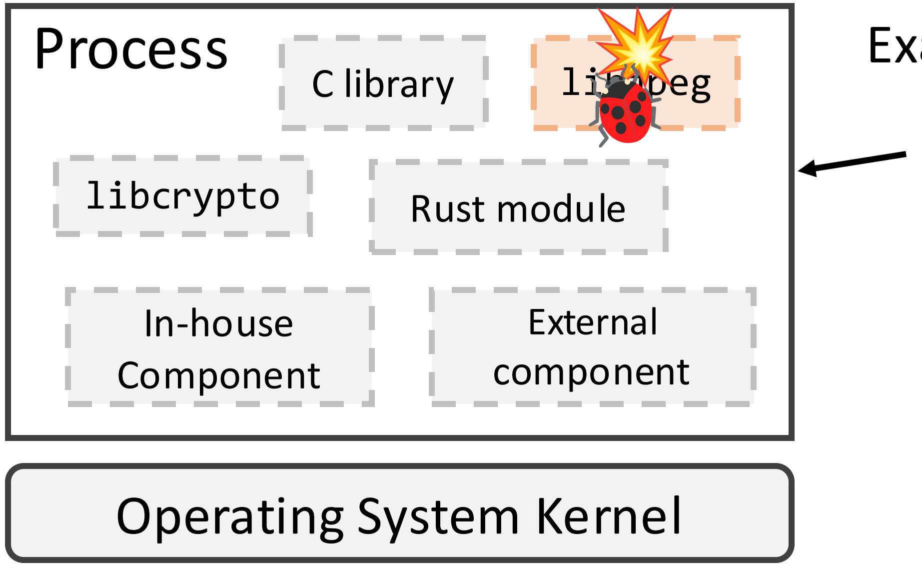
Historically, **program = unit of trust**

Historically, **program = unit of trust**



Example: fictive application

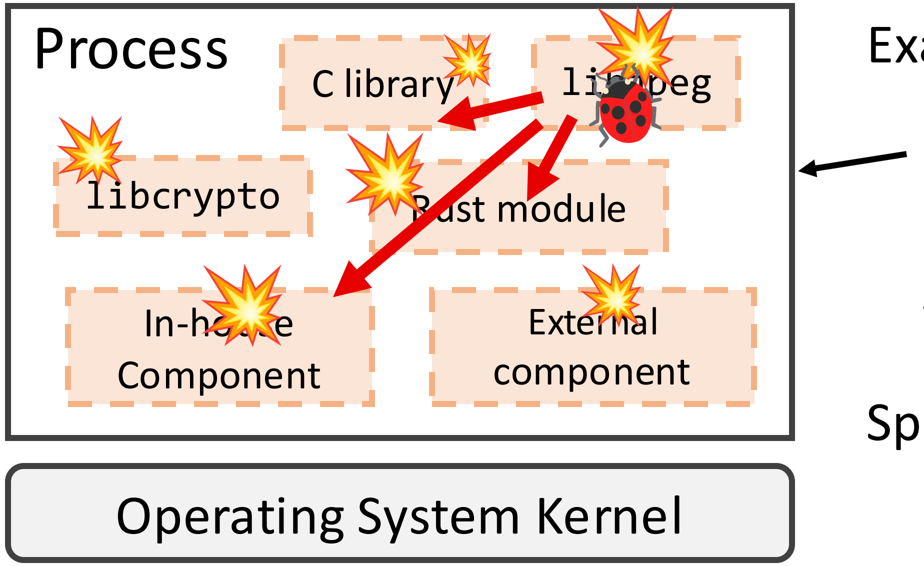
Historically, **program = unit of trust**



Example: fictive application

Attacker exploits bug in one of the libraries

Historically, **program = unit of trust**

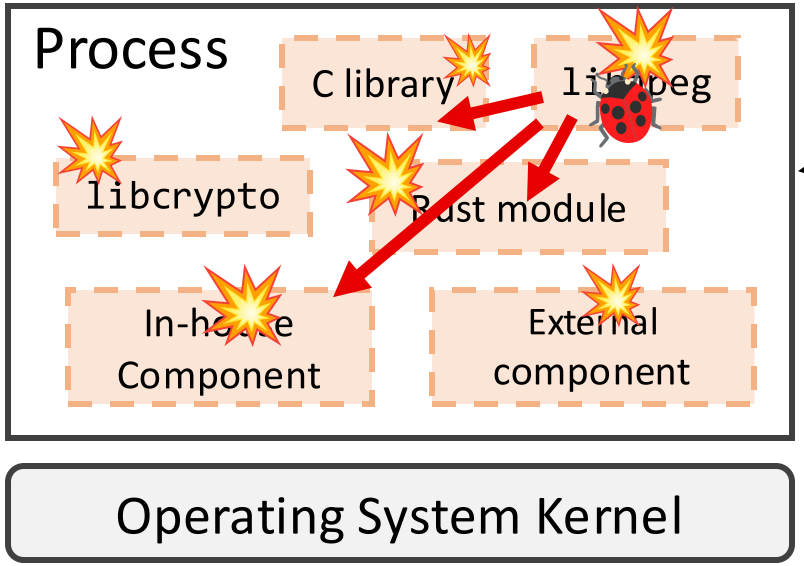


Example: fictive application

Attacker exploits bug in one of the libraries

Spreads trivially because of **ambient trust**

Historically, **program = unit of trust**



Example: fictive application

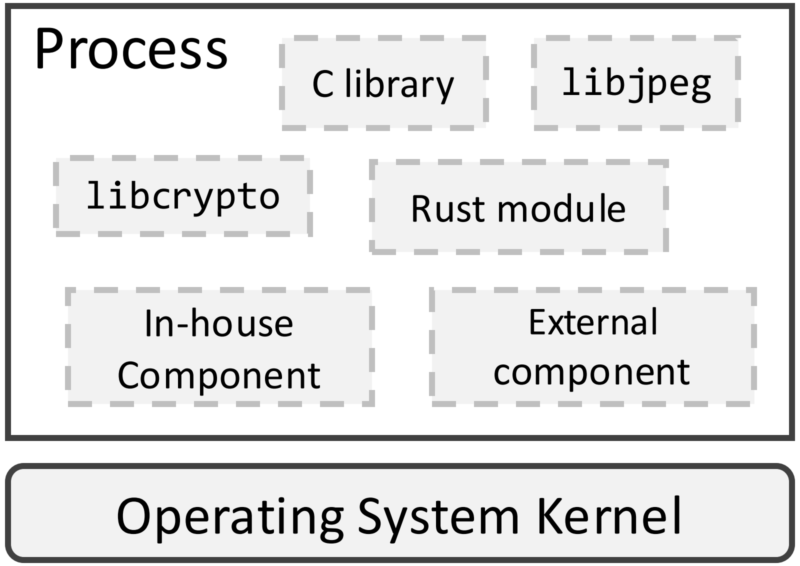
Unit of isolation

Attacker exploits bug in one of the libraries

Spreads trivially because of **ambient trust**

This historical approach is **increasingly problematic**

Historically, **program = unit of trust**



This historical approach is **increasingly problematic**

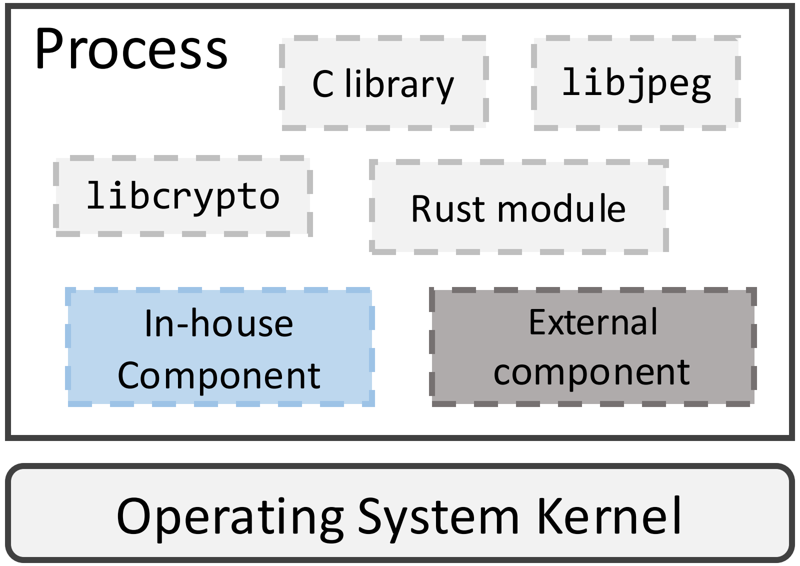
Among others...

Modern programs are **deeply heterogeneous**

Among others...

Historically, **program = unit of trust**

Modern programs are **deeply heterogeneous**



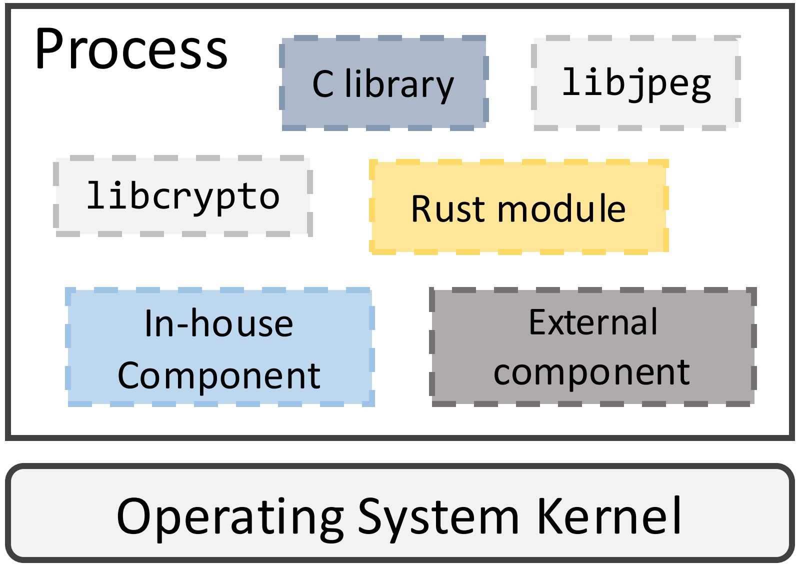
Different origins...

This historical approach is **increasingly problematic**

Among others...

Historically, **program = unit of trust**

Modern programs are **deeply heterogeneous**

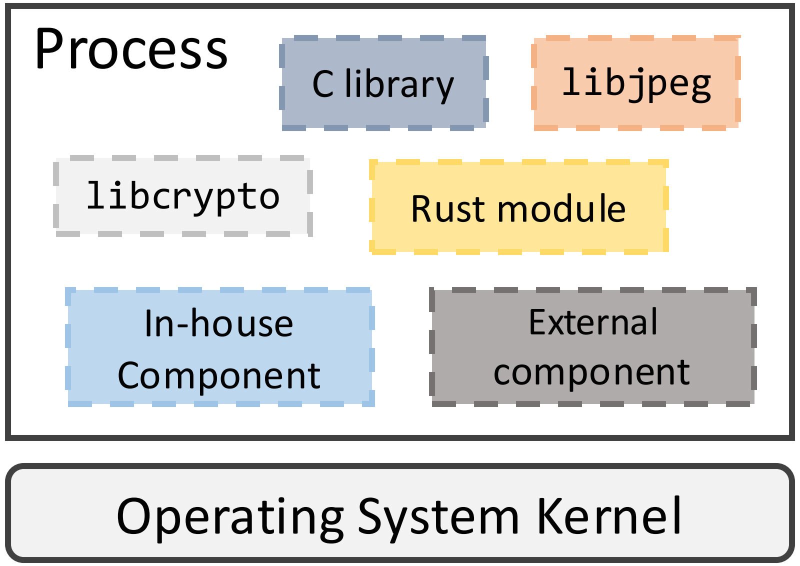


Different programming languages...

Different origins...

This historical approach is **increasingly problematic**

Historically, **program = unit of trust**



This historical approach is **increasingly problematic**

Among others...

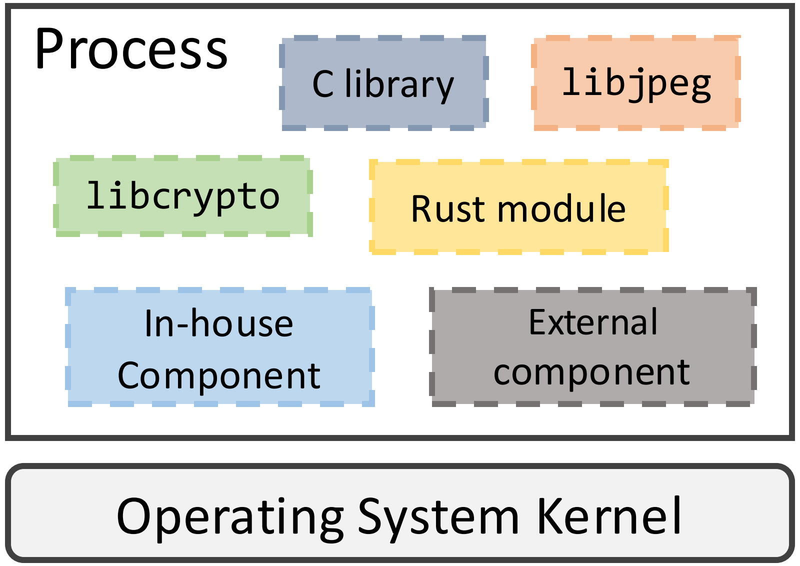
Modern programs are **deeply heterogeneous**

Different programming languages...

Different origins...

Different degrees of risk...

Historically, **program = unit of trust**



This historical approach is **increasingly problematic**

Among others...

Modern programs are **deeply heterogeneous**

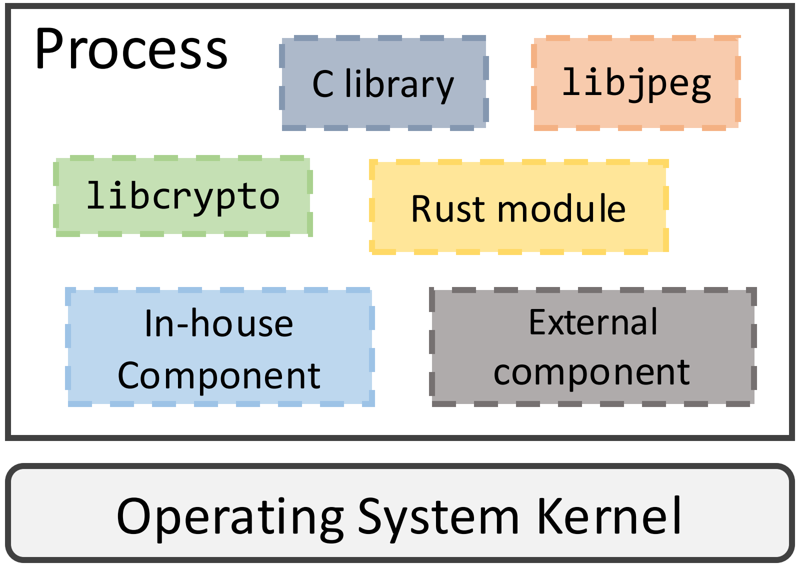
Different programming languages...

More or less valuable data...

Different origins...

Different degrees of risk...

Historically, **program = unit of trust**



This historical approach is
increasingly problematic

Among others...

Modern programs are
deeply heterogeneous

Different programming languages...

More or less valuable data...

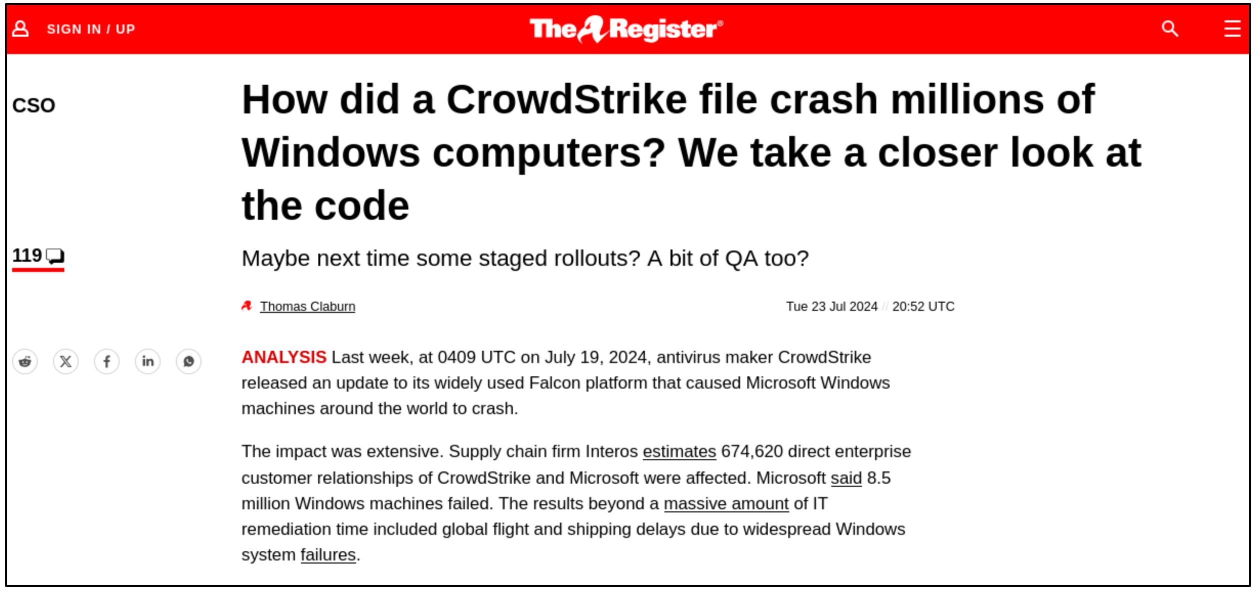
Different origins...

Different degrees of risk...

(1)

The average application has around 180 open-source components *— that's an increase from around 150 from which we found last year.*

(1) 10th Annual State of the Software Supply Chain, Sonatype Inc.


SIGN IN / UP

The Register

CSO

How did a CrowdStrike file crash millions of Windows computers? We take a closer look at the code

119

Maybe next time some staged rollouts? A bit of QA too?

Thomas Claburn Tue 23 Jul 2024 // 20:52 UTC

ANALYSIS Last week, at 0409 UTC on July 19, 2024, antivirus maker CrowdStrike released an update to its widely used Falcon platform that caused Microsoft Windows machines around the world to crash.

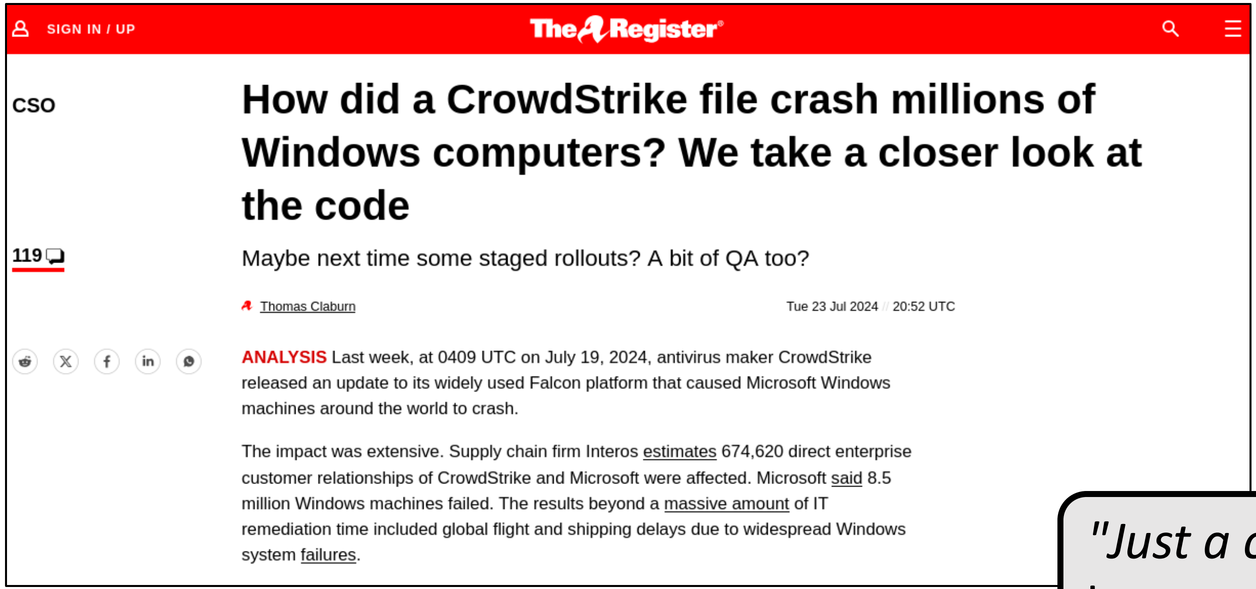
The impact was extensive. Supply chain firm Interos estimates 674,620 direct enterprise customer relationships of CrowdStrike and Microsoft were affected. Microsoft said 8.5 million Windows machines failed. The results beyond a massive amount of IT remediation time included global flight and shipping delays due to widespread Windows system failures.

(1)

Illustration of this problem:

Last year's Crowdstrike disaster where **a bug in an external antivirus plug-in crashes millions of computers**

(1) `https://www.theregister.com/2024/07/23/crowdstrike_failure_shows_need_for/`



SIGN IN / UP

The Register

CSO

How did a CrowdStrike file crash millions of Windows computers? We take a closer look at the code

119

Maybe next time some staged rollouts? A bit of QA too?

Thomas Claburn — Tue 23 Jul 2024 // 20:52 UTC

ANALYSIS Last week, at 0409 UTC on July 19, 2024, antivirus maker CrowdStrike released an update to its widely used Falcon platform that caused Microsoft Windows machines around the world to crash.

The impact was extensive. Supply chain firm Interos estimates 674,620 direct enterprise customer relationships of CrowdStrike and Microsoft were affected. Microsoft said 8.5 million Windows machines failed. The results beyond a massive amount of IT remediation time included global flight and shipping delays due to widespread Windows system failures.

(1)

"Just a crash": what if it had been a supply-chain attack?

Illustration of this problem:

Last year's Crowdstrike disaster where **a bug in an external antivirus plug-in crashes millions of computers**

(1) `https://www.theregister.com/2024/07/23/crowdstrike_failure_shows_need_for/`


SIGN IN / UP
The Register
CSO
How did a CrowdStrike file crash millions of Windows computers? We take a closer look at the code
Maybe next time some staged rollouts? A bit of QA too?
119
Thomas Claburn Tue 23 Jul 2024 // 20:52 UTC

ANALYSIS Last week, at 0409 UTC on July 19, 2024, antivirus maker CrowdStrike released an update to its widely used Falcon platform that caused Microsoft Windows machines around the world to crash.

The impact was extensive. Supply chain firm Interos estimates 674,620 direct enterprise customer relationships of CrowdStrike and Microsoft were affected. Microsoft said 8.5 million Windows machines failed. The results beyond a massive amount of IT remediation time included global flight and shipping delays due to widespread Windows system failures.

(1)

"Just a crash": what if it had been a supply-chain attack?

Illustration of this problem:
Last year's Crowdstrike disaster where **a bug in an external antivirus plug-in crashes millions of computers**

Program = single unit of trust

(1) `https://www.theregister.com/2024/07/23/crowdstrike_failure_shows_need_for/`

Compartmentalization

Compartmentalization

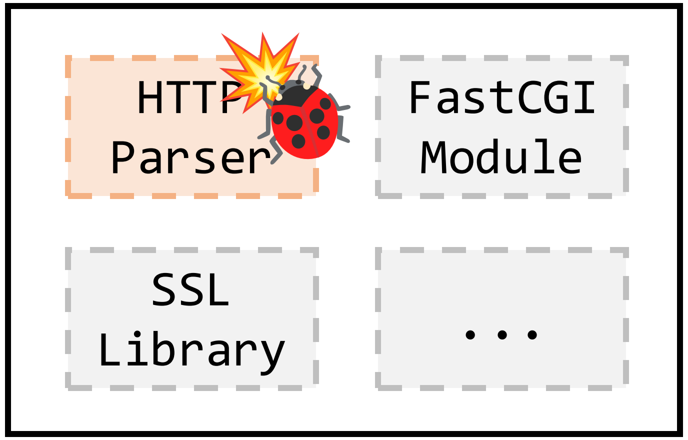
Software compartmentalization =

- design programs **split into distrusting and isolated components**
- such that if one of the components is breached, the attacker **does not compromise the whole program**

Compartmentalization

Software compartmentalization =

- design programs **split into distrusting and isolated components**
- such that if one of the components is breached, the attacker **does not compromise the whole program**



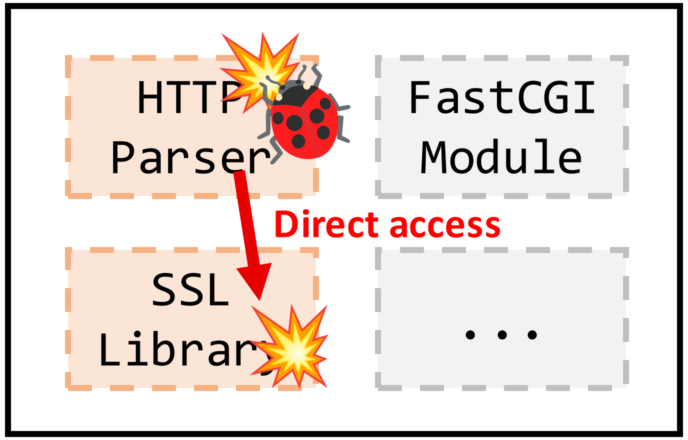
Monolithic

Example: a web server (Apache)

Compartmentalization

Software compartmentalization =

- design programs **split into distrusting and isolated components**
- such that if one of the components is breached, the attacker **does not compromise the whole program**



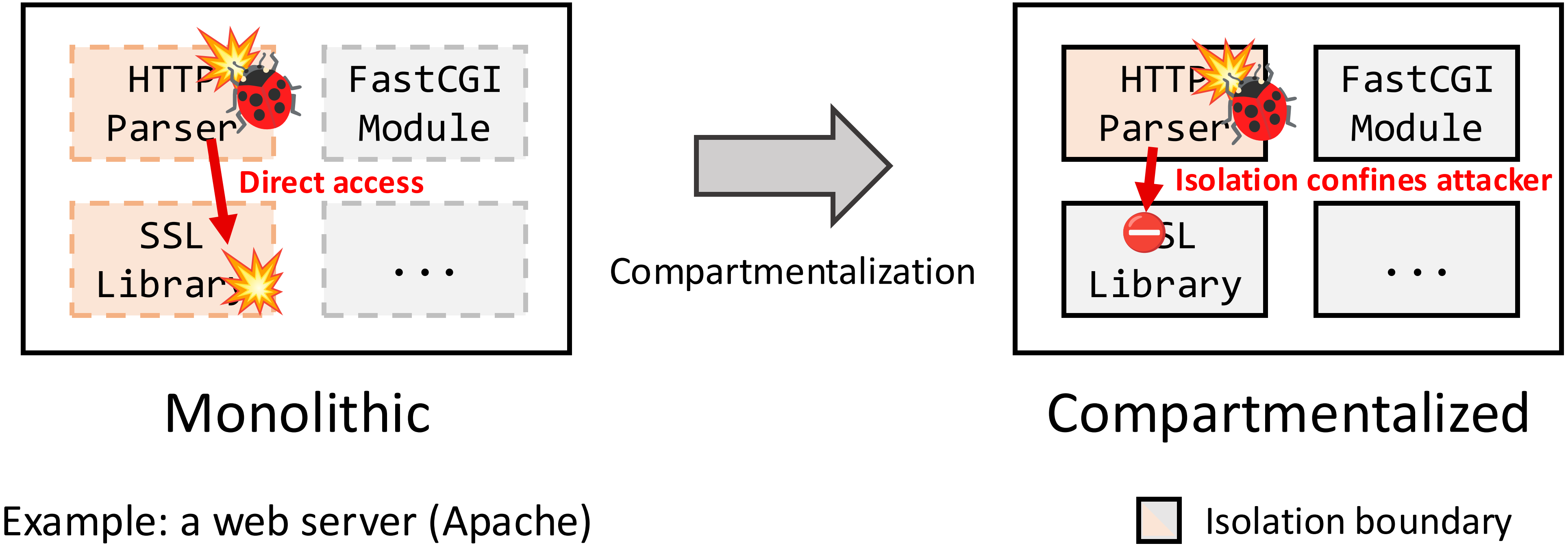
Monolithic

Example: a web server (Apache)

Compartmentalization

Software compartmentalization =

- design programs **split into distrusting and isolated components**
- such that if one of the components is breached, the attacker **does not compromise the whole program**



Example: a web server (Apache)

Compartmentalization

Software compartmentalization =

- design programs **split into distrusting and isolated components**
- such that if one of the components is breached, the attacker **does not compromise the whole program**

Compartmentalization is **applied to high-profile software**

Server Software

Web Browsers

But also...

Compartmentalization

Significant research funding in recent years

Softwa

- desig … solated components
- such … hed, the attacker **does not** c

£70M

Area of investment and support

Digital security by design

The digital security by design (DSbD) challenge funds business and researchers to update the foundation of the insecure digital computing infrastructure by creating a new, more secure hardware and software ecosystem.

Budget: £70 million

Duration: From 2020 to 2025

Partners involved: Innovate UK, Engineering and Physical Sc (EPSRC)

£12M

Funding oppo

Contracts for innovation: DSbD Advancing CHERI Tools and software

Opportunity status:	Closed
Funders:	Innovate UK
Co-funders:	Department for Science, Innovation and Technology
Funding type:	Other
Publication date:	19 May 2025
Opening date:	14 May 2025 9:30am UK time
Closing date:	18 June 2025 11:00am UK time

See the full opportunity details on the Innovation Funding Service.

Organisations can apply for a share of up to £12 million, inclusive of VAT, to work on maturing and enabling the availability of CHERI Tools and Software components for RISC-V embedded devices that implement the CHERI architecture extensions.

$50M

INDUSTRY EMERGING INNOVATION INDUSTRY NEWS

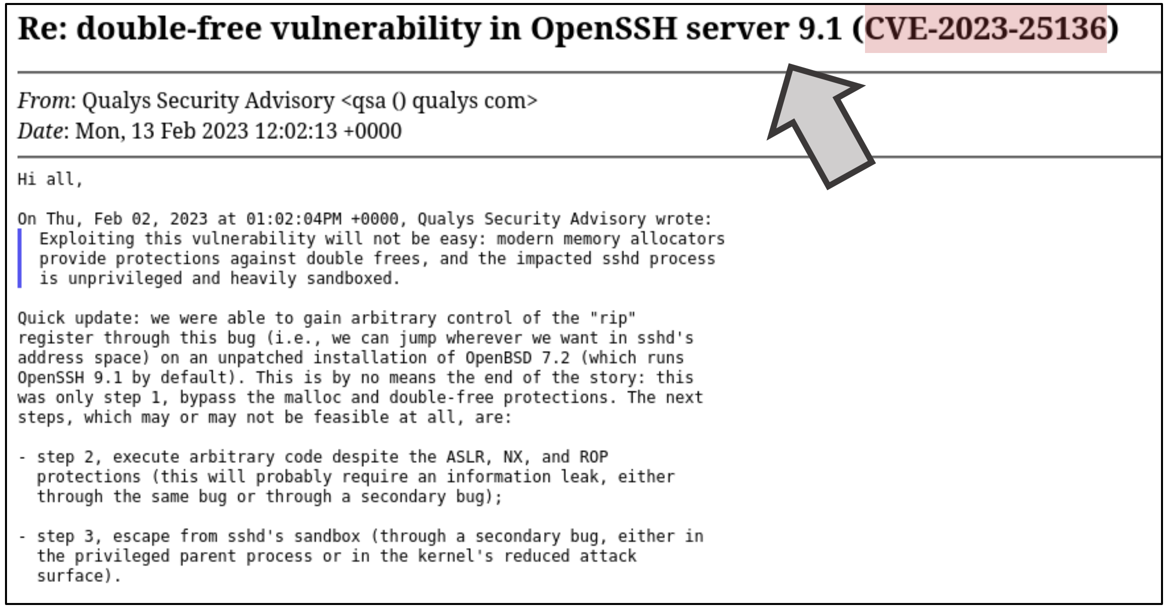
DARPA to Hold Compartmentalization and Privilege Management (CPM) Proposers Day

Full solutions are anticipated to require new hardware support, though software-only solutions that can meet performance expectations are in-scope for the program.

By **Homeland Security Today** April 1, 2023

We know from this experience that **compartmentalization works**

We know from this experience that **compartmentalization works**



Re: double-free vulnerability in OpenSSH server 9.1 (CVE-2023-25136)

From: Qualys Security Advisory <qsa () qualys com>
Date: Mon, 13 Feb 2023 12:02:13 +0000

```
Hi all,

On Thu, Feb 02, 2023 at 01:02:04PM +0000, Qualys Security Advisory wrote:
  Exploiting this vulnerability will not be easy: modern memory allocators
  provide protections against double frees, and the impacted sshd process
  is unprivileged and heavily sandboxed.

Quick update: we were able to gain arbitrary control of the "rip"
register through this bug (i.e., we can jump wherever we want in sshd's
address space) on an unpatched installation of OpenBSD 7.2 (which runs
OpenSSH 9.1 by default). This is by no means the end of the story: this
was only step 1, bypass the malloc and double-free protections. The next
steps, which may or may not be feasible at all, are:

- step 2, execute arbitrary code despite the ASLR, NX, and ROP
  protections (this will probably require an information leak, either
  through the same bug or through a secondary bug);

- step 3, escape from sshd's sandbox (through a secondary bug, either in
  the privileged parent process or in the kernel's reduced attack
  surface).
```

(1)

- OpenSSH is **vulnerable to a bug with remote-code execution**
- It is exploitable: attackers can **break allocator-based mitigations**
- Compartmentalization ultimately **mitigates the exploit** by containing it in an unprivileged process

(1) https://seclists.org/oss-sec/2023/q1/92

We know from this experience that **compartmentalization works**

Re: double-free vulnerability in OpenSSH server 9.1 (CVE-2023-25136) (1)

From: Qualys Security Advisory <qsa () qualys com>
Date: Mon, 13 Feb 2023 12:02:13 +0000

```
Hi all,

On Thu, Feb 02, 2023 at 01:02:04PM +0000, Qualys Security Advisory wrote:
  Exploiting this vulnerability will not be easy: modern memory allocators
  provide protections against double frees, and the impacted sshd process
  is unprivileged and heavily sandboxed.

Quick update: we were able to gain arbitrary control of the "rip"
register through this bug (i.e., we can jump wherever we want in sshd's
address space) on an unpatched installation of OpenBSD 7.2 (which runs
OpenSSH 9.1 by default). This is by no means the end of the story: this
was only step 1, bypass the malloc and double-free protections. The next
steps, which may or may not be feasible at all, are:

- step 2, execute arbitrary code despite the ASLR, NX, and ROP
  protections (this will probably require an information leak, either
  through the same bug or through a secondary bug);

- step 3, escape from sshd's sandbox (through a secondary bug, either in
  the privileged parent process or in the kernel's reduced attack
  surface).
```

- OpenSSH is **vulnerable to a bug with remote-code execution**
- It is exploitable: attackers can **break allocator-based mitigations**
- Compartmentalization ultimately **mitigates the exploit** by containing it in an unprivileged process

(1) `https://seclists.org/oss-sec/2023/q1/92`

We know from this experience that **compartmentalization works**

Re: double-free vulnerability in OpenSSH server 9.1 (CVE-2023-25136) (1)

From: Qualys Security Advisory <qsa () qualys com>
Date: Mon, 13 Feb 2023 12:02:13 +0000

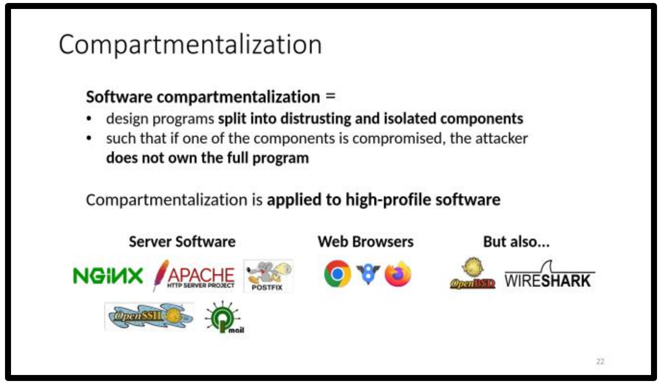
```
Hi all,

On Thu, Feb 02, 2023 at 01:02:04PM +0000, Qualys Security Advisory wrote:
  Exploiting this vulnerability will not be easy: modern memory allocators
  provide protections against double frees, and the impacted sshd process
  is unprivileged and heavily sandboxed.

Quick update: we were able to gain arbitrary control of the "rip"
register through this bug (i.e., we can jump wherever we want in sshd's
address space) on an unpatched installation of OpenBSD 7.2 (which runs
OpenSSH 9.1 by default). This is by no means the end of the story: this
was only step 1, bypass the malloc and double-free protections. The next
steps, which may or may not be feasible at all, are:

- step 2, execute arbitrary code despite the ASLR, NX, and ROP
  protections (this will probably require an information leak, either
  through the same bug or through a secondary bug);

- step 3, escape from sshd's sandbox (through a secondary bug, either in
  the privileged parent process or in the kernel's reduced attack
  surface).
```

- OpenSSH is **vulnerable to a bug with remote-code execution**
- It is exploitable: attackers can **break allocator-based mitigations**
- Compartmentalization ultimately **mitigates the exploit** by containing it in an unprivileged process

(1) `https://seclists.org/oss-sec/2023/q1/92`

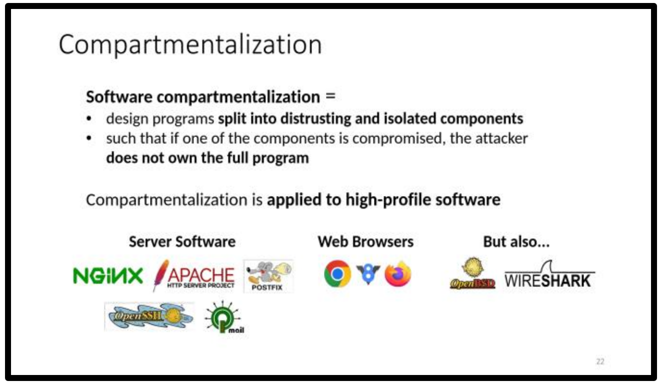
Yet, in 2025, **compartmentalization is not a widespread practice**

Yet, in 2025, **compartmentalization is not a widespread practice**



Beyond the popular software I mentioned earlier, **compartmentalization is rather rare**

Yet, in 2025, **compartmentalization is not a widespread practice**



Beyond the popular software I mentioned earlier, **compartmentalization is rather rare**

(1)

"***Less than 56 apps out of the 1,520 most popular Debian applications*** *are compartmentalized*"

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Why?

Why?

SoK: Software Compartmentalization

Hugo Lefeuvre[†1], Nathan Dautenhahn[‡1], David Chisnall[∞], Pierre Olivier[▽]
[†]*The University of British Columbia,* [‡]*Serenitix,* [∞]*SCI Semiconductor,* [▽]*The University of Manchester*

***Abstract*—Decomposing large systems into smaller components with limited privileges has long been recognized as an effective means to minimize the impact of exploits. Despite historical roots, demonstrated benefits, and a plethora of research efforts in academia and industry, the compartmentalization of software is still not a mainstream practice. This paper investigates why, and how this status quo can be improved. Noting that existing approaches are fraught with inconsistencies in terminology and analytical methods, we propose a unified model for the systematic analysis, comparison, and directing of compartmentalization approaches. We use this model to review 211 research efforts and analyze 61 mainstream compartmentalized systems, confronting them to understand the limitations of both research and production works. Among others, our findings reveal that mainstream efforts largely rely on manual methods, custom abstractions, and legacy mechanisms, poles apart from recent research. We conclude with recommendations: compartmentalization should be solved holistically; progress is needed towards simplifying the definition of compartmentalization policies; towards better challenging our threat models in the light of confused deputies and hardware limitations; as well as towards bridging the gaps we pinpoint between research and mainstream needs. This paper not only maps the historical and current landscape of compartmentalization, but also sets forth a framework to foster their evolution and adoption.**

1. Introduction

Despite decades of effort, vulnerabilities still plague software, and thwarting them remains a game of cat and mouse. The *principle of least privilege* (PoLP) [209] is the last line of defense when protections fail or when flaws are unknown. By granting each entity only the privileges needed, the PoLP ensures that a compromise of one part will not imply that of the whole. *Software compartmentalization* is a prominent implementation of the PoLP, in which developers divide a large program into smaller, lesser privileged components, to reduce the impact of potential security breaches.

Software compartmentalization inherits from a large body of work, starting with processes [93]; including OS models such as microkernels [106, 126, 150, 164, 258], security and separation kernels [56, 58, 205, 206, 247], or capability OSes [83, 125]; all the way to fine-grain application compartmentalization in the 2000s [147, 181, 232, 238]

[1]This work was primarily done while Hugo Lefeuvre was affiliated with the University of Manchester and SCI Semiconductor, and Nathan Dautenhahn with Rice University (with minor contributions at Riverside Research).

following qmail [68], Postfix [121], or OpenSSH [198]. Its promises are plenty: containing memory safety issues [72, 202], untrusted third parties [54, 181], or unsafe parts of safe languages [62, 149, 174, 201, 228], isolating cryptographic secrets [167, 171, 232] or shadow stacks [74], thwarting supply-chain attacks [111, 235] and side-channels [137, 176, 182], or providing fault resilience [164, 184].

Despite longstanding recognition within the academic sphere and proven effectiveness in seminal industry projects, the adoption of compartmentalization techniques in mainstream software remains inconsistent: compartmentalizing is still far from being a common engineering practice. Take as example the isolation of cryptographic secrets, a practice long advocated by the community [65, 70, 73, 87, 115–117, 140, 158, 165, 167, 211, 226, 232] but without viable adoption by leading cryptography libraries. At a time of growing threats, we are missing out on the security benefits compartmentalization can bring. This paper investigates the reasons behind this status quo, and how to improve it.

Research speculated that this situation is due to a lack of automation [65, 253], limitations of mechanisms [190, 259], excessive performance overheads [188, 232], a lack of strong security guarantees [129, 160], among others. All are likely part of the problem, and through decades of research progress was made on every front. Today the community lacks a global overview of this progress. Designing for or retrofitting compartmentalization is often hampered by inconsistencies in the understanding and application of its concepts. Existing models and terminologies are numerous, ad-hoc, and often contradictory, leading to confusion and a growing body of work that cannot be compared. The lack of a systematic perspective leads to a mismatch between what software compartmentalization needs to progress towards the mainstream, and the focus and framing of research efforts: most do not tackle compartmentalization's key aspects as a whole and thus produce solutions that cannot be relevant to making it a mainstream practice.

Recognizing these challenges, we propose a unified model providing a consistent framework for defining, understanding, and implementing compartmentalization. This model comes with a comprehensive taxonomy that allows to classify compartmentalization strategies based on their policy definition methods, abstractions, and mechanisms, providing a basis for systematic evaluation and comparison.

We validate our model and taxonomy by systematizing 211 research and 61 mainstream software systems implementing compartmentalization. Doing so, this study provides unique insights into where mainstream efforts have

(1)

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Systematizing Compartmentalization

Here is how we approached the question:

- Started by making a fundamental observation: experts do not even agree on what compartmentalization is
  - Define a **fundamental model** and **consistent terminology** for compartmentalization
- Propose a **taxonomy** for evaluating compartmentalization approaches and **apply it systematically** to a wide set (200+) of *research* and *mainstream* efforts
- From this: **extract a set of core challenges**

Systematizing Compartmentalization

Here is how we approached the question:

- Started by making a fundamental observation: experts do not even agree on what compartmentalization is
  - Define a **fundamental model** and **consistent terminology** for compartmentalization
- Propose a **taxonomy** for evaluating compartmentalization approaches and **apply it systematically** to a wide set (200+) of *research* and *mainstream* efforts
- From this: **extract a set of core challenges**

Systematizing Compartmentalization

Here is how we approached the question:

- Started by making a fundamental observation: experts do not even agree on what compartmentalization is
  - Define a **fundamental model** and **consistent terminology** for compartmentalization
- Propose a **taxonomy** for evaluating compartmentalization approaches and **apply it systematically** to a wide set (200+) of *research* and *mainstream* efforts
- From this: **extract a set of core challenges**

Systematizing Compartmentalization

Here is how we approached the question:

- Started by making a fundamental observation: experts do not even agree on what compartmentalization is
  - Define a **fundamental model** and **consistent terminology** for compartmentalization
- Propose a **taxonomy** for evaluating compartmentalization approaches and **apply it systematically** to a wide set (200+) of *research* and *mainstream* efforts
- From this: **extract a set of core challenges**

My promise for this talk

A journey through twenty years of compartmentalization

1. What is software compartmentalization? (slightly more formal)
2. A systematic perspective on compartmentalization
3. The *why:* compartmentalization everywhere, what will it take?

My promise for this talk

A journey through twenty years of compartmentalization

1. What is software compartmentalization? (slightly more formal)
2. A systematic perspective on compartmentalization
3. The *why:* compartmentalization everywhere, what will it take?

What is Software Compartmentalization?

Software compartmentalization is...

- a **software engineering practice**

What is Software Compartmentalization?

Software compartmentalization is...

- a **software engineering practice** where developers **break down a program into groups of isolated/distrusting components**

What is Software Compartmentalization?

Software compartmentalization is...

- a **software engineering practice** where developers **break down a program into groups of isolated/distrusting components**, each controlling only the resources they precisely require

What is Software Compartmentalization?

Software compartmentalization is...

- a **software engineering practice** where developers **break down a program into groups of isolated/distrusting components**, each controlling only the resources they precisely require
- an embodiment of the **principle of least-privilege**

What is Software Compartmentalization?

Software compartmentalization is...

- a **software engineering practice** where developers **break down a program into groups of isolated/distrusting components**, each controlling only the resources they precisely require
- an embodiment of the **principle of least-privilege**

Fundamental idea:

- if a component is compromised, attackers are **restricted to the permissions of the compromised component**

What is Software Compartmentalization?

Software compartmentalization is...

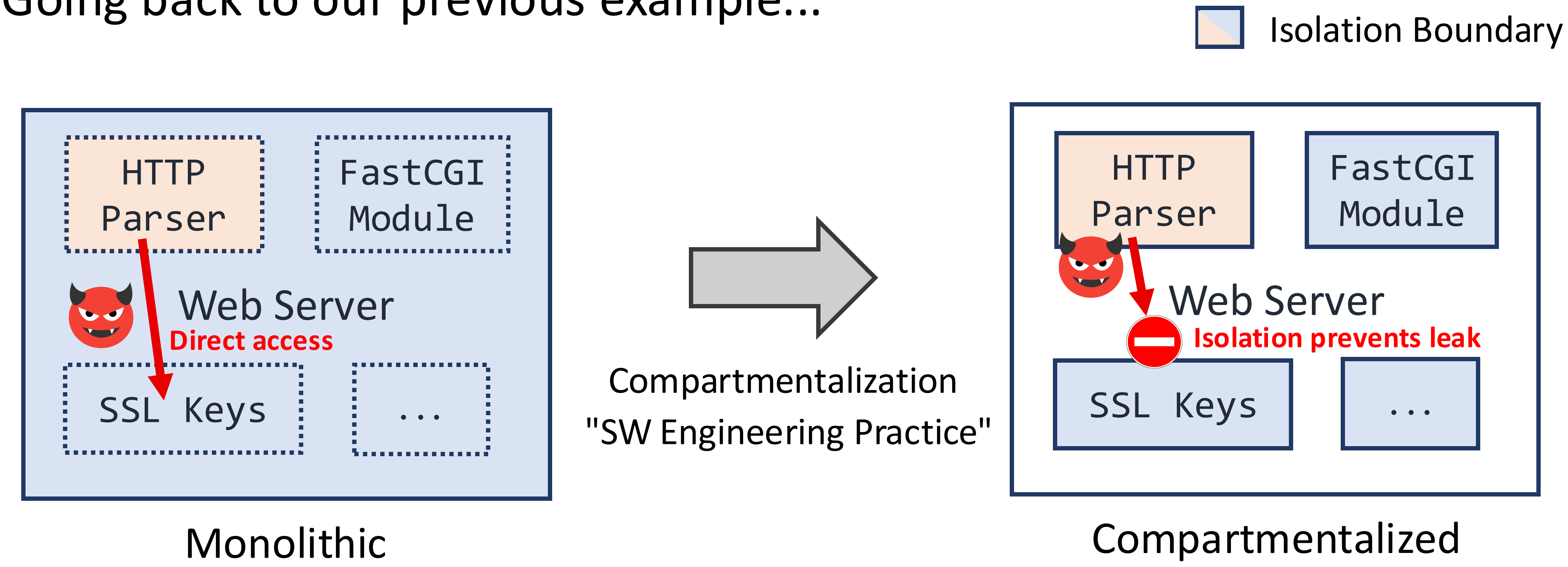
- a **software engineering practice** where developers **break down a program into groups of isolated/distrusting components**, each controlling only the resources they precisely require
- an embodiment of the **principle of least-privilege**

Fundamental idea:

- if a component is compromised, attackers are **restricted to the permissions of the compromised component**
- to escalate privileges and compromise the rest of the system, attackers must **find and exploit additional software vulnerabilities**

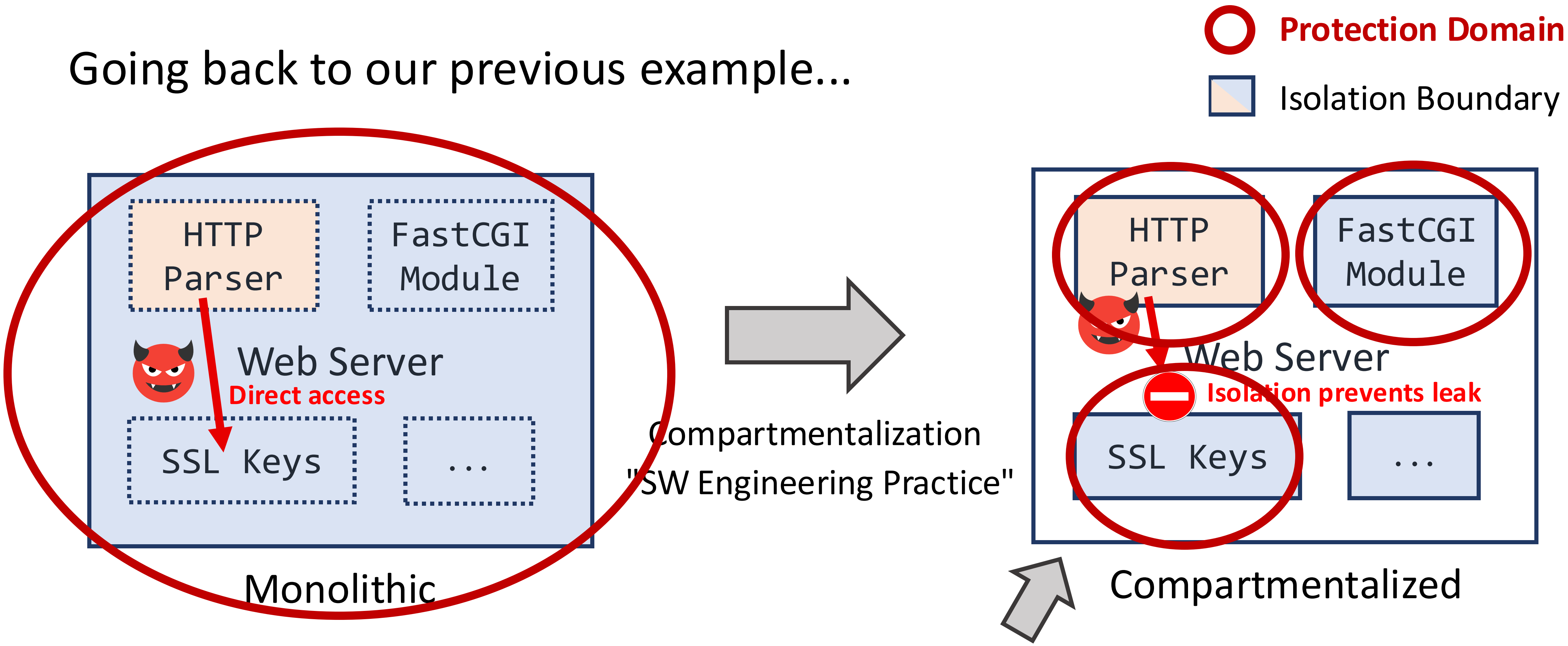
What is Software Compartmentalization?

Going back to our previous example...



What is Software Compartmentalization?

Going back to our previous example...

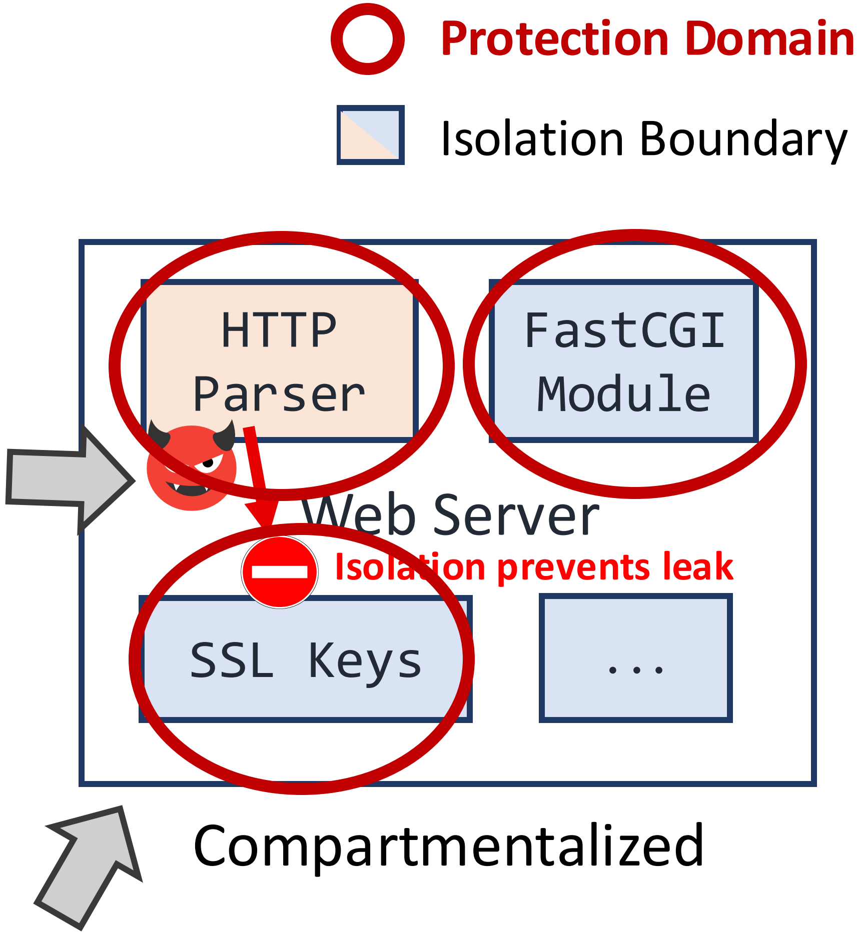


Components now **distrust each other**

What is Software Compartmentalization?

Going back to our previous example...

Compartmentalizing **adds a** (potentially very tricky) **step on the attacker's path** towards taking over the program



Components now **distrust each other**

What is Software Compartmentalization?

Software compartmentalization is applicable to any kind of program

Web Browsers

Server Software

qmail

NGINX

APACHE
HTTP SERVER PROJECT

OpenSSH

Microkernels

seL4
Security. Performance. Proof.

Fuchsia

What is Software Compartmentalization?

Software compartmentalization is applicable to any kind of program

Web Browsers

Server Software

qmail

NGINX

APACHE
HTTP SERVER PROJECT

OpenSSH

Microkernels

seL4
Security. Performance. Proof.

Fuchsia

Applications

What is Software Compartmentalization?

Software compartmentalization is applicable to any kind of program

Web Browsers

Server Software

Microkernels

OS kernels

What is Software Compartmentalization?

Software compartmentalization is applicable to any kind of program

Web Browsers

Server Software

qmail

NGINX

APACHE HTTP SERVER PROJECT

OpenSSH

Microkernels

seL4 Security. Performance. Proof.

Fuchsia

But also: hypervisors, firmware, etc.

What is Software Compartmentalization?

Software compartmentalization can be **applied to new as well as existing programs**:

- When applied to existing programs, we talk about **retrofitting**

What is Software Compartmentalization?

Software compartmentalization can be **applied to new as well as existing programs**:

- When applied to existing programs, we talk about **retrofitting**

Some programs which included compartmentalization in their design in the first place (non-exhaustive!):

seL4

qmail*

*Mail transfer agent, one of the first compartmentalized programs to the best of my knowledge

What is Software Compartmentalization?

Software compartmentalization can be **applied to new as well as existing programs**:

- When applied to existing programs, we talk about **retrofitting**

Some programs which included compartmentalization in their design in the first place (non-exhaustive!):

qmail*

...and some where it was retrofitted:

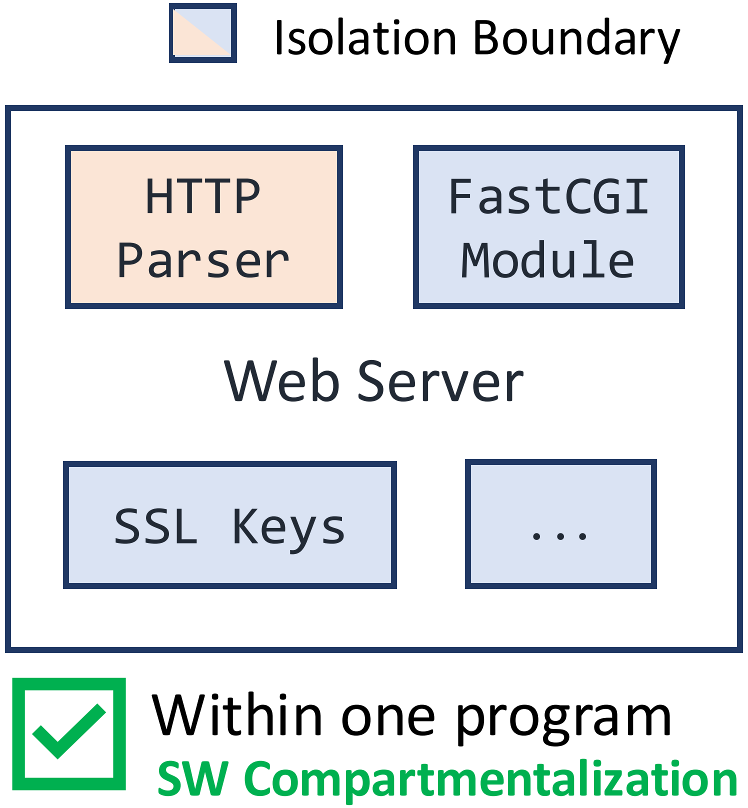
*Mail transfer agent, one of the first compartmentalized programs to the best of my knowledge

What is Software Compartmentalization?

Still, software compartmentalization as we refer to here applies **within one program**, **not across programs**

What is Software Compartmentalization?

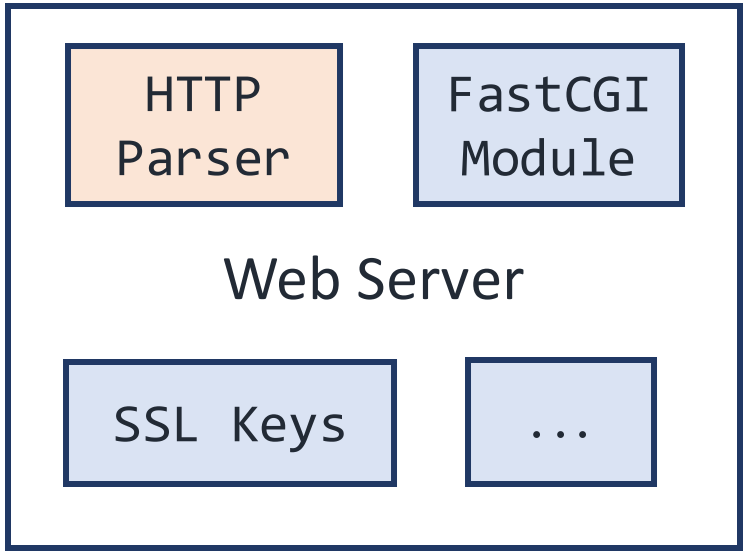
Still, software compartmentalization as we refer to here applies **within one program, not across programs**



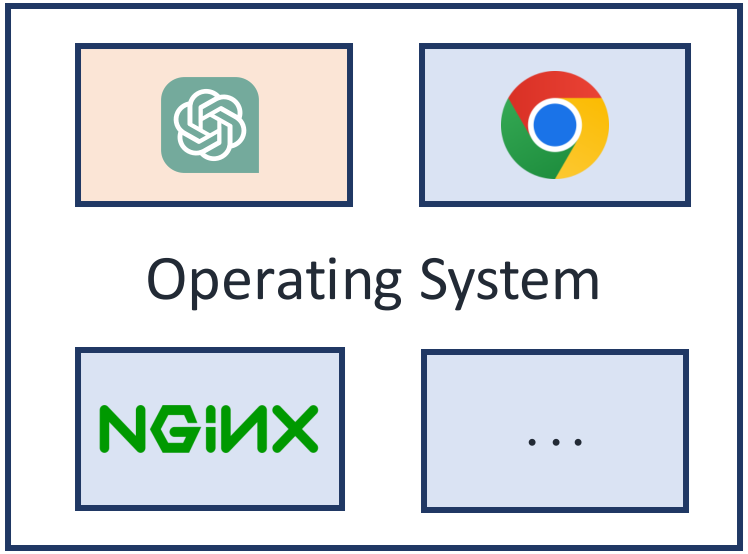
What is Software Compartmentalization?

Still, software compartmentalization as we refer to here applies **within one program, not across programs**

Isolation Boundary



Within one program
SW Compartmentalization

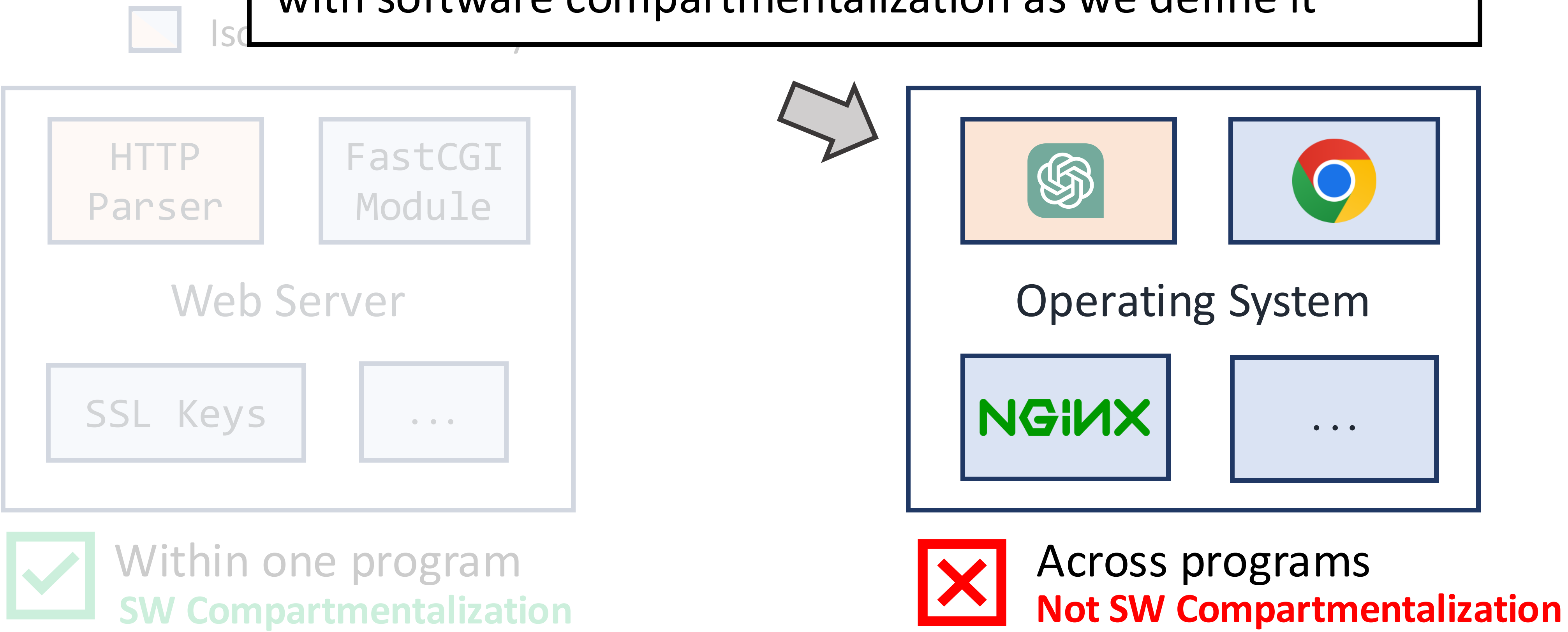


Across programs
Not SW Compartmentalization

What is Software Compartmentalization?

Still, software compartmentalization as we refer to here applies **within one progran**

More general isolation: share challenges and solutions with software compartmentalization as we define it

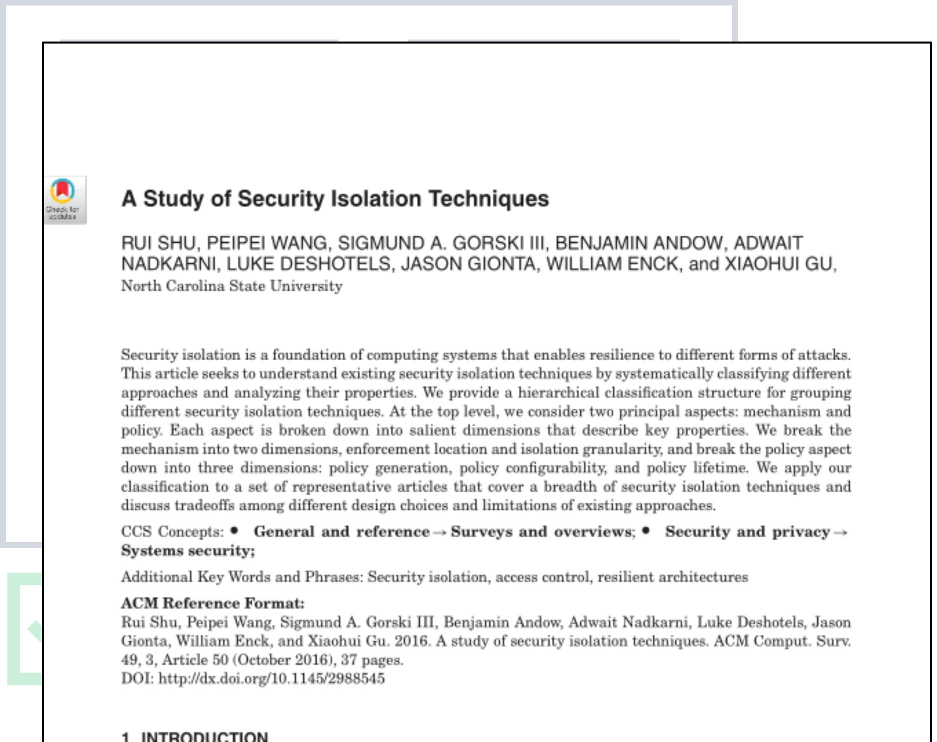


(1) Shu et al., A Study of Security Isolation Techniques, CSUR 2016

What is Software Compartmentalization?

Still, software compartmentalization as we refer to here applies **within one progran**

More general isolation**:** share challenges and solutions with software compartmentalization as we define it


A Study of Security Isolation Techniques

RUI SHU, PEIPEI WANG, SIGMUND A. GORSKI III, BENJAMIN ANDOW, ADWAIT NADKARNI, LUKE DESHOTELS, JASON GIONTA, WILLIAM ENCK, and XIAOHUI GU, North Carolina State University

Security isolation is a foundation of computing systems that enables resilience to different forms of attacks. This article seeks to understand existing security isolation techniques by systematically classifying different approaches and analyzing their properties. We provide a hierarchical classification structure for grouping different security isolation techniques. At the top level, we consider two principal aspects: mechanism and policy. Each aspect is broken down into salient dimensions that describe key properties. We break the mechanism into two dimensions, enforcement location and isolation granularity, and break the policy aspect down into three dimensions: policy generation, policy configurability, and policy lifetime. We apply our classification to a set of representative articles that cover a breadth of security isolation techniques and discuss tradeoffs among different design choices and limitations of existing approaches.

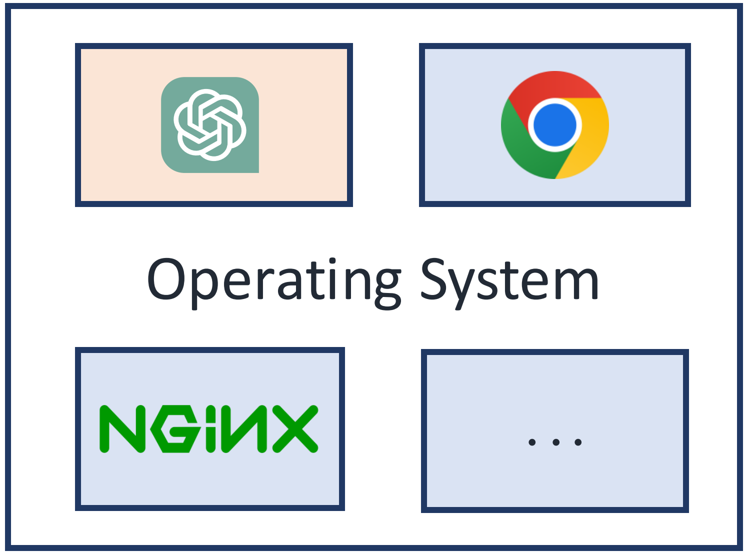
CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Security and privacy** → **Systems security**;

Additional Key Words and Phrases: Security isolation, access control, resilient architectures

ACM Reference Format:
Rui Shu, Peipei Wang, Sigmund A. Gorski III, Benjamin Andow, Adwait Nadkarni, Luke Deshotels, Jason Gionta, William Enck, and Xiaohui Gu. 2016. A study of security isolation techniques. ACM Comput. Surv. 49, 3, Article 50 (October 2016), 37 pages.
DOI: http://dx.doi.org/10.1145/2988545

1 INTRODUCTION

(1)



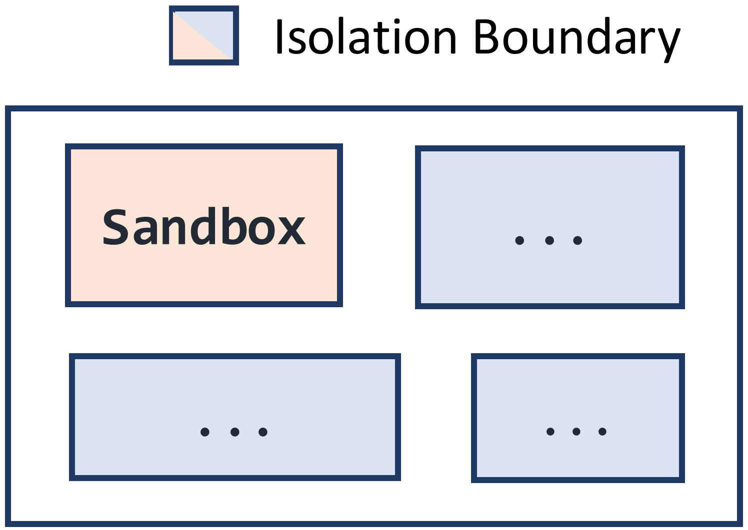
Across programs
Not SW Compartmentalization

What is Software Compartmentalization?

Software compartmentalization can target **different trust models**

What is Software Compartmentalization?

Software compartmentalization can target **different trust models**

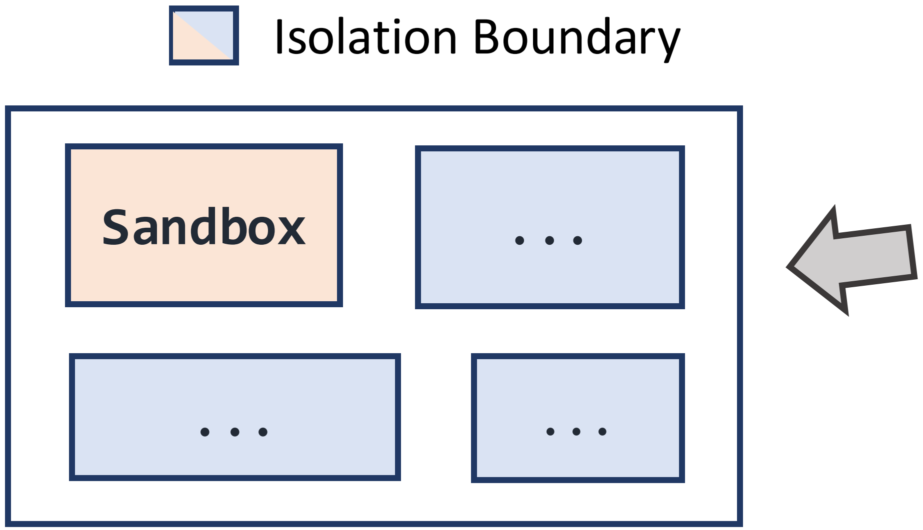


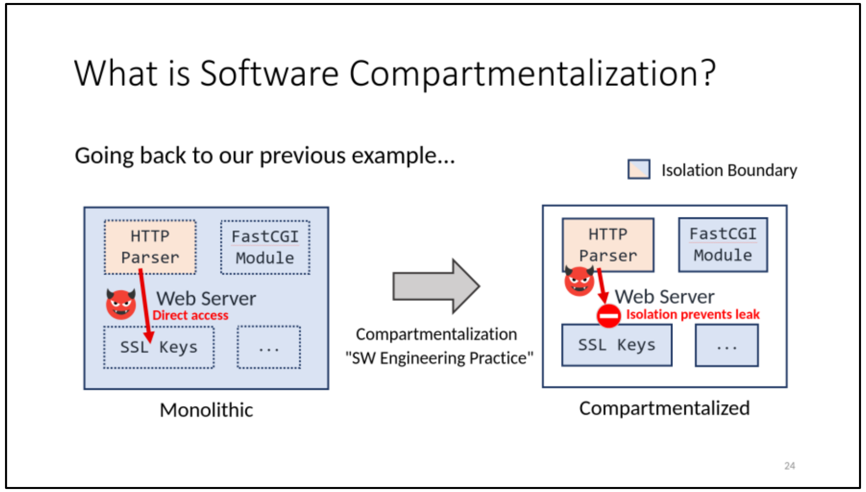
Sandbox: component isolated to protect the rest of the system

All but the sandbox is trusted

What is Software Compartmentalization?

Software compartmentalization can target **different trust models**



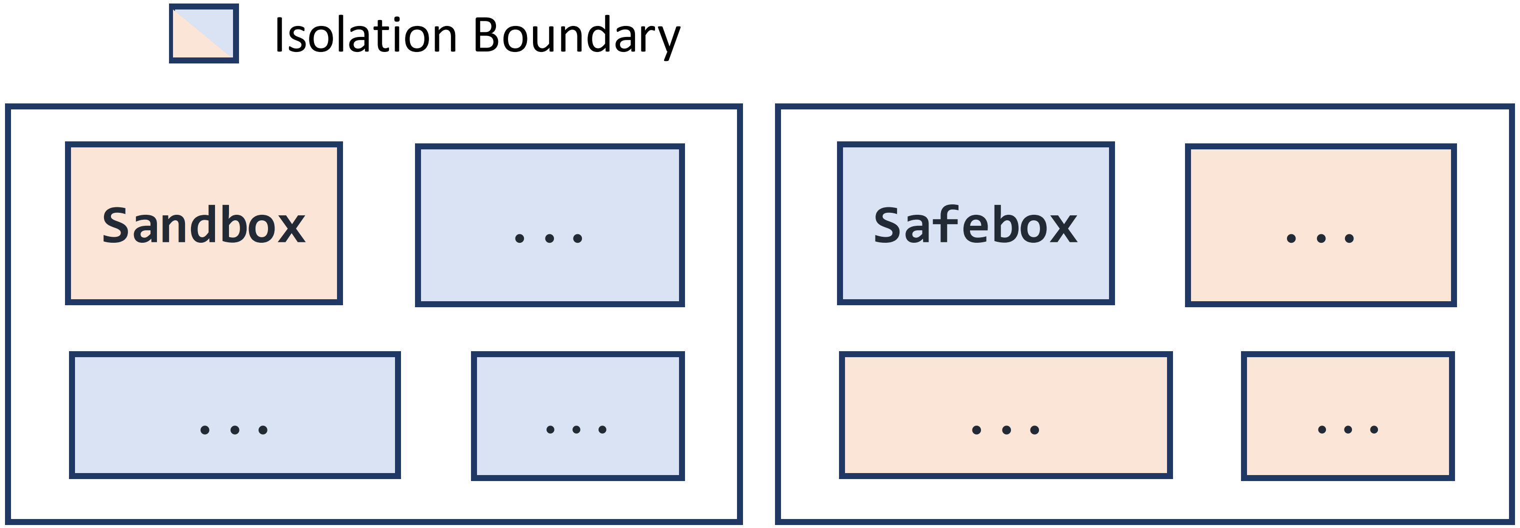


Sandbox: component isolated to protect the rest of the system

All but the sandbox is trusted

What is Software Compartmentalization?

Software compartmentalization can target **different trust models**



Sandbox: component isolated to protect the rest of the system

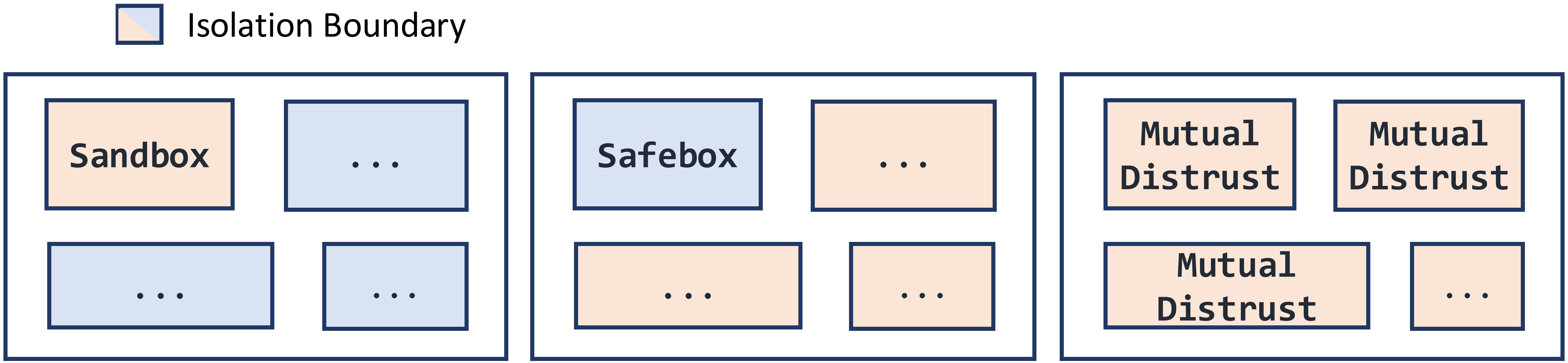
All but the sandbox is trusted

Safebox: component isolated to protect it from others

Only the safebox is trusted

What is Software Compartmentalization?

Software compartmentalization can target **different trust models**



Sandbox: component isolated to protect the rest of the system

All but the sandbox is trusted

Safebox: component isolated to protect it from others

Only the safebox is trusted

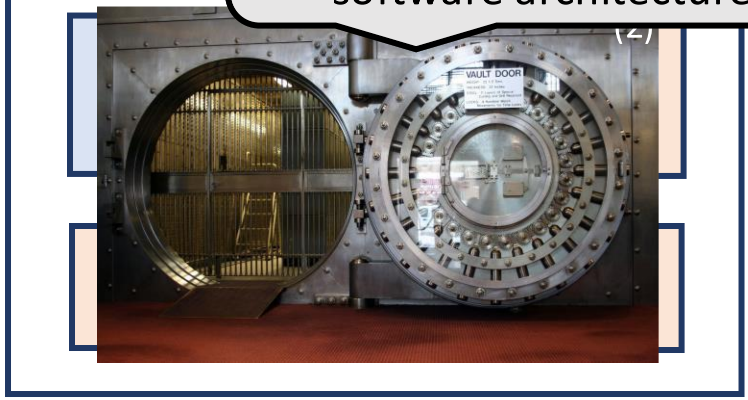
Mutual distrust: components distrust each other

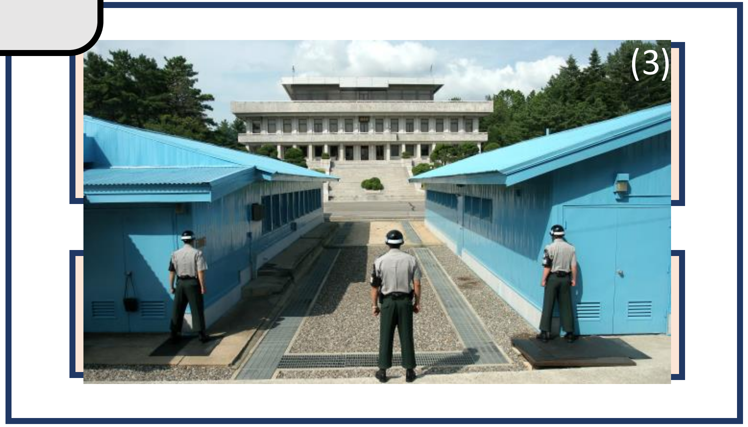
What is Software Compartmentalization?

Software compartmentalization can target **different trust models**

Isolation Boundary

These trust models are distinct and result in different software architectures





Sandbox: component isolated to protect the rest of the system

All but the sandbox is trusted

Safebox: component isolated to protect it from others

Only the safebox is trusted

Mutual distrust: components distrust each other

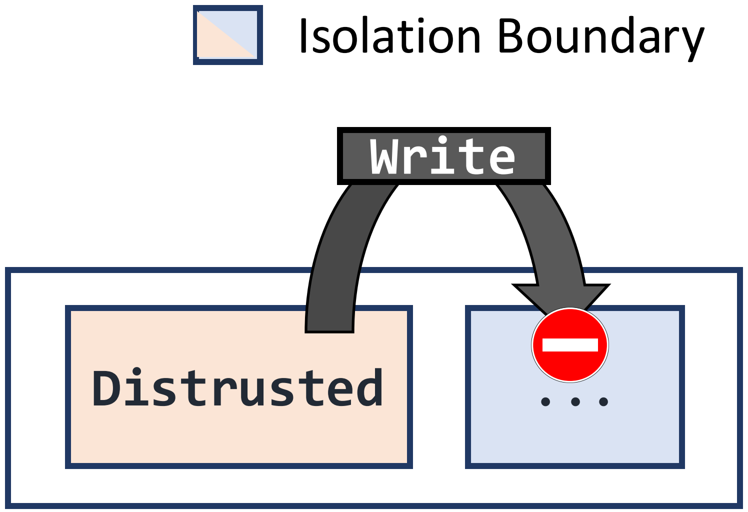
(1) https://fr.wikipedia.org/wiki/Panoptique#/media/Fichier:Kilmainham_1.JPG (2) https://en.wikipedia.org/wiki/Bank_vault#/media/File:WinonaSavingsBankVault.JPG (3) https://en.wikipedia.org/wiki/File:Panmunjeom_DMZ.png

What is Software Compartmentalization?

Software compartmentalization can target **different properties**

What is Software Compartmentalization?

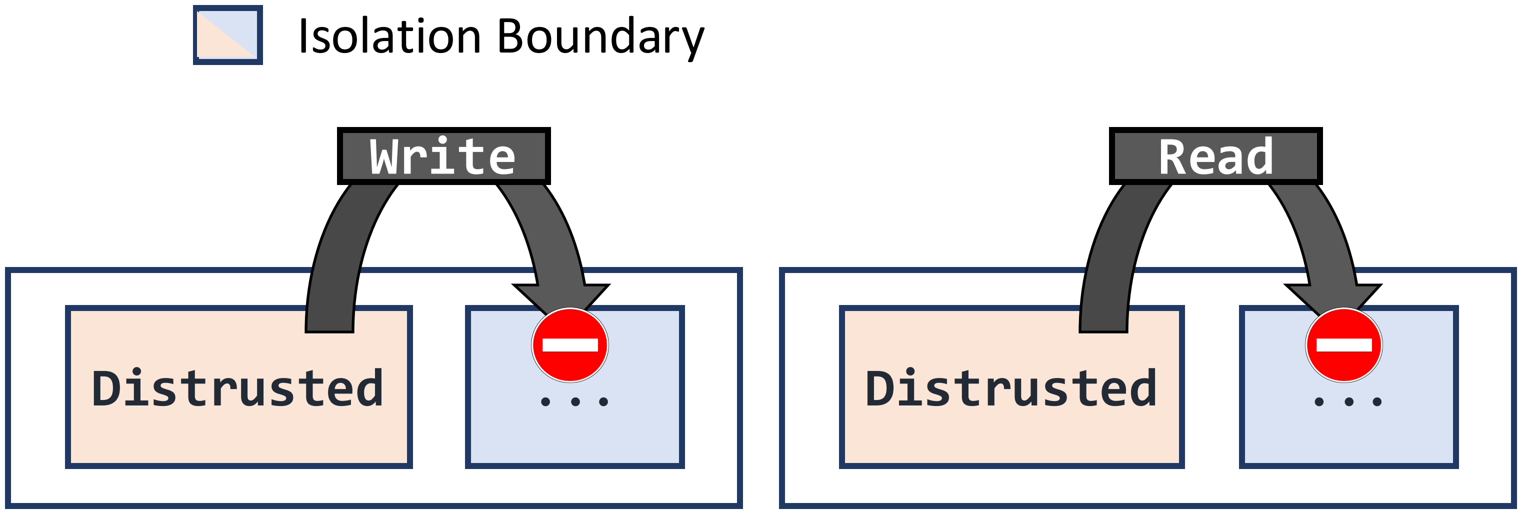
Software compartmentalization can target **different properties**



Integrity: compartment cannot alter other compartment's data

What is Software Compartmentalization?

Software compartmentalization can target **different properties**

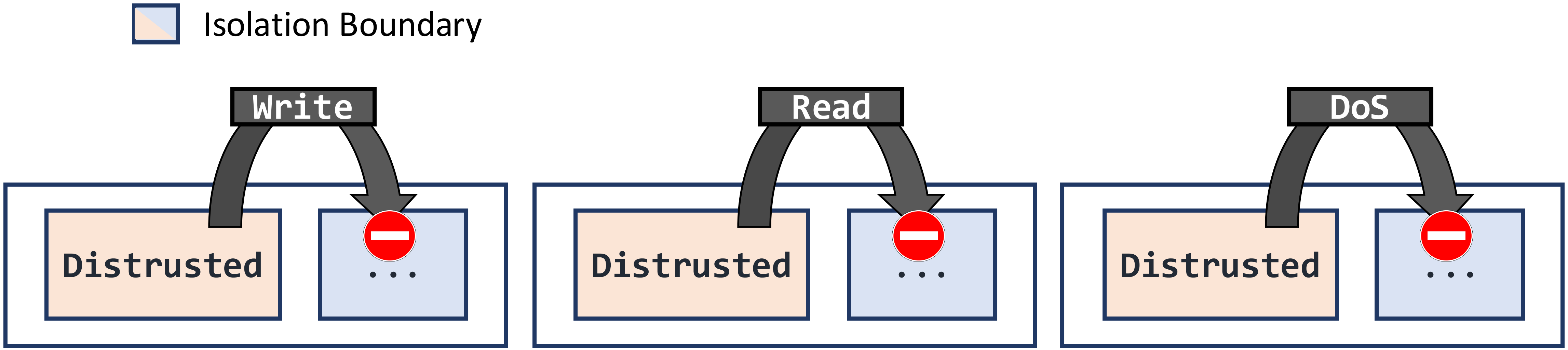


Integrity: compartment cannot alter other compartment's data

Confidentiality: compartment cannot read other compartment's data

What is Software Compartmentalization?

Software compartmentalization can target **different properties**



Integrity: compartment cannot alter other compartment's data

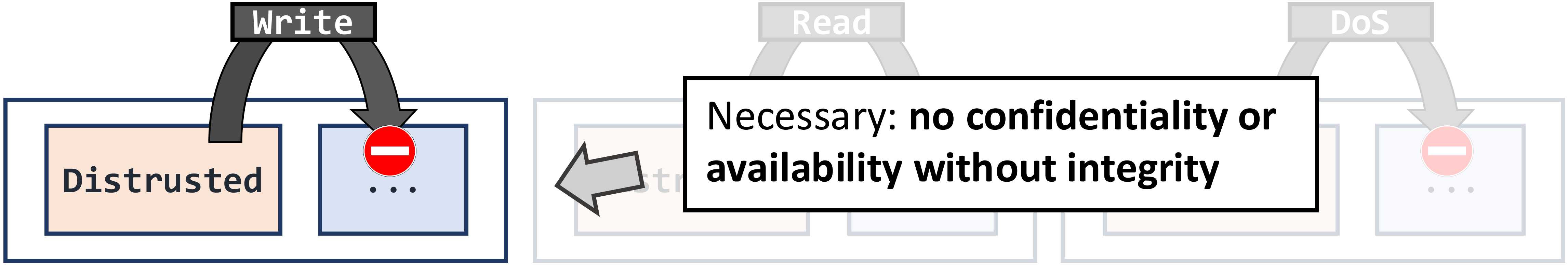
Confidentiality: compartment cannot read other compartment's data

Availability: compartment cannot prevent other compartments from operating

What is Software Compartmentalization?

Software compartmentalization can target **different properties**

Isolation Boundary



Integrity: compartment cannot alter other compartment's data

Confidentiality: compartment cannot read other compartment's data

Availability: compartment cannot prevent other compartments from operating

My promise for this talk

A journey through twenty years of compartmentalization

1. ~~What is software compartmentalization? (slightly more formal)~~
2. A systematic perspective on compartmentalization
3. The *why:* compartmentalization everywhere, what will it take?

Take a step back: How do practitioners compartmentalize? What does research say? What are open challenges?

Bigger Picture of Compartmentalization

Propose to view compartmentalization as 3 problems:

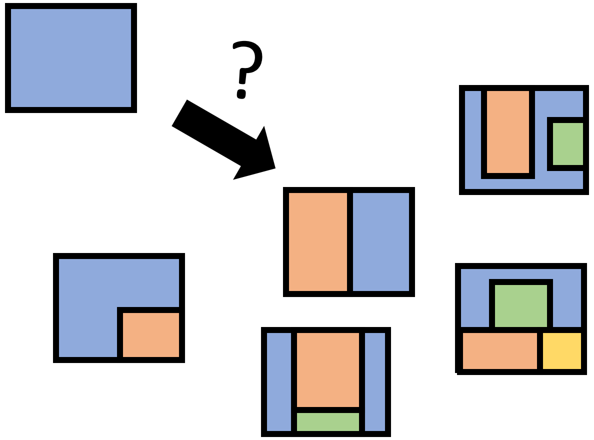
1. **How to determine the right policy to enforce?**
   - Done with a p*olicy definition method*

2. **How to integrate the notion of compartmentalization policies in software / programming models / idioms?**
   - Done with a c*ompartmentalization abstraction*

3. **How to enforce policies at runtime?**
   - Done with a c*ompartmentalization mechanism*

Bigger Picture of Compartmentalization

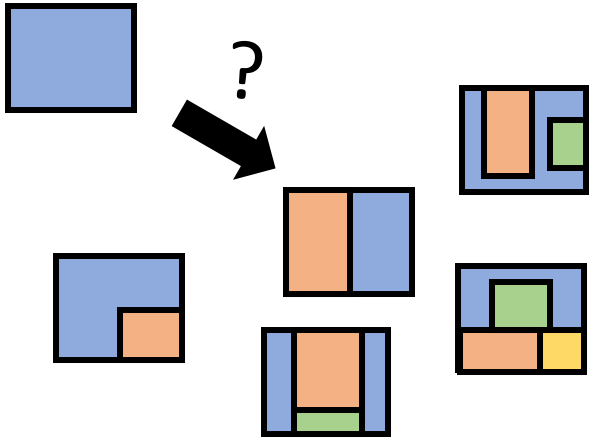
Propose to view compartmentalization as 3 problems:



1. **How to determine the right policy to enforce?**
   - Done with a *policy definition method*
2. **How to integrate the notion of compartmentalization policies in software / programming models / idioms?**
   - Done with a *compartmentalization abstraction*
3. **How to enforce policies at runtime?**
   - Done with a *compartmentalization mechanism*

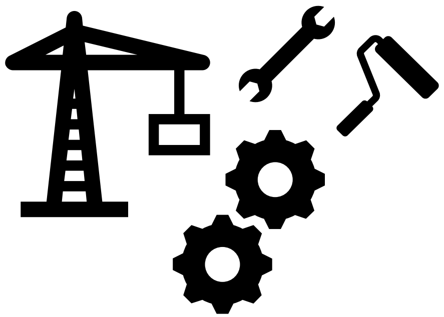
Bigger Picture of Compartmentalization

Propose to view compartmentalization as 3 problems:



1. **How to determine the right policy to enforce?**
   - Done with a *policy definition method*

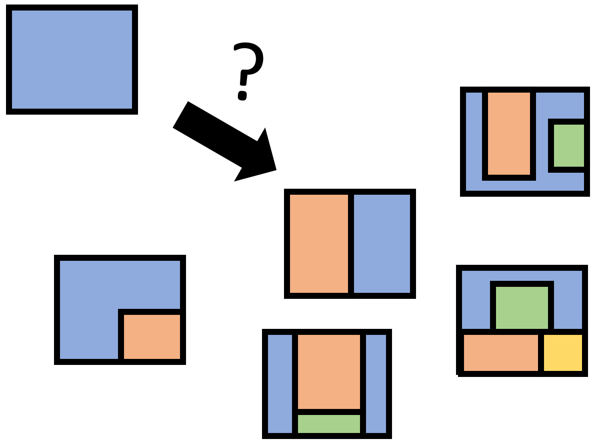
2. **How to integrate the notion of compartmentalization policies in software / programming models / idioms?**
   - Done with a *compartmentalization abstraction*



3. **How to enforce policies at runtime?**
   - Done with a *compartmentalization mechanism*

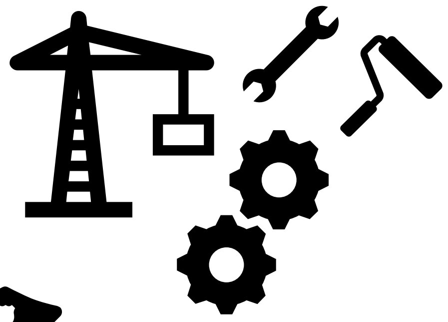
Bigger Picture of Compartmentalization

Propose to view compartmentalization as 3 problems:



1. **How to determine the right policy to enforce?**
   - Done with a *policy definition method*

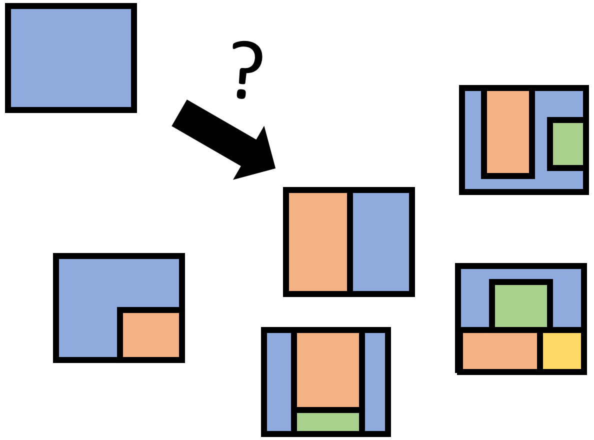
2. **How to integrate the notion of compartmentalization policies in software / programming models / idioms?**
   - Done with a *compartmentalization abstraction*



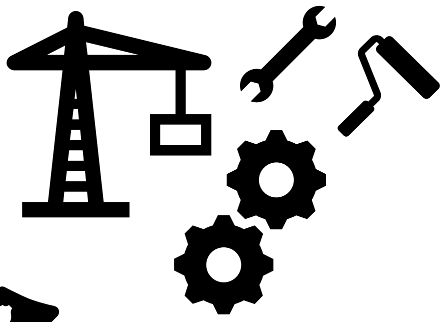
3. **How to enforce policies at runtime?**
   - Done with a *compartmentalization mechanism*

Bigger Picture of Compartmentalization

Propose to view compartmentalization as 3 problems:



1. **How to determine the right policy to enforce?**
   - Done with a *policy definition method*



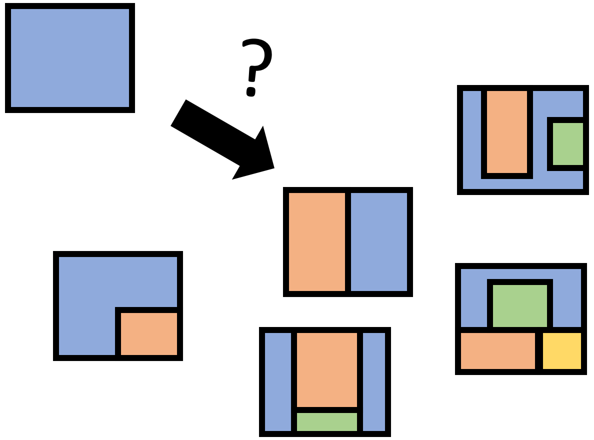
2. **How to integrate the notion of compartmentalization policies in software / programming models / idioms?**
   - Done with a *compartmentalization abstraction*

3. **How to enforce policies at runtime?**
   - Done with a *compartmentalization mechanism*

Problem #1

How to determine the right policy to enforce?

Developers must define **which components are to be separated** and **which properties should be enforced** on the resulting compartments

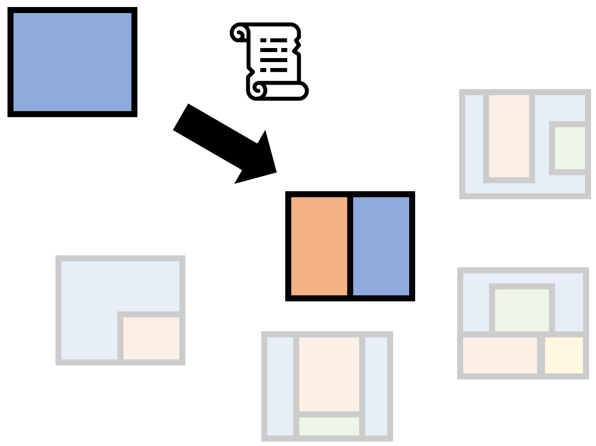


Problem #1

How to determine the right policy to enforce?

Developers must define **which components are to be separated** and **which properties should be enforced** on the resulting compartments

- The result is a **compartmentalization policy**



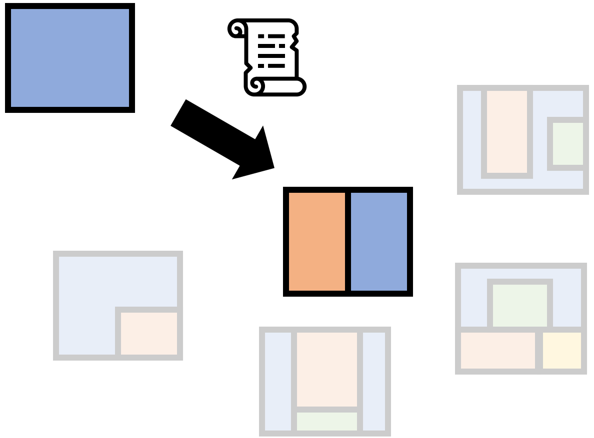
Problem #1

How to determine the right policy to enforce?

Developers must define **which components are to be separated** and **which properties should be enforced** on the resulting compartments

- The result is a **compartmentalization policy**

To design policies, developers employ a **Policy Definition Method**



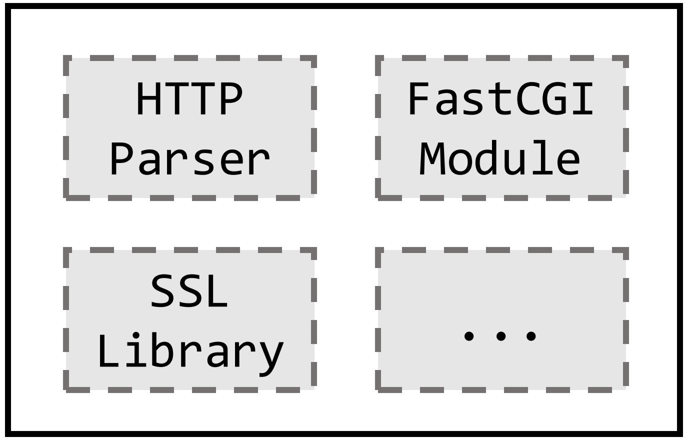
Policy Definition Methods

Historically, people have done this **manually**.

Policy Definition Methods

Historically, people have done this **manually**.

Example: *we want to split this C program.*

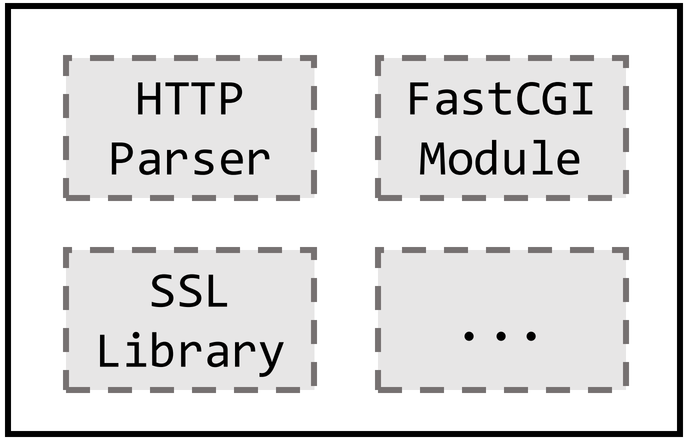


(Fictive monolithic program)

Policy Definition Methods

Historically, people have done this **manually**.

Example: *we want to split this C program.*



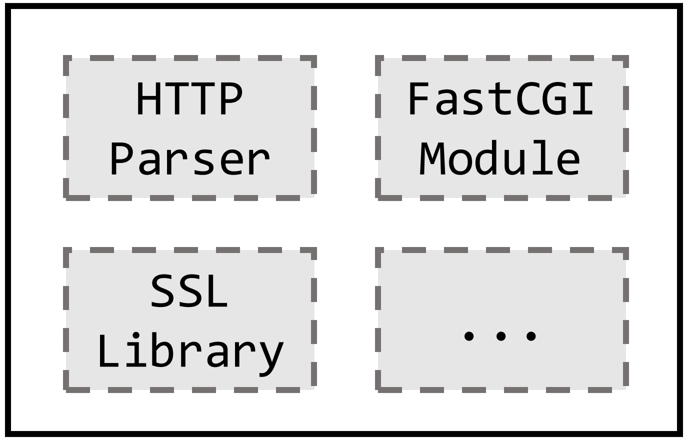
(Fictive monolithic program)

A developer looks attentively at the program...

Policy Definition Methods

Historically, people have done this **manually**.

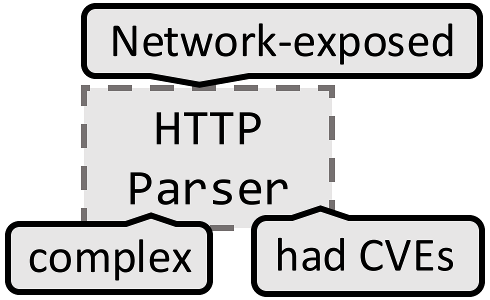
Example: w*e want to split this C program.*



(Fictive monolithic program)

A developer looks attentively at the program...

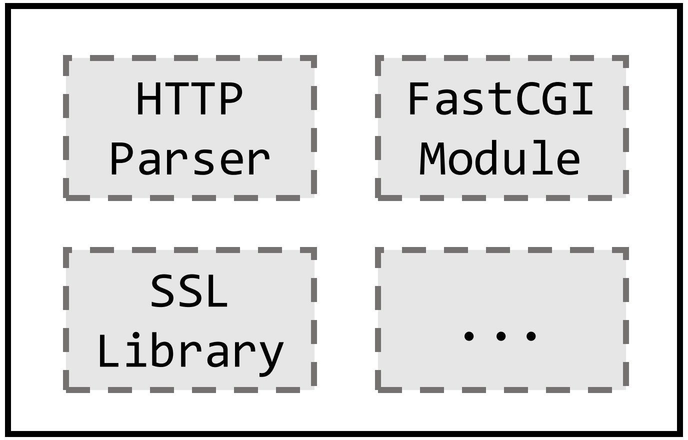
"I do not trust this component"



Policy Definition Methods

Historically, people have done this **manually**.

Example: w*e want to split this C program.*



(Fictive monolithic program)

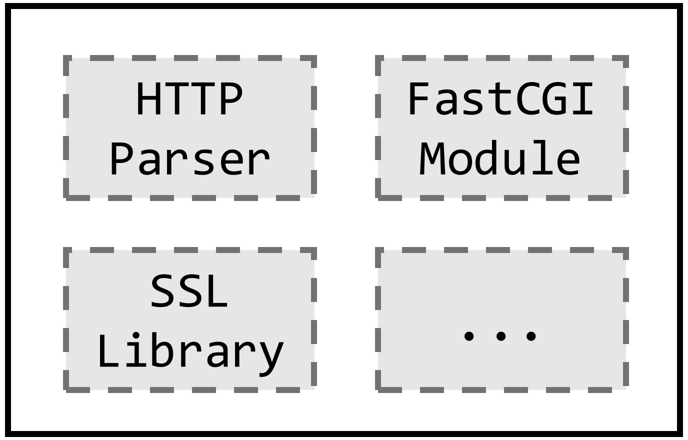
A developer looks attentively at the program...

"I do not trust this component"

Policy Definition Methods

Historically, people have done this **manually**.

Example: *we want to split this C program.*

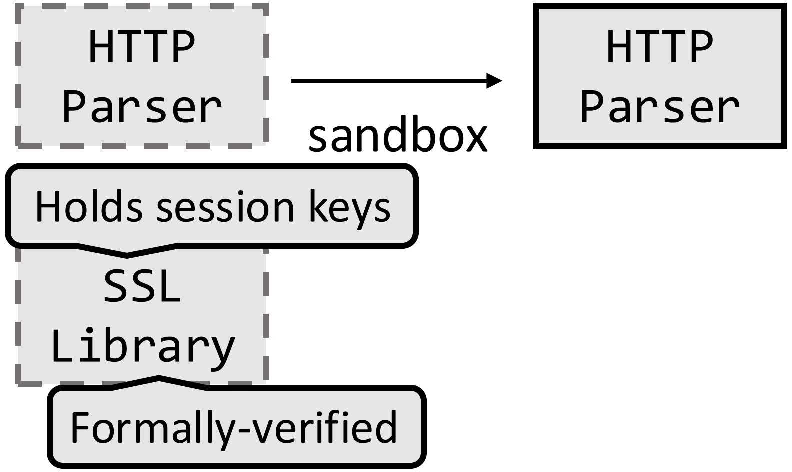


(Fictive monolithic program)

A developer looks attentively at the program...

"I do not trust this component"

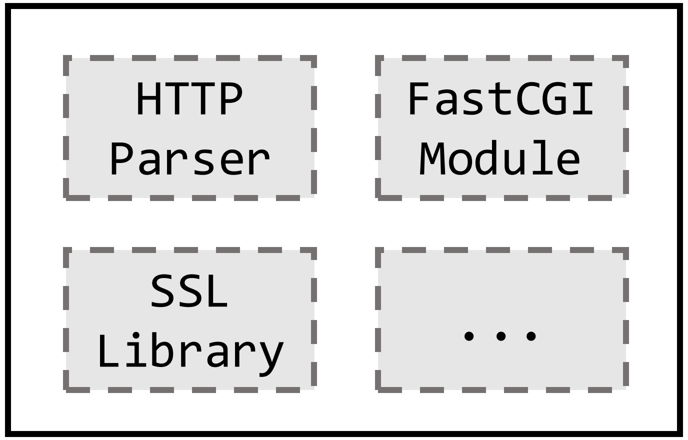
"I particularly value this component"



Policy Definition Methods

Historically, people have done this **manually**.

Example: *we want to split this C program.*

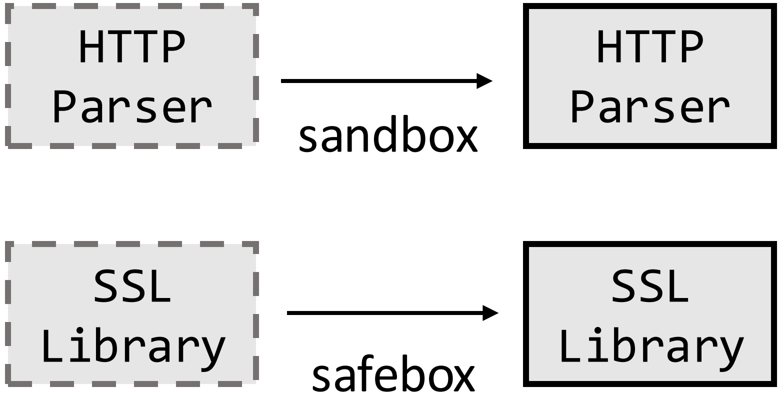


(Fictive monolithic program)

A developer looks attentively at the program...

"I do not trust this component"

"I particularly value this component"



This is still the most common way to do it today.

Systematizing Policy Definition Methods

Policy Definition Methods are a very active research area, with goals such as...

Minimize developer effort
Maximize soundness
Minimize over-privilege
Maximize performance
Maximize security of interfaces

We systematized these works in our SoK

TABLE 1: *Taxonomy of Policy Definition Methods.* PDMs that also propose an abstraction are marked with *. Manual (○) and fully automated (●) policies do not leverage a policy language, thus the column features Not/Applicable.

Separation Method	Automation ○◔◓●[1]	Policy Language Type: Annotations	Policy Language Type: Placement Rules	Separation Granularity	Analysis Approach	Subject Selection	Language Specific	Additional Goals of Automation: Performance	Additional Goals of Automation: Interface Safety
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

(1)

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Systematizing Policy Definition Methods

Policy Definition Methods are a very active research area, with goals such as...

Minimize developer effort

Maximize soundness

Minimize over-privilege

Maximize performance

Maximize security of interfaces

We systematized these works in our SoK

TABLE 1: *Taxonomy of Policy Definition Methods.* PDMs that also propose an abstraction are marked with ∗. Manual (○) and fully automated (●) policies do not leverage a policy language, thus the column features Not/Applicable.

Separation Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	µLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	µLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
µSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

(1)

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Systematizing Policy Definition Methods

Policy definition methods table from the paper

TABLE 1: *Taxonomy of Policy Definition Methods.* PDMs that also propose an abstraction are marked with ∗. Manual (○) and fully automated (●) policies do not leverage a policy language, thus the column features Not/Applicable.

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs∗ [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS∗ [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI∗ [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS∗ [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox∗ [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP∗ [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage∗ [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit∗ [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans∗ [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring∗ [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds∗ [87], CAPACITY∗ [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield∗ [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift∗ [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif∗ [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit∗ [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali∗ [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS∗ [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure∗ [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp∗ [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist∗ [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES∗ [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Names of the policy definition methods we consider

Characteristics we included in the taxonomy

Policy Definition Method	Automation ○◔◓●[1]	Policy Language Type: Annotations	Policy Language Type: Placement Rules	Separation Granularity	Analysis Approach	Subject Selection	Language Specific	Additional Goals of Automation: Performance	Additional Goals of Automation: Interface Safety
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Names of the policy definition methods we consider

Characteristics we included in the taxonomy

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Names of the policy definition methods we consider

Characteristics we included in the taxonomy

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

This talk: present a few interesting aspects, many more in the paper 🙂

Automation is a key research topic in PDMs

Entirely manual

Policy Definition Method	*Automation* ○ ◔ ◓ ●[1]	*Policy Language Type* *Annotations*	*Placement Rules*	*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation* *Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Fully automated

Automation is a key research topic in PDMs

Entirely manual

Policy Definition Method	Automation ○ ◔ ◓ ●[1]	Policy Language Type: Annotations	Policy Language Type: Placement Rules	Separation Granularity	Analysis Approach	Subject Selection	Language Specific	Additional Goals of Automation: Performance	Additional Goals of Automation: Interface Safety
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	[illegible]	[illegible]	[illegible]	[illegible]ynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manu[illegible]er-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Fully automated

They achieve automation leveraging **different kinds of inputs from developers**.

Automation is a key research topic in PDMs

Entirely manual

Policy Definition Method	Automation ○ ◔ ◓ ● [1]	Policy Language Type: Annotations	Policy Language Type: Placement Rules	Separation Granularity	Analysis Approach	Subject Selection	Language Specific	Additional Goals of Automation: Performance	Additional Goals of Automation: Interface Safety
Manual [128], [. . .]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	[illegible]	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	[illegible]	[illegible]	[illegible]	[illegible]	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full autom[illegible] dynamic analysis possible [90].

Fully automated

They employ **different kinds of analysis techniques** (static, dynamic, hybrid)

Automation is a key research topic in PDMs

Specializing on a particular programming language is often necessary to simplify the problem

Entirely manual

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type* *Annotations*	*Placement Rules*	*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation* *Performance*	*Interface Safety*
Manual [128], [. . .]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Fully automated

(1) Liu et al., PtrSplit: Supporting General Pointers in Automatic Program Partitioning, CCS 2017

Pick one influential example from the literature: **PtrSplit**(1)

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]	◔	●	●	μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]	◔	●	○	Function	Static	Any	●	○	●
SOAAP* [117]	◔	●	○	Any	Hybrid	Any	●	●	●
SeCage* [171]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Pick one influential example from the literature: **PtrSplit**(1)

Policy Definition Method	*Automation* ○ ◔ ◓ ●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]				μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]				Function	Static	Any	●	○	●
SOAAP* [117]				Any	Hybrid	Any	●	●	●
SeCage* [171]				Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓	●	○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [165]	◓	●	○	Function	Static	*Code*-centric	●	○	●
Shreds* [87], CAPACITY* [105]	◓	●	○	Any	Static	*Code*-centric	○	○	●
DataShield* [77]	◓	●	○	Any	Static	*Code*-centric	○	○	●
Swift* [89]	◓	●	○	Any	Static	*Code*-centric	●	○	●
Jif* [261]	◓	●	●	Any	Static	*Code*-centric	●	○	●
PM [170]	◓	●	●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]	◓	●	●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Semi-automated tool for *safeboxing* code

Pick one influential example from the literature: **PtrSplit**[(1)]

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●	μLibrary	Static[2]	*Code*-centric	●	○	○
Google SAPI* [22]	◔	●	●	Function	Static	*Code*-centric	●	○	○
FlexOS* [143, 161]				μLibrary	Dynamic	*Code*-centric	●	●	○
RLBox* [181]				Function	Static	Any	●	○	●
SOAAP* [117]				Any	Hybrid	Any	●	●	●
SeCage* [171]				Function	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓		○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [			○	Function	Static	*Code*-centric	●	○	●
Shreds* [87],			○	Any	Static	*Code*-centric	○	○	●
DataShield*			○	Any	Static	*Code*-centric	○	○	●
Swift* [89]			○	Any	Static	*Code*-centric	●	○	●
Jif* [261]			●	Any	Static	*Code*-centric	●	○	●
PM [170]			●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]			●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Semi-automated tool for *safeboxing* code

Annotate valuable data, PtrSplit then automatically cuts the program to safebox them

Pick one influential example from the literature: **PtrSplit**(1)

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●			*ode*-centric	●	○	○
Google SAPI* [22]	◔	●	●			*de*-centric	●	○	○
FlexOS* [143, 161]						*de*-centric	●	●	○
RLBox* [181]						Any	●	○	●
SOAAP* [117]						Any	●	●	●
SeCage* [171]					Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓		○	Function	Hybrid	*Code*-centric	●	○	○
Glamdring* [			○	Function	Static	*Code*-centric	●	○	●
Shreds* [87],			○	Any	Static	*Code*-centric	○	○	●
DataShield*			○	Any	Static	*Code*-centric	○	○	●
Swift* [89]			○	Any	Static	*Code*-centric	●	○	●
Jif* [261]			●	Any	Static	*Code*-centric	●	○	●
PM [170]			●	Function	Hybrid	*Code*-centric	●	●	○
KSplit* [133]			●	Driver	Static	*Code*-centric	●	○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Semi-automated tool for *safeboxing* code

The program is then split at *arbitrary function boundaries*

Annotate valuable data, PtrSplit then automatically cuts the program to safebox them

(1) Liu et al., PtrSplit: Supporting General Pointers in Automatic Program Partitioning, CCS 2017

Pick one influential example from the literature: **PtrSplit**[(1)]

Policy Definition Method	*Automation* ○◔◓●[1]	*Policy Language Type*		*Separation Granularity*	*Analysis Approach*	*Subject Selection*	*Language Specific*	*Additional Goals of Automation*	
		Annotations	*Placement Rules*					*Performance*	*Interface Safety*
Manual [128], [...]	○	N/A	N/A	Any	Manual	Any	○	N/A	N/A
Crowbar [70]	◔	○	○	Function	Dynamic	*Code*-centric	○	○	○
MPDs* [191, 192]	◔	○	○	Component	Hybrid	*Code*-centric	●	●	○
CubicleOS* [211]	◔	●	●			*ode*-centric	●	○	○
Google SAPI* [22]	◔	●	●			*de*-centric	●	○	○
FlexOS* [143, 161]						*de*-centric	●	●	○
RLBox* [181]						Any	●	○	●
SOAAP* [117]						Any	●	●	●
SeCage* [171]				...on	Hybrid	*Code*-centric	●	○	○
PtrSplit* [169]	◓	●	○	Function	Static	*Code*-centric	●	○	○
PrivTrans* [73]	◓		○	Function		*Code*-centric	●	○	○
Glamdring* [			○	F...				○	●
Shreds* [87],			○					○	●
DataShield*			○					○	●
Swift* [89]			○					○	●
Jif* [261]			●					○	●
PM [170]			●	F...				●	○
KSplit* [133]			●					○	○
Cali* [65]	◓	○	●	Library	Static	*Code*-centric	○	○	○
CompartOS* [55]	◓	○	●	Linkage Unit	Static	*Code*-centric	○	○	○
Enclosure* [111]	◓	○	●	Package	Static	*Code*-centric	●	○	○
BreakApp* [235]	◓	○	●	Package	Static	*Code*-centric	●	○	○
CompARTist* [132]	◓	○	●	Library	Static	*Code*-centric	●	○	○
ACES* [90]	◓	○	●	Function	Any[3]	*Code*-centric	○	○	○
ProgramCutter [253]	●	N/A	N/A	Function	Dynamic	*Code*-centric	●	○	○
μSCOPE [202], SCALPEL [203]	●	N/A	N/A	Any	Dynamic	*Code*-centric	○	●	○

[1] ○ = manual, ◔ = guided manual, ◓ = policy refinement, ● = full automation. [2] Loader-based. [3] Implemented with static analysis, dynamic analysis possible [90].

Semi-automated tool for *safeboxing* code

The program is then split at *arbitrary function boundaries*

Annotate valuable data, PtrSplit then automatically cuts the program to safebox them

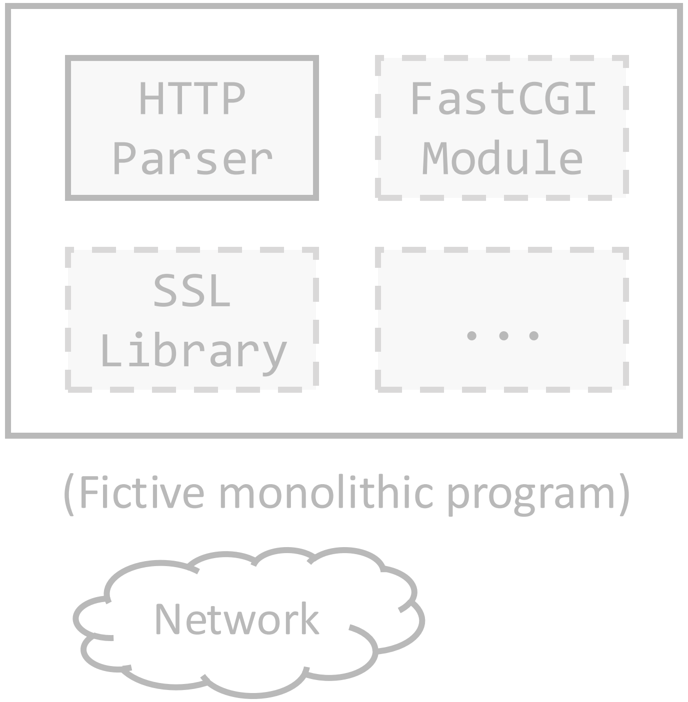
The split is done *statically* by the tool, i.e., just by looking at the code (vs. a dynamic tool that would run the program)

Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.



(Fictive monolithic program)

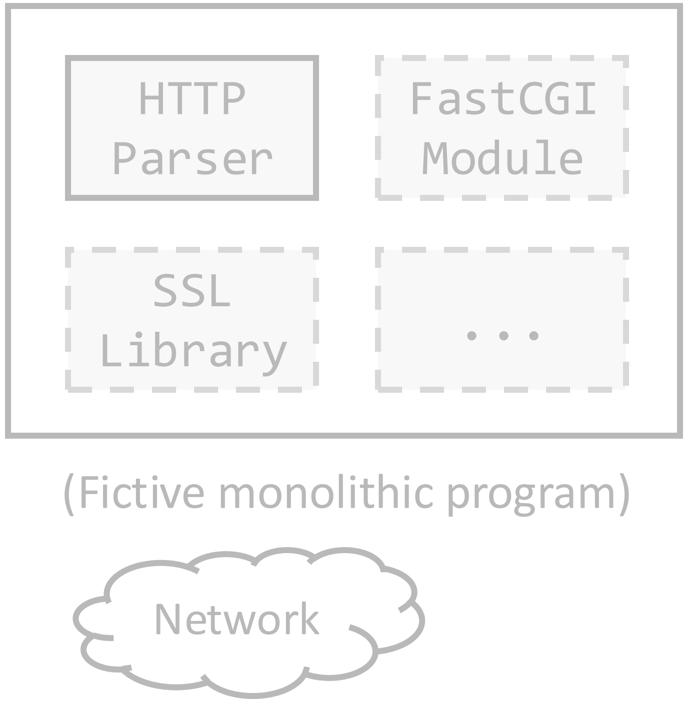
Which trade-offs are desirable in practice?

Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.



(Fictive monolithic program)

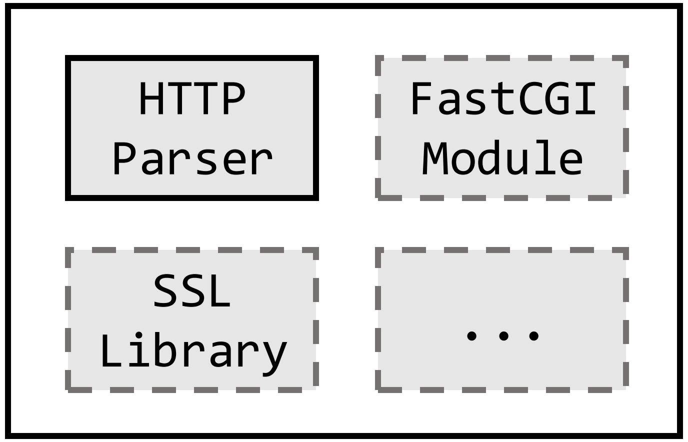
Which trade-offs are desirable in practice?

Open Problem in Policy Definition Methods

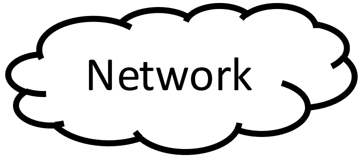
Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.



(Fictive monolithic program)



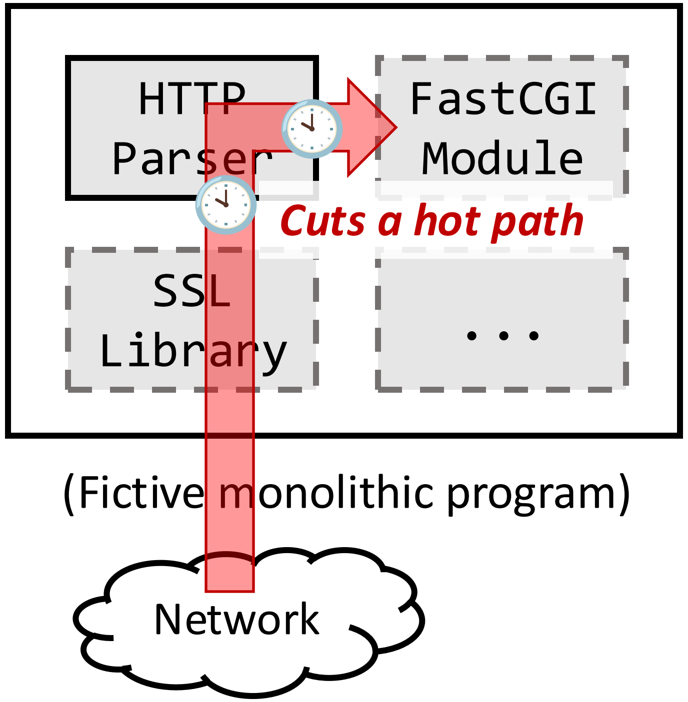
Which trade-offs are desirable in practice?

Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort.**

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.



(Fictive monolithic program)

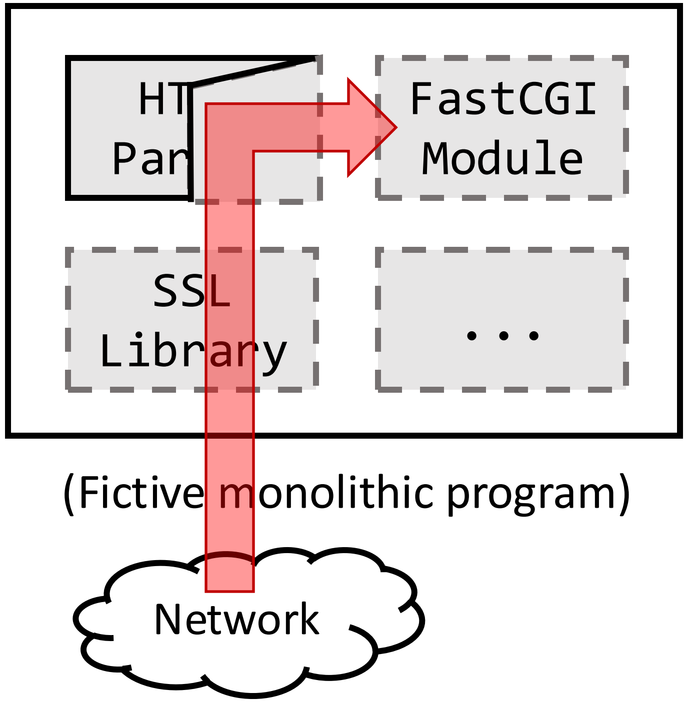
Which trade-offs are desirable in practice?

Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.



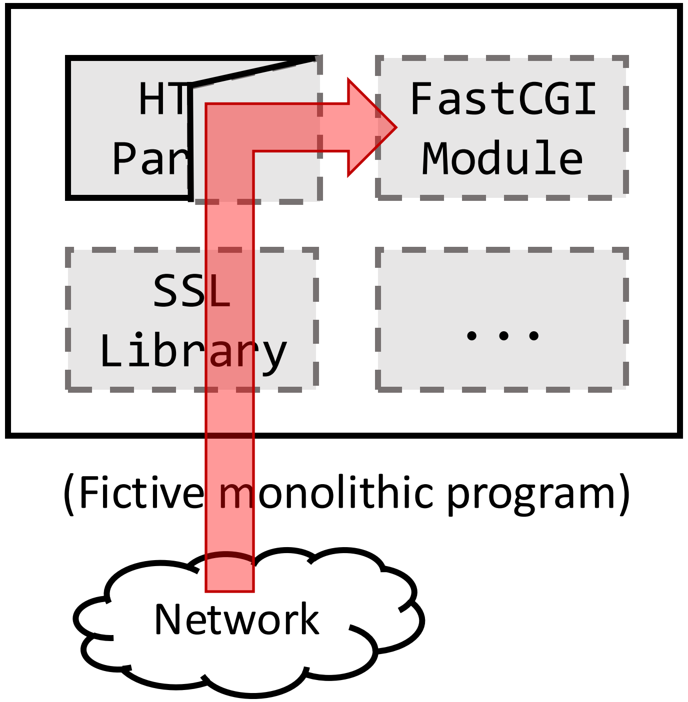
Which trade-offs are desirable in practice?

Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.



Rice's theorem: "*non-trivial semantic properties of programs are undecidable*".

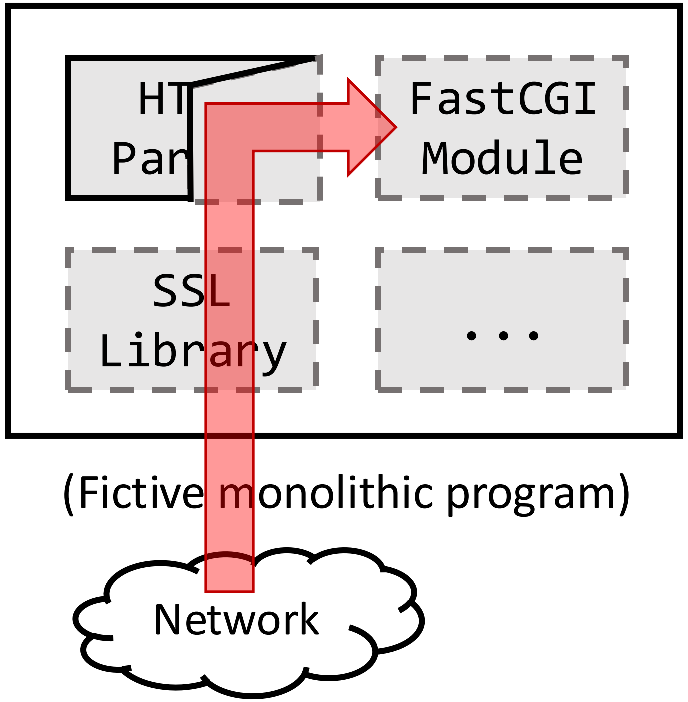
Which trade-offs are desirable in practice?

Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.



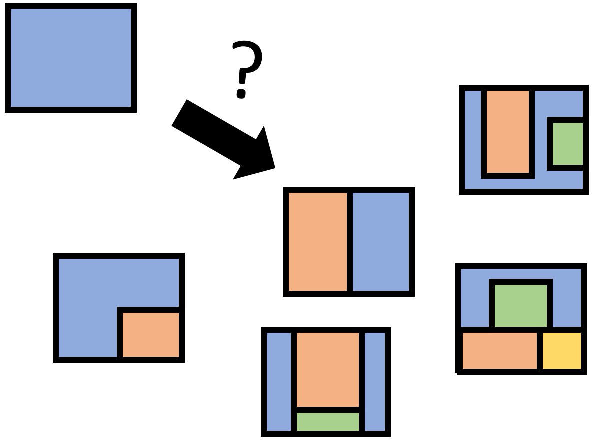
(Fictive monolithic program)

Rice's theorem: "*non-trivial semantic properties of programs are undecidable*".

Which trade-offs are desirable in practice?

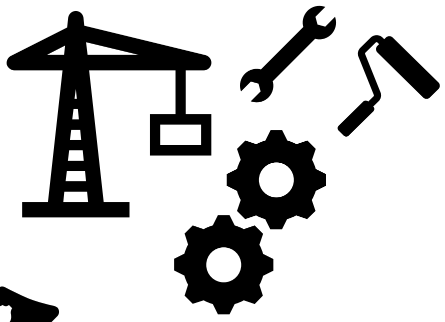
Bigger Picture of Compartmentalization

Propose to view compartmentalization as 3 problems:



1. **How to determine the right policy to enforce?**
   - Done with a *policy definition method*

2. **How to integrate the notion of compartmentalization policies in software / programming models / idioms?**
   - Done with a *compartmentalization abstraction*



3. **How to enforce policies at runtime?**
   - Done with a *compartmentalization mechanism*

Problem #2

How to implement policies?

Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

Problem #2

How to implement policies?

Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

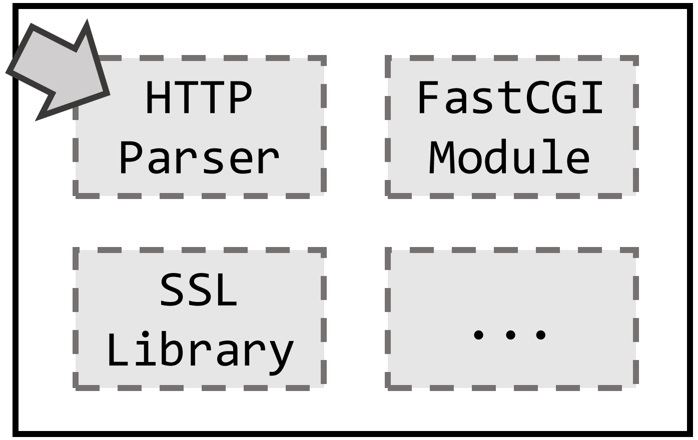
Historically, people have done this with **processes** (=the *process abstraction*).

Problem #2

How to implement policies?

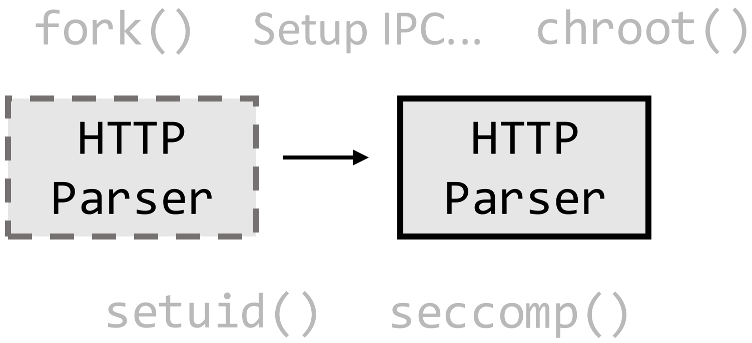
Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

Historically, people have done this with **processes** (=the *process abstraction*).



(Fictive monolithic program)

Example: *assume we sandbox the HTTP parser.*



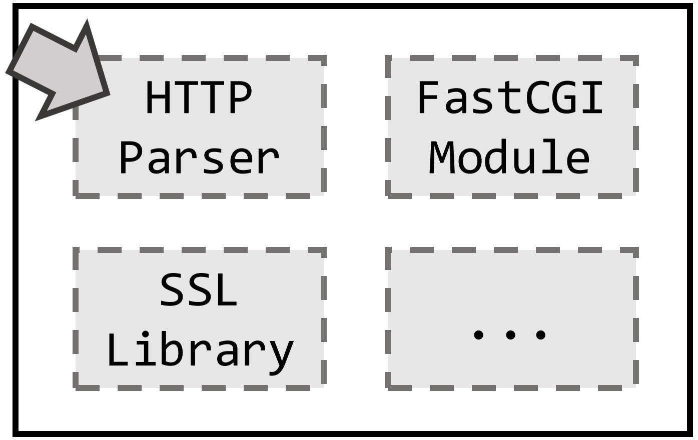
This is still the most common way to do it today.

Problem #2

How to implement policies?

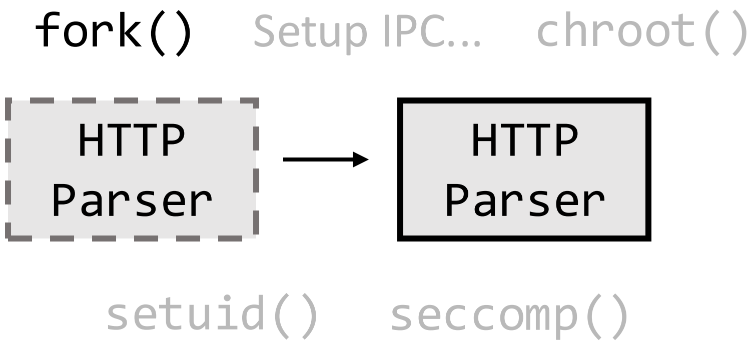
Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

Historically, people have done this with **processes** (=the *process abstraction*).



(Fictive monolithic program)

Example: *assume we sandbox the HTTP parser.*



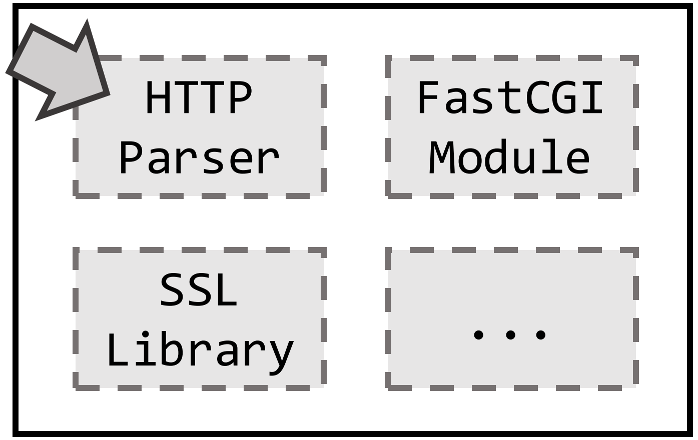
This is still the most common way to do it today.

Problem #2

How to implement policies?

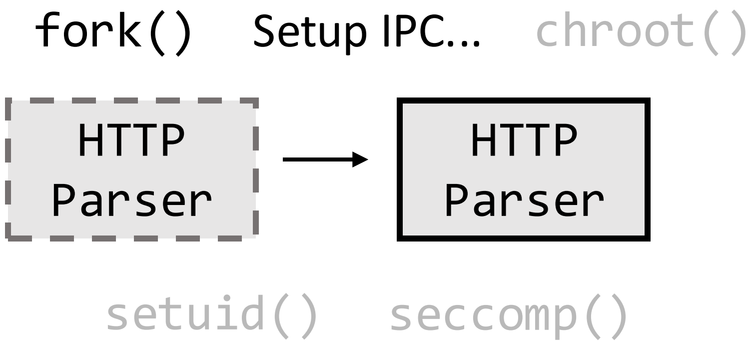
Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

Historically, people have done this with **processes** (=the *process abstraction*).



(Fictive monolithic program)

Example: *assume we sandbox the HTTP parser.*



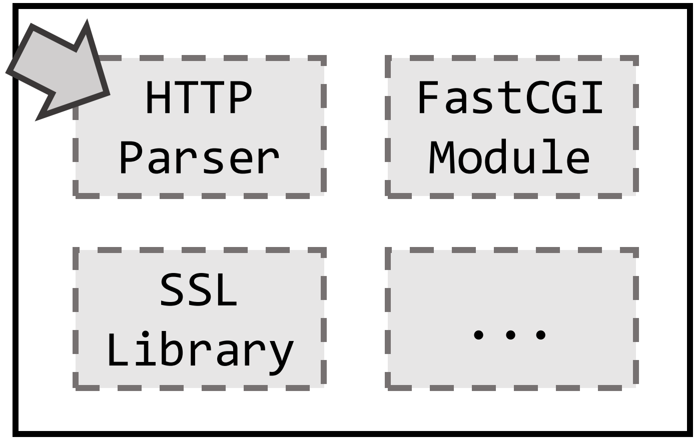
This is still the most common way to do it today.

Problem #2

How to implement policies?

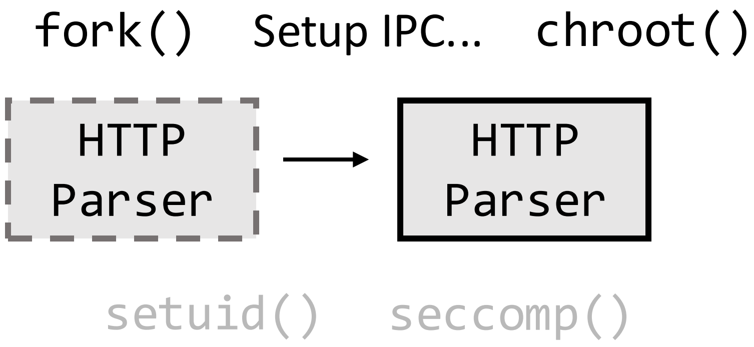
Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

Historically, people have done this with **processes** (=the *process abstraction*).



(Fictive monolithic program)

Example: *assume we sandbox the HTTP parser.*



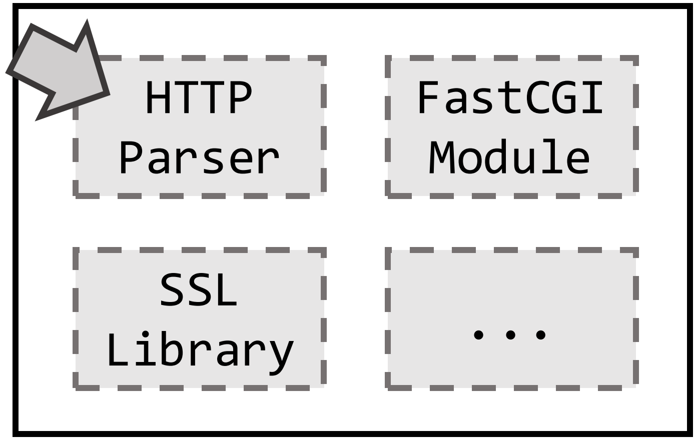
This is still the most common way to do it today.

Problem #2

How to implement policies?

Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

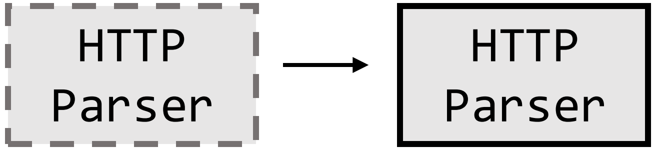
Historically, people have done this with **processes** (=the *process abstraction*).



(Fictive monolithic program)

Example: *assume we sandbox the HTTP parser.*

`fork()` Setup IPC... `chroot()`



`setuid()` `seccomp()`

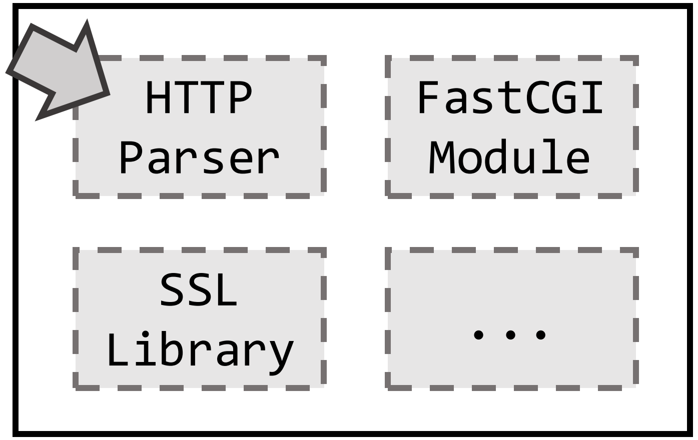
This is still the most common way to do it today.

Problem #2

How to implement policies?

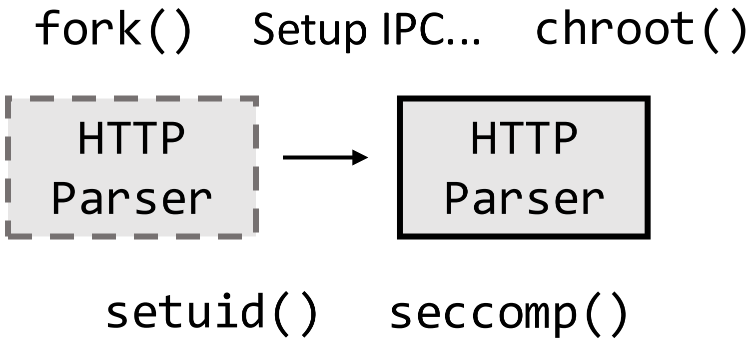
Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

Historically, people have done this with **processes** (=the *process abstraction*).



(Fictive monolithic program)

Example: *assume we sandbox the HTTP parser.*



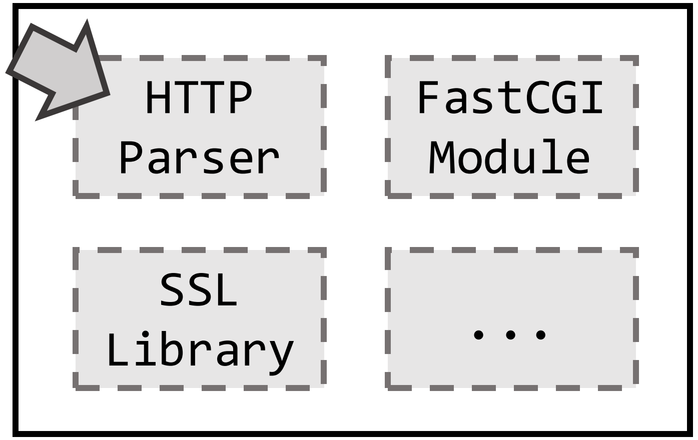
This is still the most common way to do it today.

Problem #2

How to implement policies?

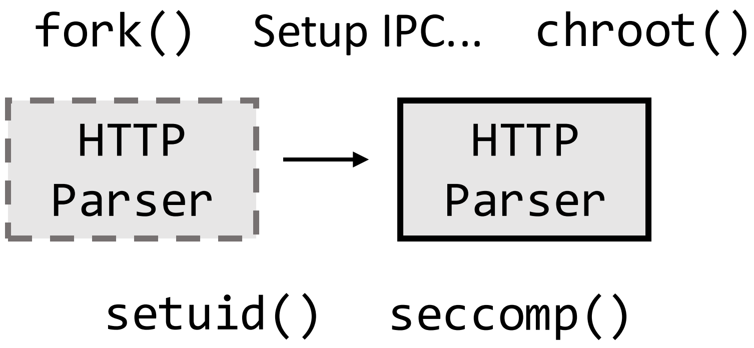
Having defined a policy, we need to **express it in the program**.
Developers perform this using **programming abstractions.**

Historically, people have done this with **processes** (=the *process abstraction*).



(Fictive monolithic program)

Example: *assume we sandbox the HTTP parser.*



This is still the most common way to do it today.

Compartmentalization Abstractions

Programming abstractions for compartmentalization are also a very active research area:

- Minimize developer effort (intuitive, easy to use)
- Leverage domain-specific knowledge (threat model, deployment)
- Maximize security properties that can be achieved

- Making the most of a specific enforcement mechanism
- Support generic mechanisms

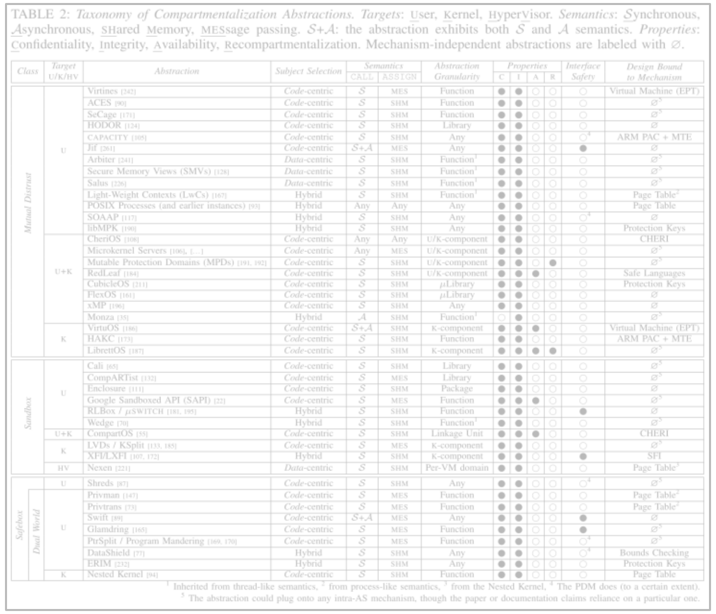


TABLE 2: *Taxonomy of Compartmentalization Abstractions. Targets*: User, Kernel, HyperVisor. *Semantics*: $\mathcal{S}$ynchronous, $\mathcal{A}$synchronous, SHared Memory, MESsage passing. $\mathcal{S}+\mathcal{A}$: the abstraction exhibits both $\mathcal{S}$ and $\mathcal{A}$ semantics. *Properties*: Confidentiality, Integrity, Availability, Recompartmentalization. Mechanism-independent abstractions are labeled with ∅.

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

(1)

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Compartmentalization Abstractions

Programming abstractions for compartmentalization are also a very active research area:

- Minimize developer effort (intuitive, easy to use)
- Leverage domain-specific knowledge (threat model, deployment)
- Maximize security properties that can be achieved

- Making the most of a specific enforcement mechanism
- Support generic mechanisms

TABLE 2: *Taxonomy of Compartmentalization Abstractions. Targets*: User, Kernel, HyperVisor. *Semantics*: $\mathcal{S}$ynchronous, $\mathcal{A}$synchronous, SHared Memory, MESsage passing. $\mathcal{S}+\mathcal{A}$: the abstraction exhibits both $\mathcal{S}$ and $\mathcal{A}$ semantics. *Properties*: Confidentiality, Integrity, Availability, Recompartmentalization. Mechanism-independent abstractions are labeled with ∅.

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent). [5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.


SoK: Software Compartmentalization

Hugo Lefeuvre, Nathan Dautenhahn, David Chisnall, Pierre Olivier

The University of British Columbia, Riverside, SCI Semiconductor, The University of Manchester

...towards simplifying the definition of compartmentalization policies; towards better challenging our threat models in the light of confused deputies and hardware limitations; as well as towards bridging the gaps we pinpoint between research and mainstream needs. This paper not only maps the historical and current landscape of compartmentalization, but also sets forth a framework to foster their evolution and adoption.

1. Introduction

Despite decades of effort, vulnerabilities still plague software, and thwarting them remains a game of cat and mouse. The *principle of least privilege* (PoLP) [209] is the last line of defense when protections fail or when flaws are unknown. By granting each entity only the privileges needed, the PoLP ensures that a compromise of one part will not imply that of the whole. *Software compartmentalization* is a prominent implementation of the PoLP, in which developers divide a large program into smaller, lesser privileged components, to reduce the impact of potential security breaches.

Software compartmentalization inherits from a large body of work, starting with processes [93]; including OS models such as microkernels [106, 126, 150, 164, 258], security and separation kernels [56, 58, 205, 206, 247], or capability OSes [83, 125]; all the way to fine-grain application compartmentalization...

...compartmentalization can bring. This paper investigates the reasons behind this status quo, and how to improve it.

Research speculated that this situation is due to a lack of automation [65, 253], limitations of mechanisms [190, 259], excessive performance overheads [188, 232], a lack of strong security guarantees [129, 160], among others. All are likely part of the problem, and through decades of research progress was made on every front. Today the community lacks a global overview of this progress. Designing for or retrofitting compartmentalization is often hampered by inconsistencies in the understanding and application of its concepts. Existing models and terminologies are numerous, ad-hoc, and often contradictory, leading to confusion and a growing body of work that cannot be compared. The lack of a systematic perspective leads to a mismatch between what software compartmentalization needs to progress towards the mainstream, and the focus and framing of research efforts: most do not tackle compartmentalization's key aspects as a whole and thus produce solutions that cannot be relevant to making it a mainstream practice.

Recognizing these challenges, we propose a unified model providing a consistent framework for defining, understanding, and implementing compartmentalization. This model comes with a comprehensive taxonomy that allows to classify compartmentalization strategies...

(1)

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	Properties I	Properties A	Properties R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Names of the abstractions we consider

Characteristics we included in the taxonomy

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	Properties I	Properties A	Properties R	Interface Safety	Desig[illegible]nd to [illegible]sm
Mutual Distrust	U	Virtines [illegible]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual [illegible] (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Names of the abstractions we consider

Characteristics we included in the taxonomy

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	I	A	R	Interface Safety	Desig[illegible]nd to[illegible]sm
Mutual Distrust	U	Virtines [illegible]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual [illegible] (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light Weight Contexts (LwCs) [illegible]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [illegible]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

This talk: present a few interesting aspects, full discussion in the paper 🙂

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	Properties I	Properties A	Properties R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Abstractions often **specialize** on a particular threat model

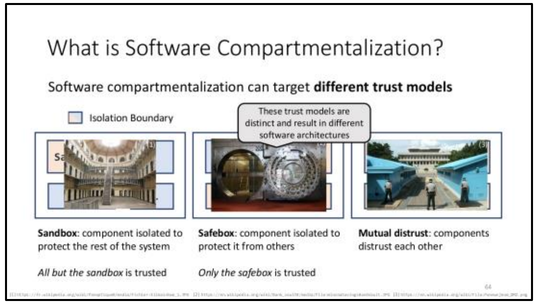
Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	Properties I	Properties A	Properties R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

vs. processes, which are quite generic but also trickier to use

For **sandboxing** (libraries, drivers)

For **safeboxing** (crypto keys, privileged code)



A lot of interest in abstractions for **kernel compartmentalization**

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	Properties I	Properties A	Properties R	Interface Safety	Design Bound to Mechanism
	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
	U	ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
	U	SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
	U	HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
	U	CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
	U	Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
	U	Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U	[128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U		*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U	[167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
	U	stances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
	U		Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
	U		Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K		*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
	U+K	Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
	U+K	Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
	U+K	RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
	U+K	CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
	U+K	FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
	U+K	xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
	U+K	Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
	K	HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
	K	LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
		[65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		pARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	
		osure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	
Sandbox		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
	K	XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox / Dual Wo	U		*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
	U		*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
	U		*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
	U	[89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
	U	Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
	U	PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
	U	DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
	U	ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Compartmentalize **userland and kernel** together

Kernel only...

Or **hypervisors**

This came as a revival of research in microkernels

A lot of interest in rethinking **communication** and **sharing** between domains

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	ASSIGN	Abstraction Granularity	Properties C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-ce									∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybr									Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybr									Page Table
		SOAAP [117]	Hybr									∅
		libMPK [190]	Hybr									Protection Keys
	U+K	CheriOS [108]	*Code*-ce									CHERI
		Microkernel Servers [106], [...]	*Code*-ce									∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$		U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric		SHM	Package	●	●	○	○	○	∅
		Google Sandb					●	●	●	○	○	∅[5]
		RLBox / μS				tion	●	●	○	○	●	∅
		Wedge [70]				tion[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS				e Unit	●	●	●	○	○	CHERI
	K	LVDs / KSp				ponent	●	●	○	○	○	∅[5]
		XFI/LXFI [1				ponent	●	●	○	○	●	SFI
	HV	Nexen [221]				domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]				ny	●	●	○	○	○[4]	∅[5]
Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Leverage **shared memory to speed up sharing** between domains (vs. message passing)

Exposing **communication between domains as function calls** (vs. IPCs) to be more intuitive

Interest in abstractions that **compose with different isolation mechanisms**

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	Properties I	Properties A	Properties R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	[illegible]	[illegible]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	[illegible]	[illegible]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	[illegible]	[illegible]
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	[illegible]	[illegible]
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	[illegible]	[illegible]
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	[illegible]	[illegible]
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	[illegible]	[illegible]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	[illegible]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

For example: *new abstractions that leverage the specifics of safe languages for isolation*

Pick one influential example from the literature: **RLBox**(1)

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics: CALL	Semantics: ASSIGN	Abstraction Granularity	Properties: C	Properties: I	Properties: A	Properties: R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Pick one influential example from the literature: **RLBox**[1]

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	Properties I	Properties A	Properties R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], […]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		[illegible]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		[illegible]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		[illegible]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
		[illegible]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		[illegible]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		[illegible]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	[illegible]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		[illegible]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	**Hybrid**	$\mathcal{S}$	**SHM**	**Function**	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Abstraction specialized for sandboxing untrusted code (such as libraries)

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Pick one influential example from the literature: **RLBox**[(1)]

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
			Code-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
			Code-centric	$\mathcal{S}$	SHM	Any					○	∅
												∅[5]
												Virtual Machine (EPT)
												ARM PAC + MTE
												∅[5]
												∅[5]
												∅[5]
Sandbox	U	Enclosure [111]									○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric		MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	**Hybrid**	**$\mathcal{S}$**	**SHM**	**Function**	**●**	**●**	**○**	**○**	**●**	**∅**
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Abstraction specialized for sandboxing untrusted code (such as libraries)

Implements cross-compartment communication with function calls, which is intuitive

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Pick one influential example from the literature: **RLBox**[1]

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	ASSIGN	Abstraction Granularity	Properties C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
			Code-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
			Code-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
											○	∅[5]
												Virtual Machine (EPT)
												ARM PAC + MTE
												∅[5]
Sandbox	U											∅[5]
												∅[5]
		Enclosure [111]									○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric		MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$		Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]					●	●	●	○	○	CHERI
	K	LVDs / KSplit [133]					●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]					●	●	○	○	●	SFI
	HV	Nexen [221]					●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]					●	●	○	○	○[4]	∅[5]
Safebox / Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Abstraction specialized for sandboxing untrusted code (such as libraries)

Implements cross-compartment communication with function calls, which is intuitive

Builds on shared memory to speed up cross-compartment data sharing

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Pick one influential example from the literature: **RLBox**(1)

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], […]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
			Code-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
			Code-centric	$\mathcal{S}$	SHM	Any					○	∅
												∅[5]
												Virtual Machine (EPT)
												ARM PAC + MTE
												∅[5]
Sandbox	U											∅[5]
												∅[5]
		Enclosure [111]									○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric		MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	**Hybrid**	**$\mathcal{S}$**	**SHM**	**Function**	**●**	**●**	**○**	**○**	**●**	**∅**
		Wedge [70]	Hybrid	$\mathcal{S}$		Function[1]						∅[5]
	U+K	CompartOS [55]										
	K	LVDs / KSplit [133, …]										
		XFI/LXFI [107, 172]										
	HV	Nexen [221]					●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]					●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Abstraction specialized for sandboxing untrusted code (such as libraries)

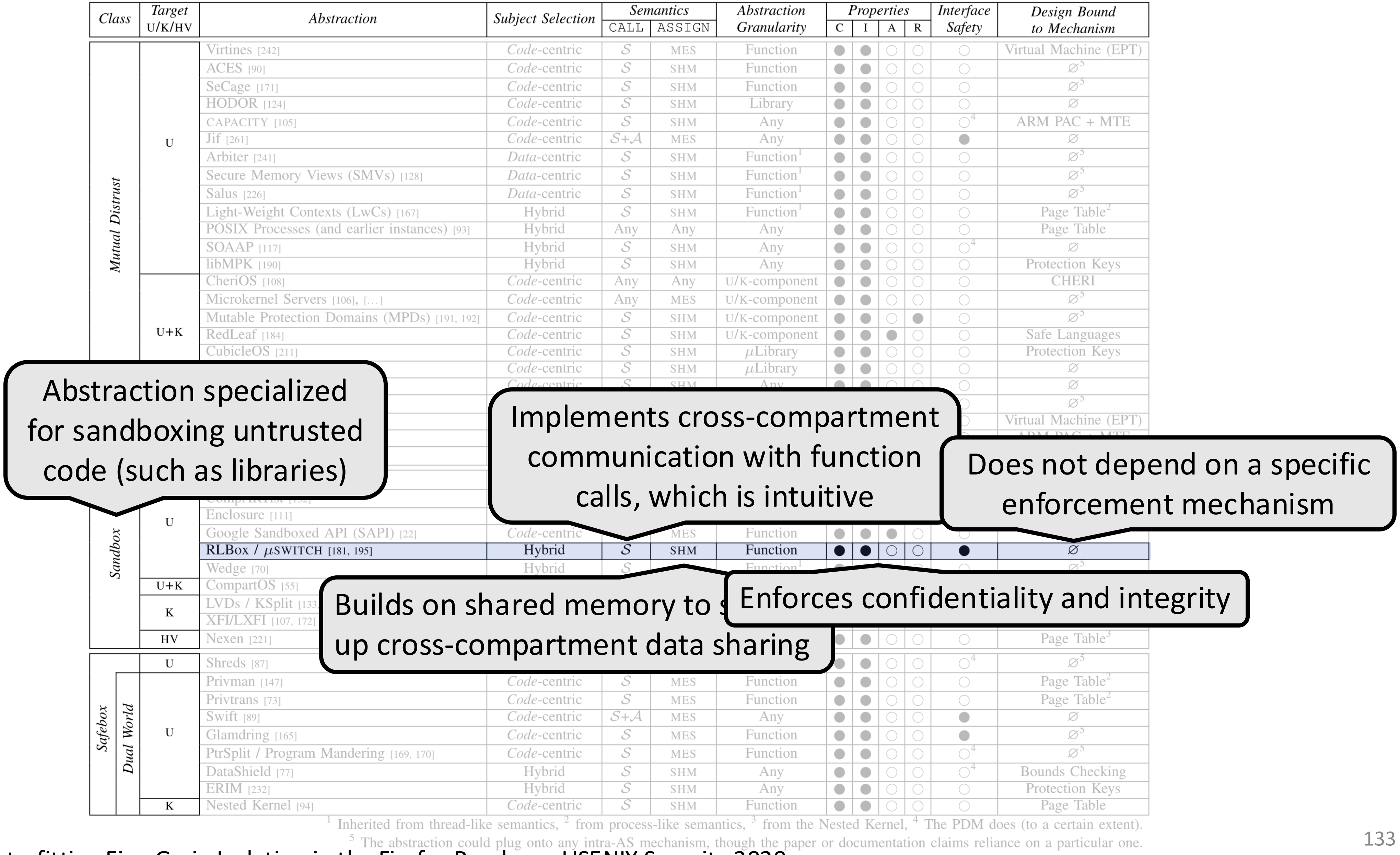
Implements cross-compartment communication with function calls, which is intuitive

Builds on shared memory to s[...] up cross-compartment data sharing

Enforces confidentiality and integrity

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Pick one influential example from the literature: **RLBox**(1)



Class	Target U/K/HV	Abstraction	Subject Selection	Semantics: CALL	Semantics: ASSIGN	Abstraction Granularity	Properties: C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], […]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
			Code-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
Sandbox	U	Enclosure [111]										
		Google Sandboxed API (SAPI) [22]	*Code*-centric		MES	Function	●	●	●	○	○	
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$		Function[1]						∅[5]
	U+K	CompartOS [55]										
	K	LVDs / KSplit [133]										
		XFI/LXFI [107, 172]										
	HV	Nexen [221]					●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]					●	●	○	○	○[4]	∅[5]
Safebox / Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

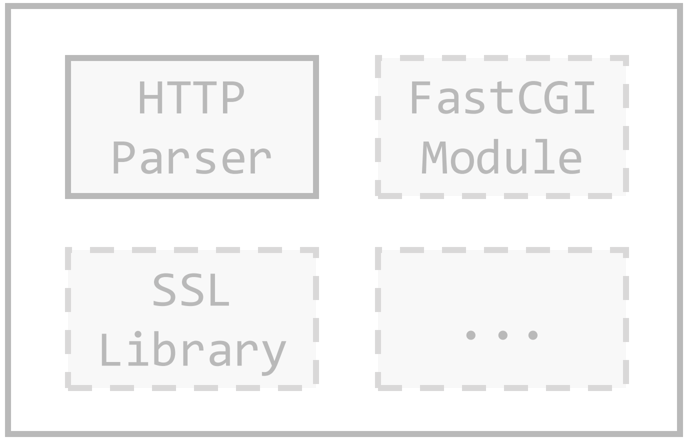
[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Open Problem in Compartmentalization Abstractions

Hardening compartment interfaces is key to obtaining strong security properties with compartmentalization

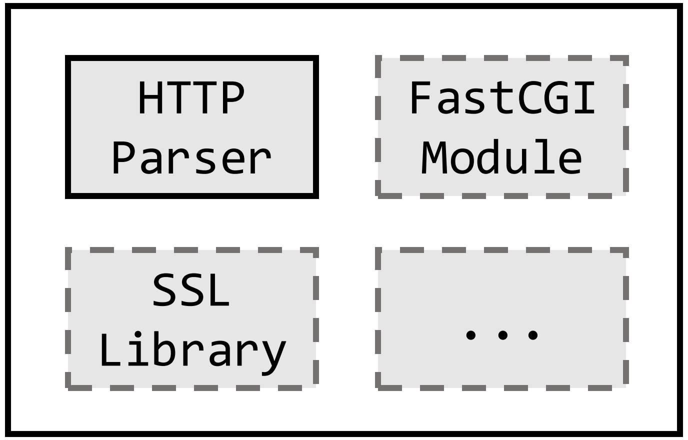
Example: assume the developer implemented our boundaries. Compartments *still need to communicate*, which opens an attack surface.



Open Problem in Compartmentalization Abstractions

Hardening compartment interfaces is key to obtaining strong security properties with compartmentalization

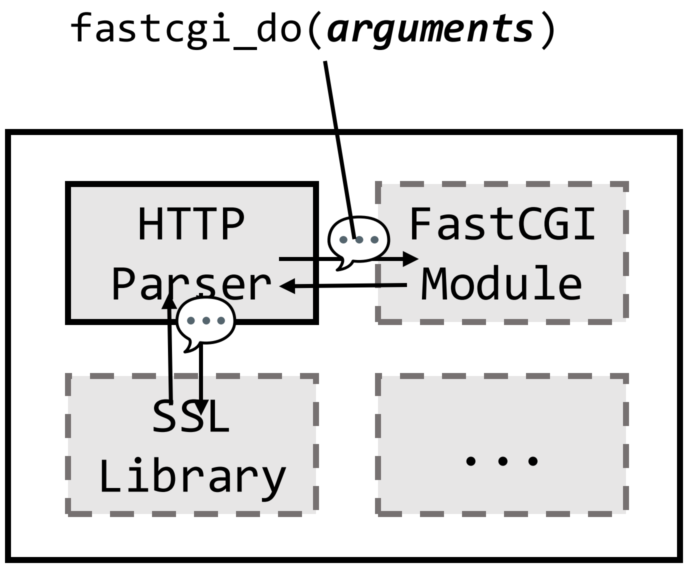
Example: assume the developer implemented our boundaries. Compartments *still need to communicate*, which opens an attack surface.



Open Problem in Compartmentalization Abstractions

Hardening compartment interfaces is key to obtaining strong security properties with compartmentalization

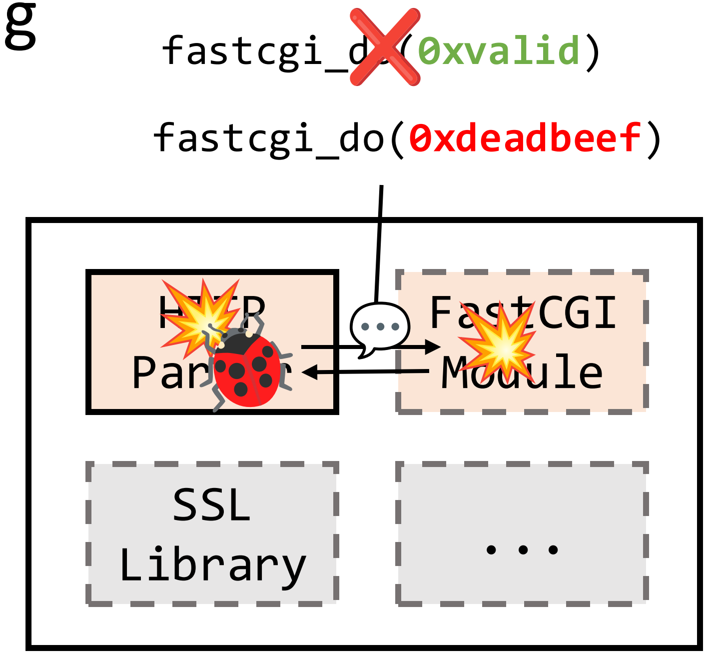
Example: assume the developer implemented our boundaries. Compartments *still need to communicate*, which opens an attack surface.



Open Problem in Compartmentalization Abstractions

Hardening compartment interfaces is key to obtaining strong security properties with compartmentalization

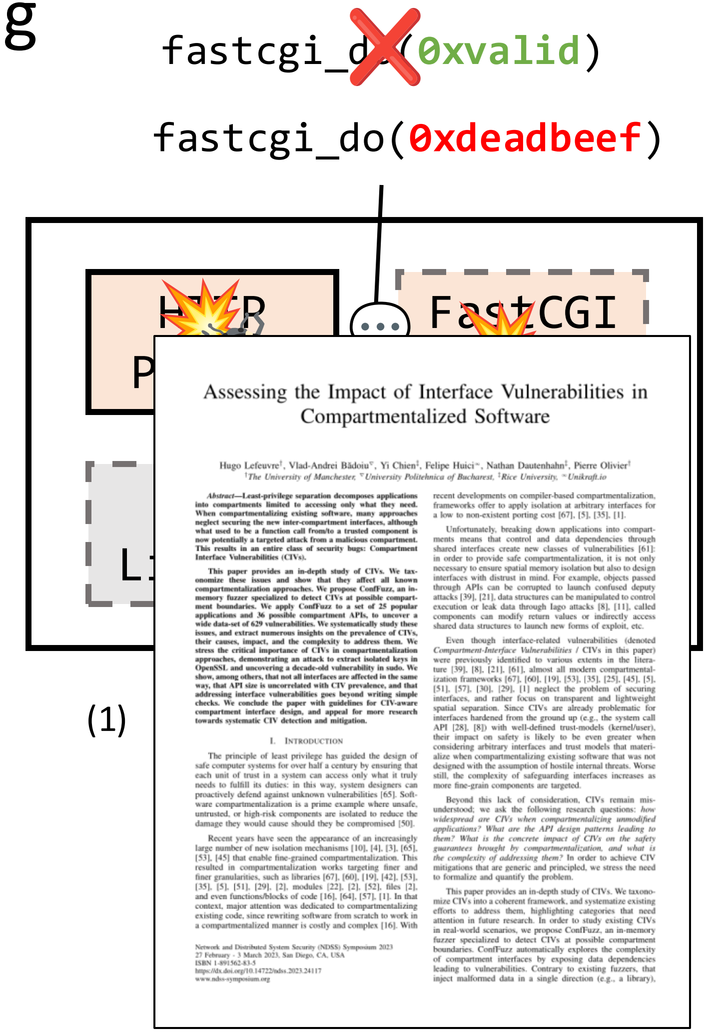
Example: assume the developer implemented our boundaries. Compartments *still need to communicate*, which opens an attack surface.



Open Problem in Compartmentalization Abstractions

Hardening compartment interfaces is key to obtaining strong security properties with compartmentalization

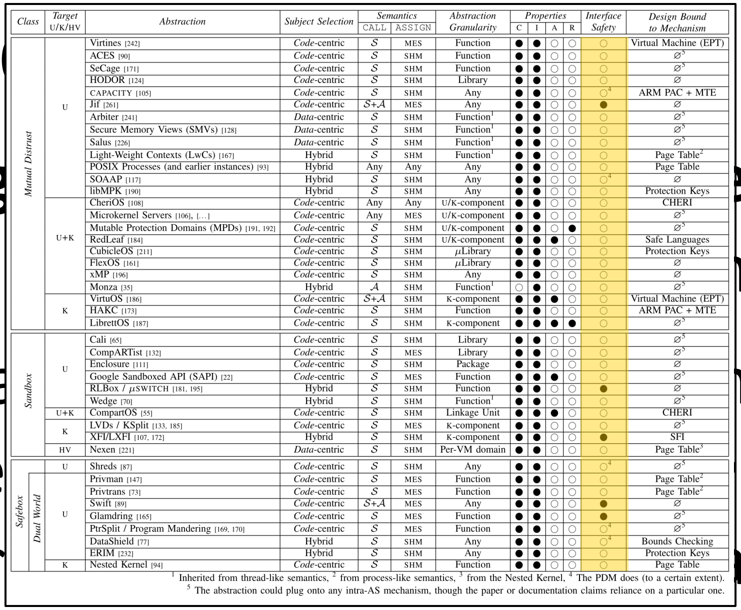
Example: assume the developer implemented our boundaries. Compartments *still need to communicate*, which opens an attack surface.



(1)

(1) Lefeuvre et al., Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software, NDSS'23

Open Problem in Compartmentalization Abstra



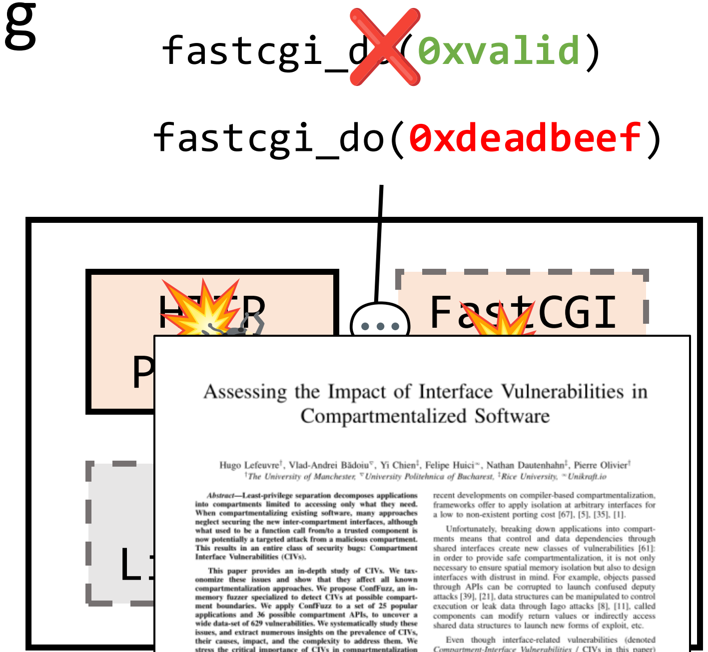
Class	Target U/K/HV	Abstraction	Subject Selection	Semantics		Abstraction Granularity	Properties				Interface Safety	Design Bound to Mechanism
				CALL	ASSIGN		C	I	A	R		
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / *Dual World*	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

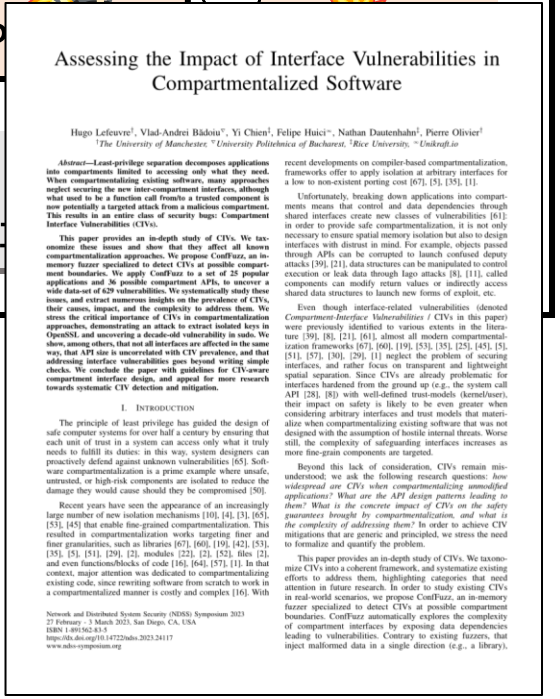
Hardening ... **es** is key to obtaining strong sec ... mpartmentalization

Example: a ... nplemented our boundarie ... *ed to communic* ... ack surface.

Most abstractions consider the **hardening of compartment interfaces** an orthogonal problem.



(1)



Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Hugo Lefeuvre[†], Vlad-Andrei Bădoiu[▽], Yi Chien[‡], Felipe Huici[~], Nathan Dautenhahn[‡], Pierre Olivier[†]
[†]The University of Manchester, [▽]University Politehnica of Bucharest, [‡]Rice University, [~]Unikraft.io

***Abstract*—Least-privilege separation decomposes applications into compartments limited to accessing only what they need. When compartmentalizing existing software, many approaches neglect securing the new inter-compartment interfaces, although what used to be a function call from/to a trusted component is now potentially a targeted attack from a malicious compartment. This results in an entire class of security bugs: Compartment Interface Vulnerabilities (CIVs).**

This paper provides an in-depth study of CIVs. We taxonomize these issues and show that they affect all known compartmentalization approaches. We propose ConfFuzz, an in-memory fuzzer specialized to detect CIVs at possible compartment boundaries. We apply ConfFuzz to a set of 25 popular applications and 36 possible compartment APIs, to uncover a wide data-set of 629 vulnerabilities. We systematically study these issues, and extract numerous insights on the prevalence of CIVs, their causes, impact, and the complexity to address them. We stress the critical importance of CIVs in compartmentalization approaches, demonstrating an attack to extract isolated keys in OpenSSL and uncovering a decade-old vulnerability in sudo. We show, among others, that not all interfaces are affected in the same way, that API size is uncorrelated with CIV prevalence, and that addressing interface vulnerabilities goes beyond writing simple checks. We conclude the paper with guidelines for CIV-aware compartment interface design, and appeal for more research towards systematic CIV detection and mitigation.

I. Introduction

The principle of least privilege has guided the design of safe computer systems for over half a century by ensuring that each unit of trust in a system can access only what it truly needs to fulfill its duties: in this way, system designers can proactively defend against unknown vulnerabilities [65]. Software compartmentalization is a prime example where unsafe, untrusted, or high-risk components are isolated to reduce the damage they would cause should they be compromised [50].

Recent years have seen the appearance of an increasingly large number of new isolation mechanisms [10], [4], [3], [65], [53], [45] that enable fine-grained compartmentalization. This resulted in compartmentalization works targeting finer and finer granularities, such as libraries [67], [60], [19], [42], [53], [35], [5], [51], [29], [2], modules [22], [2], [52], files [2], and even functions/blocks of code [16], [64], [57], [1]. In that context, major attention was dedicated to compartmentalizing existing code, since rewriting software from scratch to work in a compartmentalized manner is costly and complex [16]. With recent developments on compiler-based compartmentalization, frameworks offer to apply isolation at arbitrary interfaces for a low to non-existent porting cost [67], [5], [35], [1].

Unfortunately, breaking down applications into compartments means that control and data dependencies through shared interfaces create new classes of vulnerabilities [61]: in order to provide safe compartmentalization, it is not only necessary to ensure spatial memory isolation but also to design interfaces with distrust in mind. For example, objects passed through APIs can be corrupted to launch confused deputy attacks [39], [21], data structures can be manipulated to control execution or leak data through Iago attacks [8], [11], called components can modify return values or indirectly access shared data structures to launch new forms of exploit, etc.

Even though interface-related vulnerabilities (denoted *Compartment-Interface Vulnerabilities* / CIVs in this paper) were previously identified to various extents in the literature [39], [8], [21], [61], almost all modern compartmentalization frameworks [67], [60], [19], [53], [35], [25], [45], [5], [51], [57], [30], [29], [1] neglect the problem of securing interfaces, and rather focus on transparent and lightweight spatial separation. Since CIVs are already problematic for interfaces hardened from the ground up (e.g., the system call API [28], [8]) with well-defined trust-models (kernel/user), their impact on safety is likely to be even greater when considering arbitrary interfaces and trust models that materialize when compartmentalizing existing software that was not designed with the assumption of hostile internal threats. Worse still, the complexity of safeguarding interfaces increases as more fine-grain components are targeted.

Beyond this lack of consideration, CIVs remain misunderstood; we ask the following research questions: *how widespread are CIVs when compartmentalizing unmodified applications? What are the API design patterns leading to them? What is the concrete impact of CIVs on the safety guarantees brought by compartmentalization, and what is the complexity of addressing them?* In order to achieve CIV mitigations that are generic and principled, we stress the need to formalize and quantify the problem.

This paper provides an in-depth study of CIVs. We taxonomize CIVs into a coherent framework, and systematize existing efforts to address them, highlighting categories that need attention in future research. In order to study existing CIVs in real-world scenarios, we propose ConfFuzz, an in-memory fuzzer specialized to detect CIVs at possible compartment boundaries. ConfFuzz automatically explores the complexity of compartment interfaces by exposing data dependencies leading to vulnerabilities. Contrary to existing fuzzers, that inject malformed data in a single direction (e.g., a library),

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24117
www.ndss-symposium.org

Open Problem in Compartmentalization Abstra

Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	Semantics ASSIGN	Abstraction Granularity	Properties C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

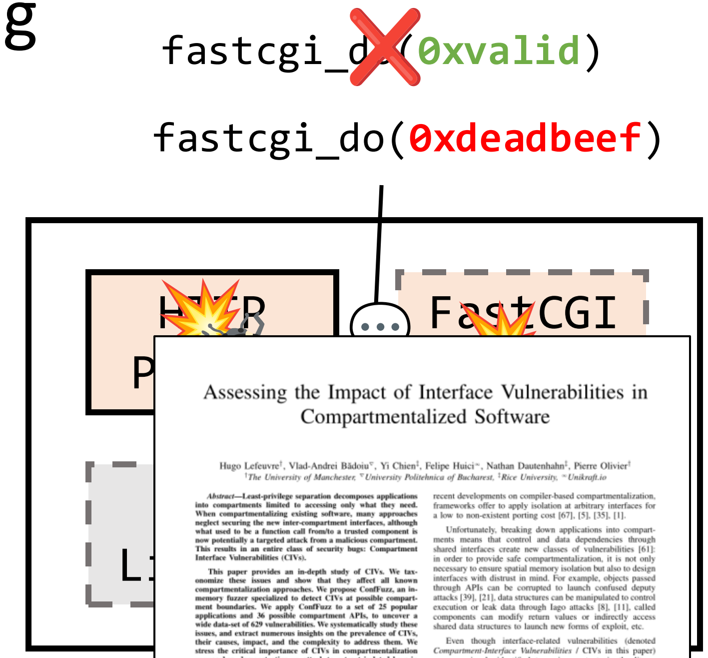
[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Hardening … **es** is key to obtaining strong sec… mpartmentalization

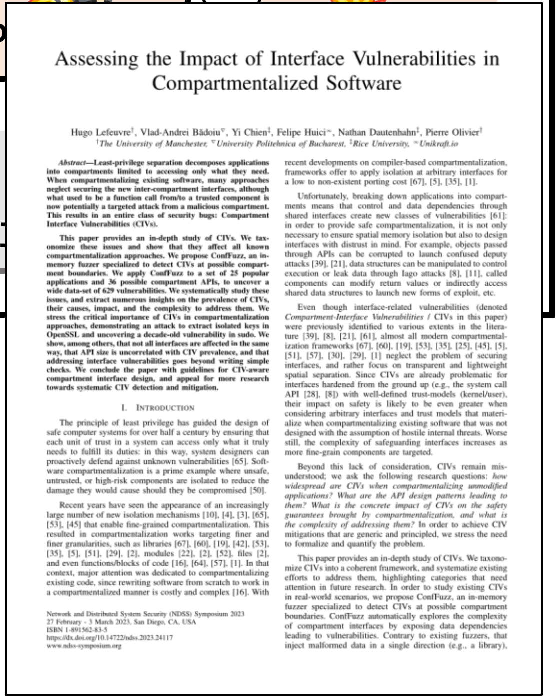
Example: a… nplemented our boundarie… *ed to communic…* …ck surface.

Most abstractions consider the **hardening of compartment interfaces** an orthogonal problem.

We stress that abstractions **can and should** facilitate the hardening of compartment interfaces.



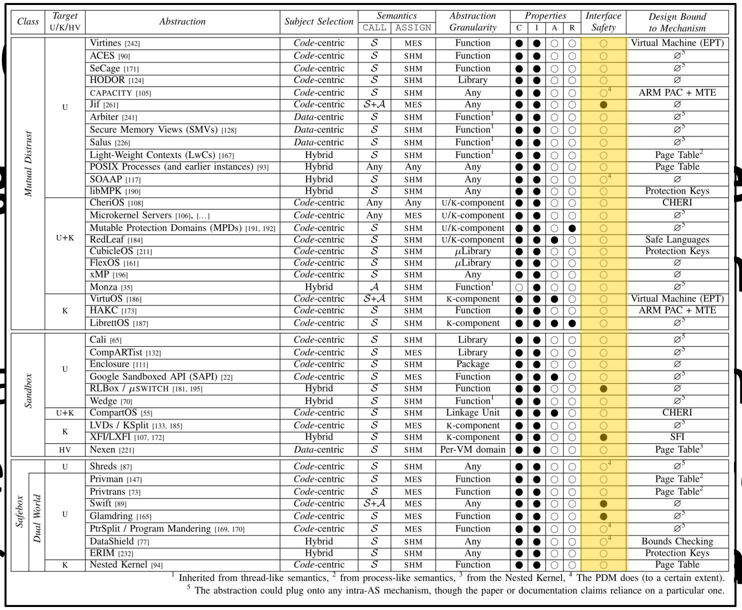
(1)


Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Hugo Lefeuvre, Vlad-Andrei Bădoiu, Yi Chien, Felipe Huici, Nathan Dautenhahn, Pierre Olivier

How can abstractions that facilitate the implementation of secure domain interfaces?

Open Problem in Compartmentalization Abstrac



Class	Target U/K/HV	Abstraction	Subject Selection	Semantics CALL	ASSIGN	Abstraction Granularity	Properties C	I	A	R	Interface Safety	Design Bound to Mechanism
Mutual Distrust	U	Virtines [242]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Virtual Machine (EPT)
		ACES [90]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		SeCage [171]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	∅[5]
		HODOR [124]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅
		CAPACITY [105]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	ARM PAC + MTE
		Jif [261]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Arbiter [241]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Secure Memory Views (SMVs) [128]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Salus [226]	*Data*-centric	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
		Light-Weight Contexts (LwCs) [167]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	Page Table[2]
		POSIX Processes (and earlier instances) [93]	Hybrid	Any	Any	Any	●	●	○	○	○	Page Table
		SOAAP [117]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅
		libMPK [190]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	U+K	CheriOS [108]	*Code*-centric	Any	Any	U/K-component	●	●	○	○	○	CHERI
		Microkernel Servers [106], [...]	*Code*-centric	Any	MES	U/K-component	●	●	○	○	○	∅[5]
		Mutable Protection Domains (MPDs) [191, 192]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	○	●	○	∅[5]
		RedLeaf [184]	*Code*-centric	$\mathcal{S}$	SHM	U/K-component	●	●	●	○	○	Safe Languages
		CubicleOS [211]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	Protection Keys
		FlexOS [161]	*Code*-centric	$\mathcal{S}$	SHM	μLibrary	●	●	○	○	○	∅
		xMP [196]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○	∅
		Monza [35]	Hybrid	$\mathcal{A}$	SHM	Function[1]	○	●	○	○	○	∅[5]
	K	VirtuOS [186]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	SHM	K-component	●	●	●	○	○	Virtual Machine (EPT)
		HAKC [173]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	ARM PAC + MTE
		LibrettOS [187]	*Code*-centric	$\mathcal{S}$	SHM	K-component	●	●	●	●	○	∅[5]
Sandbox	U	Cali [65]	*Code*-centric	$\mathcal{S}$	SHM	Library	●	●	○	○	○	∅[5]
		CompARTist [132]	*Code*-centric	$\mathcal{S}$	MES	Library	●	●	○	○	○	∅[5]
		Enclosure [111]	*Code*-centric	$\mathcal{S}$	SHM	Package	●	●	○	○	○	∅
		Google Sandboxed API (SAPI) [22]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	●	○	○	∅[5]
		RLBox / μSWITCH [181, 195]	Hybrid	$\mathcal{S}$	SHM	Function	●	●	○	○	●	∅
		Wedge [70]	Hybrid	$\mathcal{S}$	SHM	Function[1]	●	●	○	○	○	∅[5]
	U+K	CompartOS [55]	*Code*-centric	$\mathcal{S}$	SHM	Linkage Unit	●	●	●	○	○	CHERI
	K	LVDs / KSplit [133, 185]	*Code*-centric	$\mathcal{S}$	MES	K-component	●	●	○	○	○	∅[5]
		XFI/LXFI [107, 172]	Hybrid	$\mathcal{S}$	SHM	K-component	●	●	○	○	●	SFI
	HV	Nexen [221]	*Data*-centric	$\mathcal{S}$	SHM	Per-VM domain	●	●	○	○	○	Page Table[3]
Safebox	U	Shreds [87]	*Code*-centric	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	∅[5]
Safebox / Dual World	U	Privman [147]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Privtrans [73]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○	Page Table[2]
		Swift [89]	*Code*-centric	$\mathcal{S}+\mathcal{A}$	MES	Any	●	●	○	○	●	∅
		Glamdring [165]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	●	∅[5]
		PtrSplit / Program Mandering [169, 170]	*Code*-centric	$\mathcal{S}$	MES	Function	●	●	○	○	○[4]	∅[5]
		DataShield [77]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○[4]	Bounds Checking
		ERIM [232]	Hybrid	$\mathcal{S}$	SHM	Any	●	●	○	○	○	Protection Keys
	K	Nested Kernel [94]	*Code*-centric	$\mathcal{S}$	SHM	Function	●	●	○	○	○	Page Table

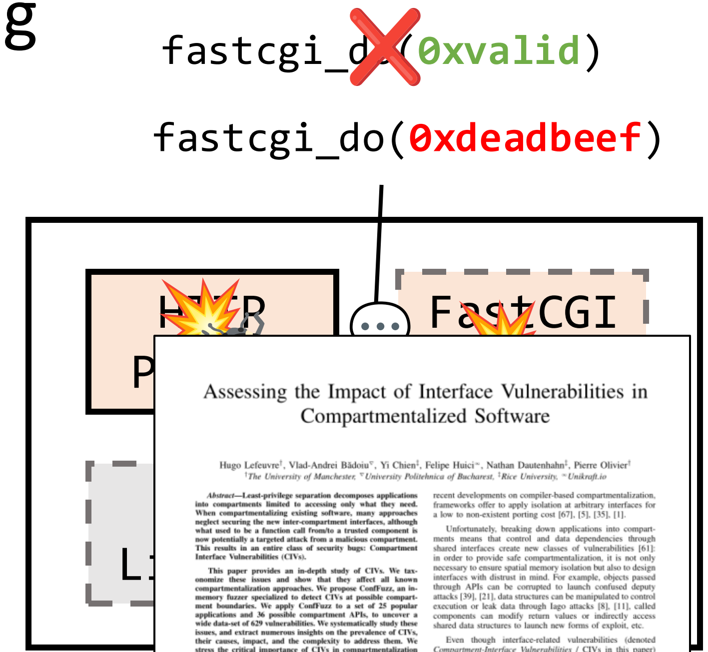
[1] Inherited from thread-like semantics, [2] from process-like semantics, [3] from the Nested Kernel, [4] The PDM does (to a certain extent).
[5] The abstraction could plug onto any intra-AS mechanism, though the paper or documentation claims reliance on a particular one.

Hardening ... **es** is key to obtaining strong sec ... mpartmentalization

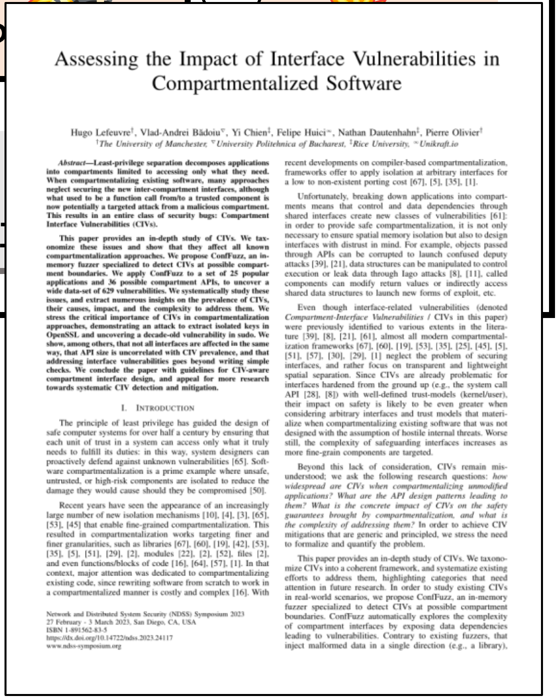
Example: a ... nplemented our boundarie ... *ed to communic* ... ck surface.

Most abstractions consider the **hardening of compartment interfaces** an orthogonal problem.

We stress that abstractions **can and should** facilitate the hardening of compartment interfaces.



(1)



Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Hugo Lefeuvre[†], Vlad-Andrei Bădoiu[▽], Yi Chien[‡], Felipe Huici[~], Nathan Dautenhahn[‡], Pierre Olivier[†]
[†]The University of Manchester, [▽]University Politehnica of Bucharest, [‡]Rice University, [~]Unikraft.io

Abstract—Least-privilege separation decomposes applications into compartments limited to accessing only what they need. When compartmentalizing existing software, many approaches neglect securing the new inter-compartment interfaces, although what used to be a function call from/to a trusted component is now potentially a targeted attack from a malicious compartment. This results in an entire class of security bugs: Compartment Interface Vulnerabilities (CIVs).

recent developments on compiler-based compartmentalization, frameworks offer to apply isolation at arbitrary interfaces for a low to non-existent porting cost [67], [5], [35], [1].

Unfortunately, breaking down applications into compartments means that control and data dependencies through shared interfaces create new classes of vulnerabilities [61]:

How can abstractions that facilitate the implementation of secure domain interfaces?

Open Problem in Compartmentalization Abstractions

Most abstractions consider the **hardening of compartment interfaces** an orthogonal problem.

Some do!

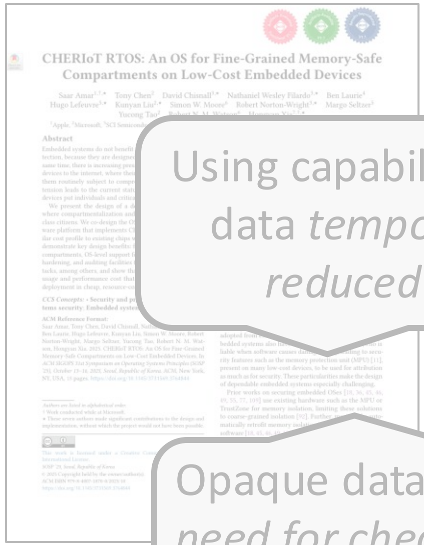
(1)


Retrofitting Fine Grain Isolation in the Firefox Renderer

Leveraging C++ types to ensure all interface-crossing data is checked

Do as many of these checks as possible *automatically*

(2)


CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices

Using capabilities to exchange data *temporarily* and with *reduced permissions*

Opaque data types *eliminate the need for checks* on exported-and-reimported objects

More work needed in that direction.

(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

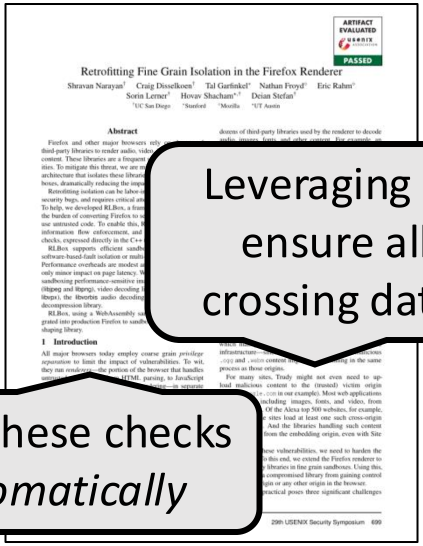
(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Open Problem in Compartmentalization Abstractions

Most abstractions consider the **hardening of compartment interfaces** an orthogonal problem.

Some do!

(1)

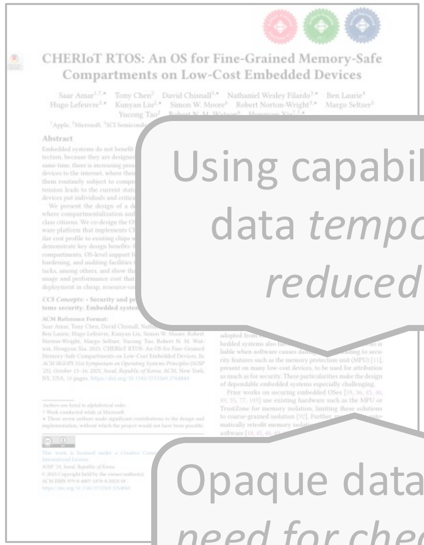

Retrofitting Fine Grain Isolation in the Firefox Renderer

Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm, Sorin Lerner, Hovav Shacham, Deian Stefan

Leveraging C++ types to ensure all interface-crossing data is checked

Do as many of these checks as possible *automatically*

(2)


CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices

Using capabilities to exchange data *temporarily* and with *reduced permissions*

Opaque data types *eliminate the need for checks* on exported-and-reimported objects

More work needed in that direction.

(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

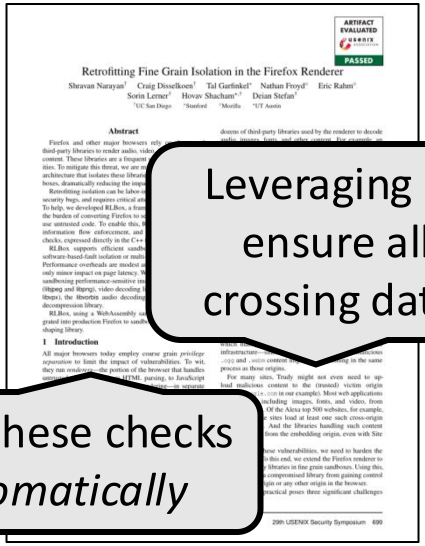
(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Open Problem in Compartmentalization Abstractions

Most abstractions consider the **hardening of compartment interfaces** an orthogonal problem.

Some do!

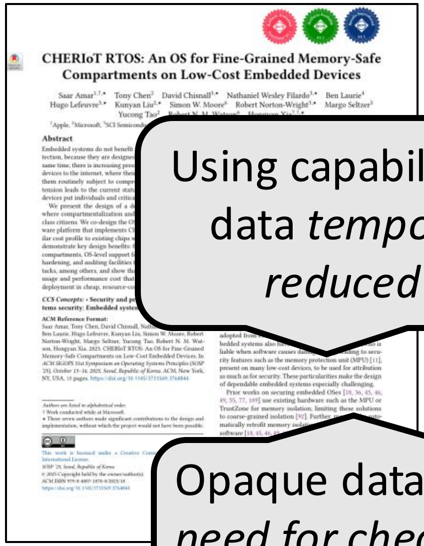
(1)



Leveraging C++ types to ensure all interface-crossing data is checked

Do as many of these checks as possible *automatically*

(2)



Using capabilities to exchange data *temporarily* and with *reduced permissions*

Opaque data types *eliminate the need for checks* on exported-and-reimported objects

More work needed in that direction.

(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

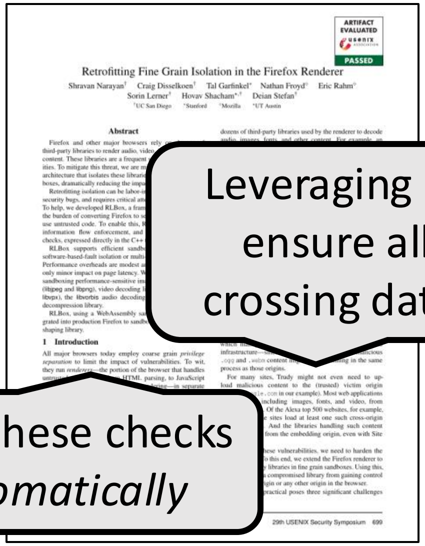
(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

Open Problem in Compartmentalization Abstractions

Most abstractions consider the **hardening of compartment interfaces** an orthogonal problem.

Some do!

(1)

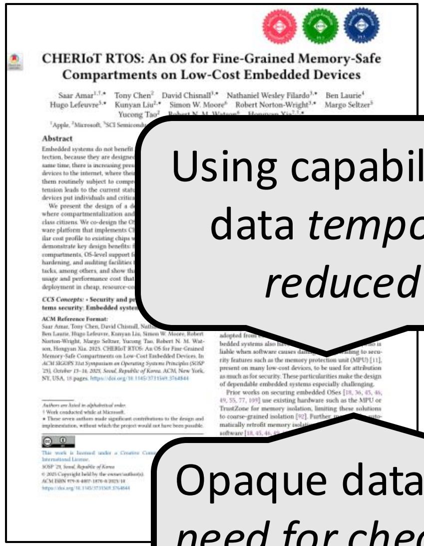


Retrofitting Fine Grain Isolation in the Firefox Renderer

Leveraging C++ types to ensure all interface-crossing data is checked

Do as many of these checks as possible *automatically*

(2)



CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices

Using capabilities to exchange data *temporarily* and with *reduced permissions*

Opaque data types *eliminate the need for checks* on exported-and-reimported objects

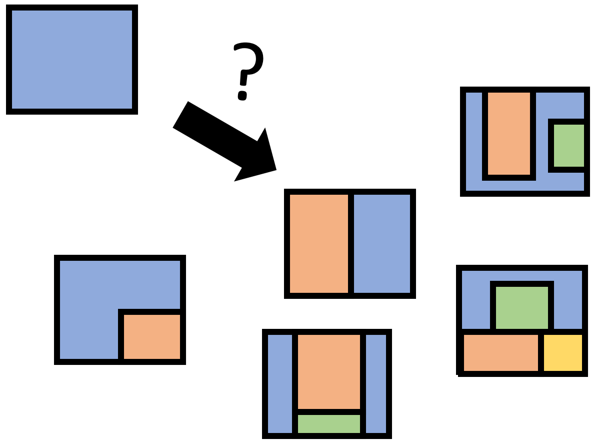
More work needed in that direction.

(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Renderer, USENIX Security 2020

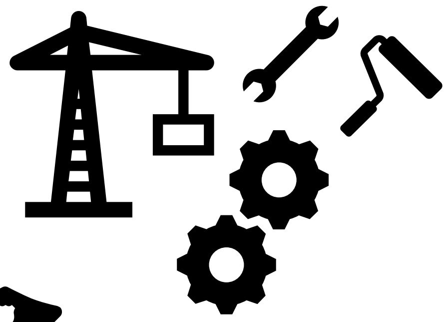
Bigger Picture of Compartmentalization

Propose to view compartmentalization as 3 problems:



1. **How to determine the right policy to enforce?**
   - Done with a *policy definition method*

2. **How to integrate the notion of compartmentalization policies in software / programming models / idioms?**
   - Done with a *compartmentalization abstraction*



3. **How to enforce policies at runtime?**
   - Done with a *compartmentalization mechanism*

Problem #3

How to enforce policies?

Finally, we must **enforce the policy at runtime**.

This is achieved with an **enforcement mechanism**.

Problem #3

How to enforce policies?

Finally, we must **enforce the policy at runtime**.
This is achieved with an **enforcement mechanism**.

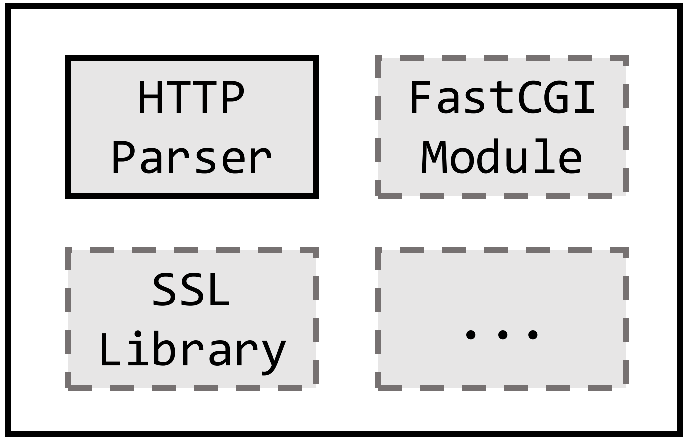
Page tables (via the MMU) are the historical enforcement mechanism.

Problem #3

How to enforce policies?

Finally, we must **enforce the policy at runtime**.
This is achieved with an **enforcement mechanism**.

Page tables (via the MMU) are the historical enforcement mechanism.
Example from earlier: *assume we implemented the sandbox with processes.*



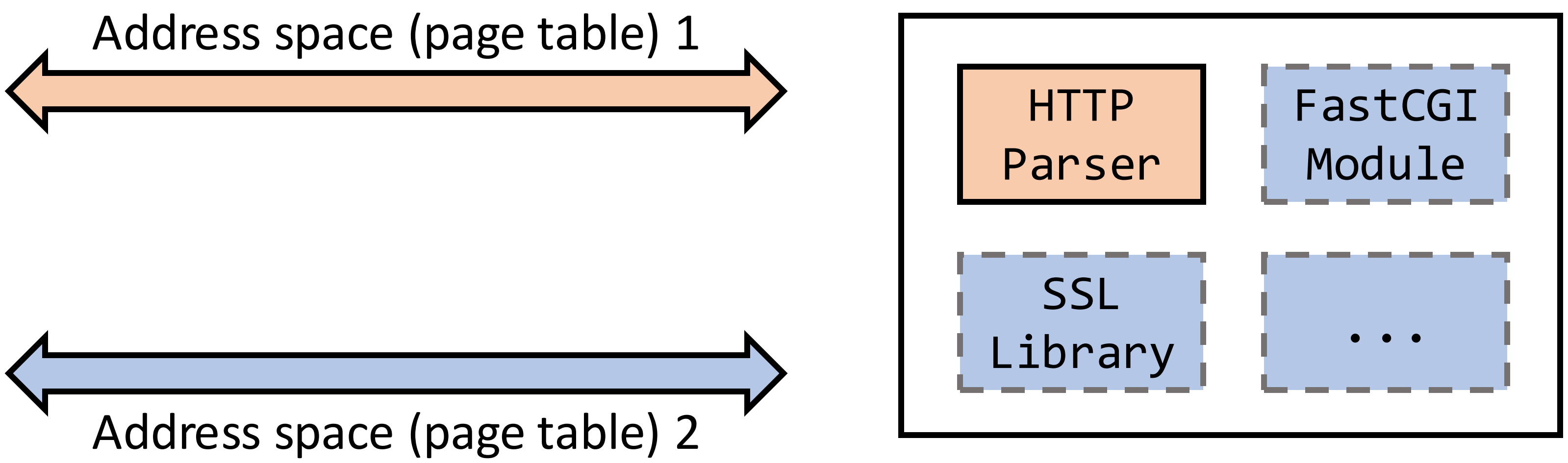
(Fictive monolithic program)

Problem #3

How to enforce policies?

Finally, we must **enforce the policy at runtime**.
This is achieved with an **enforcement mechanism**.

Page tables (via the MMU) are the historical enforcement mechanism.
Example from earlier: *assume we implemented the sandbox with processes.*

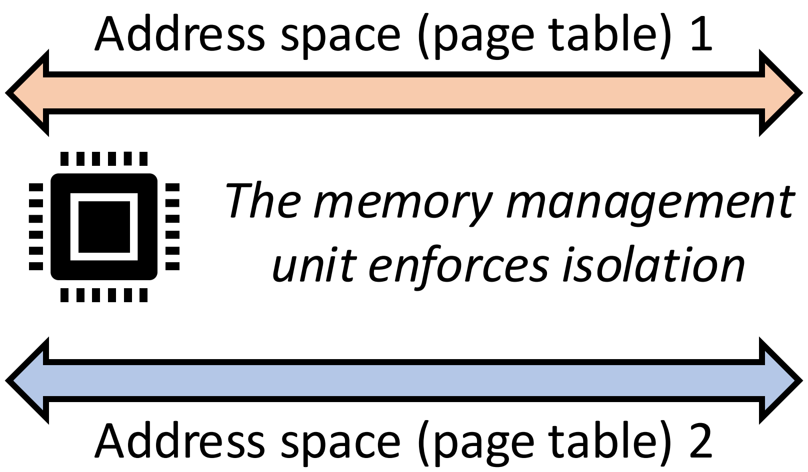


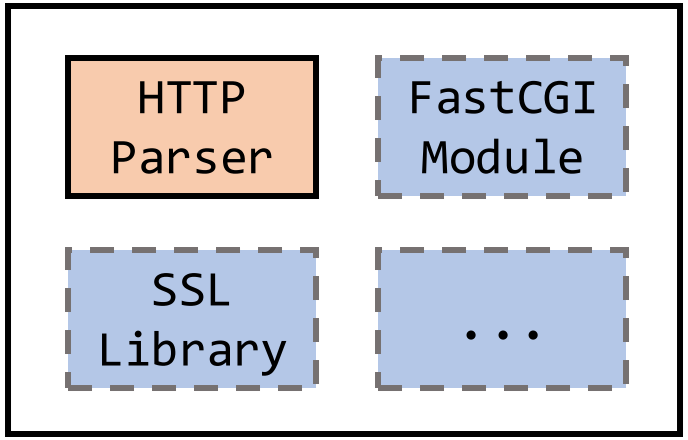
(Fictive monolithic program)

Problem #3
How to enforce policies?

Finally, we must **enforce the policy at runtime**.
This is achieved with an **enforcement mechanism**.

Page tables (via the MMU) are the historical enforcement mechanism.
Example from earlier: *assume we implemented the sandbox with processes.*





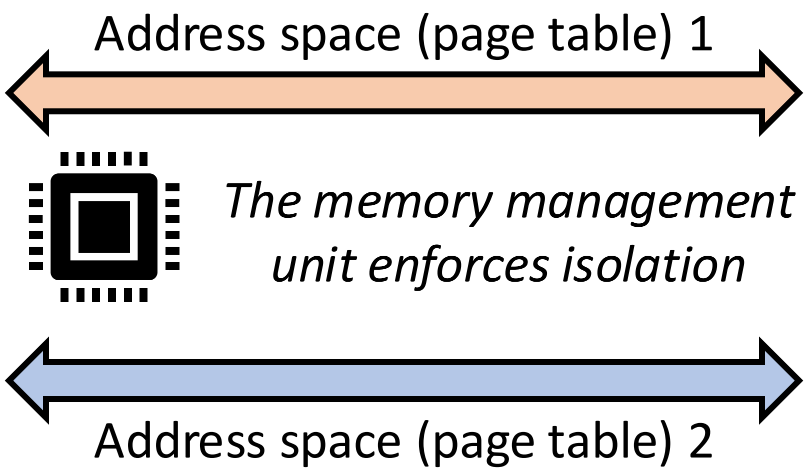
(Fictive monolithic program)

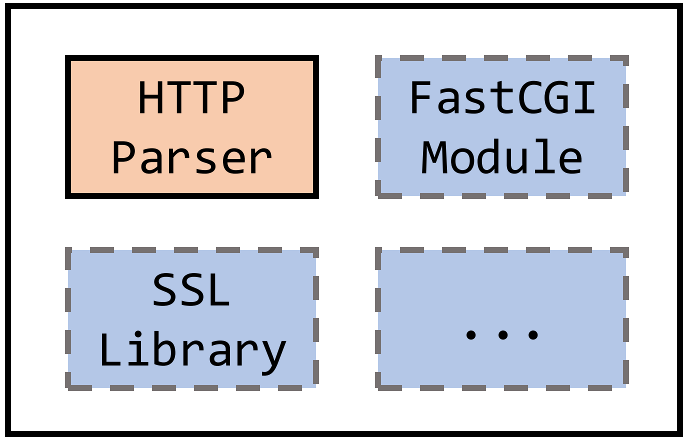
Problem #3

How to enforce policies?

Finally, we must **enforce the policy at runtime**.
This is achieved with an **enforcement mechanism**.

Page tables (via the MMU) are the historical enforcement mechanism.
Example from earlier: *assume we implemented the sandbox with processes.*





(Fictive monolithic program)

This is still the most common way to do it today.

Enforcement Mechanisms

Enforcement mechanisms (for compartmentalization) are also a hot area:

- Maximize performance
- Minimize hardware cost and complexity
- Enable for stronger security properties

(1)

TABLE 3: *Taxonomy of Compartmentalization Mechanisms.* Page-Table = PT; Permissions: Read, Write, Execute, Address (*create pointers to*), ● = supported, ◐ = supported by some, ○ = unsupported; Overhead: *free* = ○ < ◔ < ◓ < ◕ < ● = *very costly*.

		Mechanism Class	Condi-tioned	Trust Model	TCB	Permissions R	W	X	A	Granularity	№ of Domains	Domain Switch Cost (Versus Non-Separated)
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
		Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
		Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
		Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
		Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
		World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Enforcement Mechanisms

Enforcement mechanisms (for compartmentalization) are also a hot area:

- Maximize performance
- Minimize hardware cost and complexity
- Enable for stronger security properties

SoK: Software Compartmentalization

Hugo Lefeuvre, Nathan Dautenhahn, David Chisnall, Pierre Olivier

(1)

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

TABLE 3: *Taxonomy of Compartmentalization Mechanisms.* Page-Table = PT; Permissions: Read, Write, Execute, Address (*create pointers to*), ● = supported, ◐ = supported by some, ○ = unsupported; Overhead: *free* = ○ < ◔ < ◓ < ◕ < ● = *very costly*.

		Mechanism Class	*Conditioned*	*Trust Model*	*TCB*	*Permissions* R	W	X	A	*Granularity*	*№ of Domains*	*Domain Switch Cost (Versus Non-Separated)*
Hardware	Physical Separation [205]		○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware	Segmentation-like Hardware [109, 178]		○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware	Hardware Capabilities [57, 78, 180, 236, 244]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware	Bounds-Checking Hardware [47, 98, 148, 155, 212]		●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]		●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software	Bounds-Checking Software [225]		●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software	Safe Languages [50] / Software Verification [154, 163]		○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]		○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software	Memory Encryption / AES-NI [155]		○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

	Mechanism Class		Conditioned	Trust Model	TCB	Permissions R	Permissions W	Permissions X	Permissions A	Granularity	№ of Domains	Domain Switch Cost (Versus Non-Separated)
Hardware	Physical Separation [205]		○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware	Segmentation-like Hardware [109, 178]		○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware	Hardware Capabilities [57, 78, 180, 236, 244]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware	Bounds-Checking Hardware [47, 98, 148, 155, 212]		●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]		●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16[4] - ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software	Bounds-Checking Software [225]		●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software	Safe Languages [50] / Software Verification [154, 163]		○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]		○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software	Memory Encryption / AES-NI [155]		○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

Names of the mechanisms we consider

Characteristics we included in the taxonomy

		Mechanism Class	Condi-tioned	Trust Model	TCB	Permissions				Granularity	№ of Do	Domain Switch Cost (Versus Non-Separated)
						R	W	X	A			
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

Names of the mechanisms we consider

Characteristics we included in the taxonomy

		Mechanism Class	*Condi-tioned*	*Trust Model*	*TCB*	*Permissions* R	W	X	A	*Granularity*	*№ of D*	*Domain Switch Cost (Versus Non-Separated)*
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

Again, a few interesting aspects 🙂

		Mechanism Class	*Conditioned*	*Trust Model*	*TCB*	*Permissions* R	W	X	A	*Granularity*	*№ of Domains*	*Domain Switch Cost (Versus Non-Separated)*
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16[4] - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

Hardware and software are both popular research areas

Intel MPK

CHERI

WebAssembly

		Mechanism Class	Condi-tioned	Trust Model	TCB	Permissions R	W	X	A	Granularity	№ of Domains	Domain Switch Cost (Versus Non-Separated)
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software		Memory [illegible]ption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [illegible] all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

A lot of interest in enforcing memory isolation at a **byte granularity** (vs. a page)

		Mechanism Class	Conditioned	Trust Model	TCB	Permissions R	W	X	A	Granularity	№ of Domains	Domain Switch Cost (Versus Non-Separated)
Hardware	Physical Separation [205]		○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
	PT	Access Bits [25], EPT / vmfunc [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
	Segmentation-like Hardware [109, 178]		○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
	Hardware Capabilities [57, 78, 180, 236, 244]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
	Bounds-Checking Hardware [47, 98, 148, 155, 212]		●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]		●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Bounds-Checking Software [225]		●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Safe Languages [50] / Software Verification [154, 163]		○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]		○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
	Memory Encryption / AES-NI [155]		○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] S… [5] …crubbing, stack switch.

Very widely varying – 128 bits!

	Mechanism Class	Conditioned	Trust Model	TCB	Permissions R	W	X	A	Granularity	№ of Domains	Domain Switch Cost (Versus Non-Separated)
Hardware	Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT: Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT: Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT: Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT: Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware	Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE: Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE: Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE: World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware	Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware	Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software	Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software	Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software	Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

Nearly all works aim at cutting the **cost of domain switches**

(1) Watson et al., CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization, S&P 2015

(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

Pick one influential example from the literature: **CHERI** (1)(2)

		Mechanism Class	*Conditioned*	*Trust Model*	*TCB*	*Permissions* R	W	X	A	*Granularity*	*№ of Domains*	*Domain Switch Cost (Versus Non-Separated)*
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / vmfunc [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware		**Hardware Capabilities [57, 78, 180, 236, 244]**	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

(1) Watson et al., CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization, S&P 2015

(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

Pick one influential example from the literature: **CHERI** (1)(2)

Hardware mechanism that comes as an ISA extension

		Mechanism Class	Conditioned	Trust Model	TCB	Permissions R	W	X	A	Granularity	№ of Domains	Domain Switch Cost (Versus Non-Separated)
Hardware			○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
		func [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
			○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
		Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
		Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
	Segmentation-like Hardware [109, 178]		○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
	Hardware Capabilities [57, 78, 180, 236, 244]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
	Bounds-Checking Hardware [47, 98, 148, 155, 212]		●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]		●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Bounds-Checking Software [225]		●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Safe Languages [50] / Software Verification [154, 163]		○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]		○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
	Memory Encryption / AES-NI [155]		○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

(1) Watson et al., CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization, S&P 2015
(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

Pick one influential example from the literature: **CHERI** (1)(2)

Hardware mechanism that comes as an ISA extension

CHERI extends pointers with bounds and permission information. It is very *expressive*!

		Mechanism Class	Conditioned	Trust Model	TCB	Permissions R	W	X	A	Granularity	№ of Domains	Domain Switch Cost (Versus Non-Separated)
Hardware		[illegible]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
		[illegible] func [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
		[illegible]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
		Mondrian Memory Protection (MMP) [249]	○	M[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	◓ (MMP hardware + [5])
		Protection Keys [13, 27, 53, 217, 259]	●	M[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible][4]	◔ (special register flip + [5])
	Segmentation-like Hardware [109, 178]		○	Si[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]e)	◔ ([5])
	TEE	Enclaves [28, 92]	○	M[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	● (enclave call, incl. [5])
	TEE	Confidential VMs [10, 29, 30]	○	M[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	● (> EPT switch)
	TEE	World Separation [9, 14]	○	Single	TEE	[illegible]	[illegible]	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
	Hardware Capabilities [57, 78, 180, 236, 244]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
	Bounds-Checking Hardware [47, 98, 148, 155, 212]		●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]		●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Bounds-Checking Software [225]		●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Safe Languages [50] / Software Verification [154, 163]		○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]		○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
	Memory Encryption / AES-NI [155]		○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

(1) Watson et al., CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization, S&P 2015
(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

Pick one influential example from the literature: **CHERI** (1)(2)

		Mechanism Class	*Condi-tioned*	*Trust Model*	*TCB*	*Permissions* R	W	X	A	*Granularity*	*№ of Domains*	*Domain Switch Cost (Versus Non-Separated)*
Hardware			○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
		func [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
			○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
		Mondrian Memory Protection (MMP) [249]	○	M								◓ (MMP hardware + [5])
		Protection Keys [13, 27, 53, 217, 259]	●	M							][4]	◔ (special register flip + [5])
	Segmentation-like Hardware [109, 178]		○	Si							e)	◔ ([5])
	TEE	Enclaves [28, 92]	○	M								● (enclave call, incl. [5])
		Confidential VMs [10, 29, 30]	○	M								● (> EPT switch)
		World Separation [9, 14]	○	Single	TEE			●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
	Hardware Capabilities [57, 78, 180, 236, 244]		○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
	Bounds-Checking Hardware [47, 98, 148, 155, 212]		●	Mutual	Full	●[2]	●[2]	○	○	By	∞	◓ (bounds hardware + [5])
	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]		●	M							- ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]		○	M							∞	◓ – ● (impl. dep., incl. [5])
	Bounds-Checking Software [225]		●	M							∞	◓ – ● (impl. dep., incl. [5])
	Safe Languages [50] / Software Verification [154, 163]		○	Si							fe/unsafe)	○ (function call)
	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]		○	Si							∞	◔ ([5])
	Memory Encryption / AES-NI [155]		○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

Hardware mechanism that comes as an ISA extension

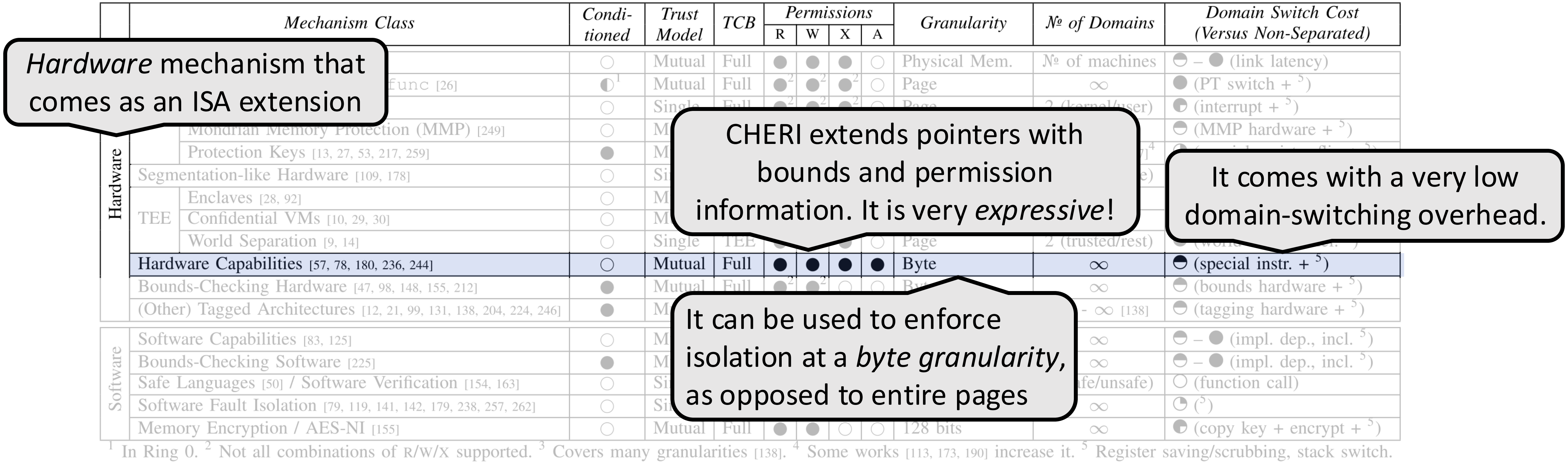
CHERI extends pointers with bounds and permission information. It is very *expressive*!

It can be used to enforce isolation at a *byte granularity*, as opposed to entire pages

(1) Watson et al., CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization, S&P 2015

(2) Amar et al., CHERIoT RTOS: An OS for Fine-Grained Memory-Safe Compartments on Low-Cost Embedded Devices, SOSP 2025

Pick one influential example from the literature: **CHERI** (1)(2)



Open Problem in Enforcement Mechanisms

Mechanisms must fulfill several properties to be suitable to enforce compartmentalization (we formalize them in the SoK)

Some (popular!) mechanisms **do not fulfill all these properties**

		Mechanism Class	*Conditioned*	*Trust Model*	*TCB*	*Permissions* R	W	X	A	*Granularity*	*№ of Domains*	*Domain Switch Cost (Versus Non-Separated)*
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◒ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◒ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◒ (special instr. + [5])
Hardware		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◒ (bounds hardware + [5])
Hardware		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◒ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◒ – ● (impl. dep., incl. [5])
Software		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◒ – ● (impl. dep., incl. [5])
Software		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

How to fairly evaluate the security and performance characteristics of isolation mechanisms?

Open Problem in Enforcement Mechanisms

Mechanisms must fulfill several properties to be suitable to enforce compartmentalization (we formalize them in the SoK)

Some (popular!) mechanisms **do not fulfill all these properties**

We call them *conditioned.*

		Mechanism Class	Condi-tioned	Trust Model	TCB	Permissions R	W	X	A	Granularity	№ of Domains	Domain Switch Co (Versus Non-Separated)
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

How to fairly evaluate the security and performance characteristics of isolation mechanisms?

Open Problem in Enforcement Mechanisms

Mechanisms must fulfill several properties to be suitable to enforce compartmentalization (we formalize them in the SoK)

Some (popular!) mechanisms **do not fulfill all these properties**

We call them *conditioned.*

Intel MPK

		Mechanism Class	*Condi-tioned*	*Trust Model*	*TCB*	*Permissions* R	W	X	A	*Granularity*	*№ of Domains*	*Domain Switch Co (Versus Non-Separated)*
Hardware		Physical Separation [205]	○	Mutual	Full	●	●	●	○	Physical Mem.	№ of machines	◓ – ● (link latency)
Hardware	PT	Access Bits [25], EPT / `vmfunc` [26]	◐[1]	Mutual	Full	●[2]	●[2]	●[2]	○	Page	∞	● (PT switch + [5])
Hardware	PT	Supervisor Bit [25, 158, 159]	○	Single	Full	●[2]	●[2]	●[2]	○	Page	2 (kernel/user)	◕ (interrupt + [5])
Hardware	PT	Mondrian Memory Protection (MMP) [249]	○	Mutual	Full	●[2]	●[2]	●[2]	○	Word	∞	◓ (MMP hardware + [5])
Hardware	PT	Protection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
Hardware		Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
Hardware	TEE	Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
Hardware	TEE	Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
Hardware	TEE	World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
Hardware		Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
Hardware		Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
Hardware		(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software		Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
Software		Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
Software		Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
Software		Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

How to fairly evaluate the security and performance characteristics of isolation mechanisms?

Open Problem in Enforcement Mechanisms

Mechanisms must fulfill several properties to be suitable to enforce compartmentalization (we formalize them in the SoK)

Some (popular!) mechanisms **do not fulfill all these properties**

We call them *conditioned.*

Intel MPK

Access permissions are stored in an unprotected register

										Granularity	*№ of Domains*	*Domain Switch Co… (Versus Non-Separated)*
Hardware	Physical Separation									Physical Mem.	№ of machines	◓ – ● (link latency)
	PT… Access Bits									Page	∞	● (PT switch + [5])
	Supervisor B…									Page	2 (kernel/user)	◕ (interrupt + [5])
	Mondrian Memory Protectio…	○								Word	∞	◓ (MMP hardware + [5])
	…rotection Keys [13, 27, 53, 217, 259]	●	Mutual	Full	●	●	◐	○	Page	8-1024 [13, 217][4]	◔ (special register flip + [5])
	Segmentation-like Hardware [109, 178]	○	Single	Full	●	●	○	○	Byte - Page [178]	2 (safe/unsafe)	◔ ([5])
	TEE Enclaves [28, 92]	○	Mutual	TEE	●	●	●	○	Page	∞	● (enclave call, incl. [5])
	TEE Confidential VMs [10, 29, 30]	○	Mutual	TEE	●	●	●	○	Page	∞	● (> EPT switch)
	TEE World Separation [9, 14]	○	Single	TEE	●	●	●	○	Page	2 (trusted/rest)	● (world switch, incl. [5])
	Hardware Capabilities [57, 78, 180, 236, 244]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ (special instr. + [5])
	Bounds-Checking Hardware [47, 98, 148, 155, 212]	●	Mutual	Full	●[2]	●[2]	○	○	Byte	∞	◓ (bounds hardware + [5])
	(Other) Tagged Architectures [12, 21, 99, 131, 138, 204, 224, 246]	●	Mutual	Full	●[2]	●[2]	○	○	Byte - Words[3]	16^4 - ∞ [138]	◓ (tagging hardware + [5])
Software	Software Capabilities [83, 125]	○	Mutual	Full	●	●	●	●	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Bounds-Checking Software [225]	●	Mutual	Full	●	●	◐	○	Byte	∞	◓ – ● (impl. dep., incl. [5])
	Safe Languages [50] / Software Verification [154, 163]	○	Single	Full	●	●	●	●	Byte	2 (safe/unsafe)	○ (function call)
	Software Fault Isolation [79, 119, 141, 142, 179, 238, 257, 262]	○	Single	Full	●	●	●	●	Byte	∞	◔ ([5])
	Memory Encryption / AES-NI [155]	○	Mutual	Full	●	●	○	○	128 bits	∞	◕ (copy key + encrypt + [5])

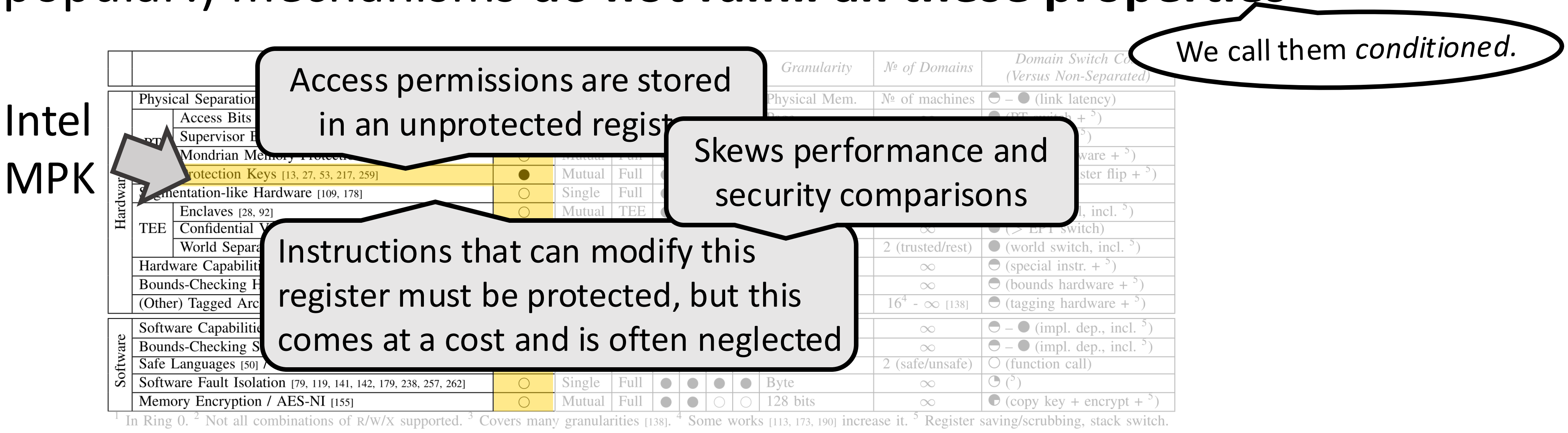
[1] In Ring 0. [2] Not all combinations of R/W/X supported. [3] Covers many granularities [138]. [4] Some works [113, 173, 190] increase it. [5] Register saving/scrubbing, stack switch.

How to fairly evaluate the security and performance characteristics of isolation mechanisms?

Open Problem in Enforcement Mechanisms

Mechanisms must fulfill several properties to be suitable to enforce compartmentalization (we formalize them in the SoK)

Some (popular!) mechanisms **do not fulfill all these properties**

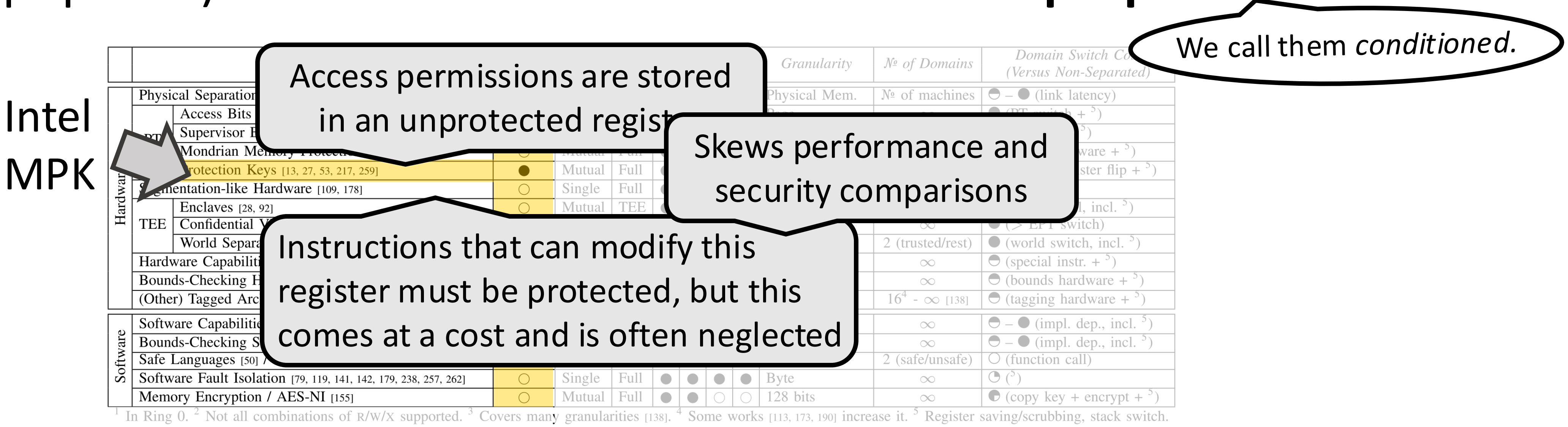


How to fairly evaluate the security and performance characteristics of isolation mechanisms?

Open Problem in Enforcement Mechanisms

Mechanisms must fulfill several properties to be suitable to enforce compartmentalization (we formalize them in the SoK)

Some (popular!) mechanisms **do not fulfill all these properties**



How to fairly evaluate the security and performance characteristics of isolation mechanisms?

My promise for this talk

A journey through twenty years of compartmentalization

1. ~~What is software compartmentalization? (slightly more formal)~~
2. ~~A systematic perspective on compartmentalization~~
3. The *why:* compartmentalization everywhere, what will it take?

What are the remaining obstacles? Where to go from there?

Potential obstacle #1: **Lack of awareness**?

Potential obstacle #1: **Lack of awareness**?

*"Out of the 13 groups of compartmentalized applications we constituted, **8 are exclusively authored by academics or security professionals**"*

SoK: Software Compartmentalization

Hugo Lefeuvre[†1], Nathan Dautenhahn[‡1], David Chisnall[∞], Pierre Olivier[▽]
[†]The University of British Columbia, [‡]Serenitix, [∞]SCI Semiconductor, [▽]The University of Manchester

***Abstract*—Decomposing large systems into smaller components with limited privileges has long been recognized as an effective means to minimize the impact of exploits. Despite historical roots, demonstrated benefits, and a plethora of research efforts in academia and industry, the compartmentalization of software is still not a mainstream practice. This paper investigates why, and how this status quo can be improved. Noting that existing approaches are fraught with inconsistencies in terminology and analytical methods, we propose a unified model for the systematic analysis, comparison, and directing of compartmentalization approaches. We use this model to review 211 research efforts and analyze 61 mainstream compartmentalized systems, confronting them to understand the limitations of both research and production works. Among others, our findings reveal that mainstream efforts largely rely on manual methods, custom abstractions, and legacy mechanisms, poles apart from recent research. We conclude with recommendations: compartmentalization should be solved holistically; progress is needed towards simplifying the definition of compartmentalization policies; towards better challenging our threat models in the light of confused deputies and hardware limitations; as well as towards bridging the gaps we pinpoint between research and mainstream needs. This paper not only maps the historical and current landscape of compartmentalization, but also sets forth a framework to foster their evolution and adoption.**

1. Introduction

Despite decades of effort, vulnerabilities still plague software, and thwarting them remains a game of cat and mouse. The *principle of least privilege* (PoLP) [209] is the last line of defense when protections fail or when flaws are unknown. By granting each entity only the privileges needed, the PoLP ensures that a compromise of one part will not imply that of the whole. *Software compartmentalization* is a prominent implementation of the PoLP, in which developers divide a large program into smaller, lesser privileged components, to reduce the impact of potential security breaches.

Software compartmentalization inherits from a large body of work, starting with processes [93]; including OS models such as microkernels [106, 126, 150, 164, 258], security and separation kernels [56, 58, 205, 206, 247], or capability OSes [83, 125]; all the way to fine-grain application compartmentalization in the 2000s [147, 181, 232, 238] following qmail [68], Postfix [121], or OpenSSH [198]. Its promises are plenty: containing memory safety issues [72, 202], untrusted third parties [54, 181], or unsafe parts of safe languages [62, 149, 174, 201, 228], isolating cryptographic secrets [167, 171, 232] or shadow stacks [74], thwarting supply-chain attacks [111, 235] and side-channels [137, 176, 182], or providing fault resilience [164, 184].

Despite longstanding recognition within the academic sphere and proven effectiveness in seminal industry projects, the adoption of compartmentalization techniques in mainstream software remains inconsistent: compartmentalizing is still far from being a common engineering practice. Take as example the isolation of cryptographic secrets, a practice long advocated by the community [65, 70, 73, 87, 115–117, 140, 158, 165, 167, 211, 226, 232] but without viable adoption by leading cryptography libraries. At a time of growing threats, we are missing out on the security benefits compartmentalization can bring. This paper investigates the reasons behind this status quo, and how to improve it.

Research speculated that this situation is due to a lack of automation [65, 253], limitations of mechanisms [190, 259], excessive performance overheads [188, 232], a lack of strong security guarantees [129, 160], among others. All are likely part of the problem, and through decades of research progress was made on every front. Today the community lacks a global overview of this progress. Designing for or retrofitting compartmentalization is often hampered by inconsistencies in the understanding and application of its concepts. Existing models and terminologies are numerous, ad-hoc, and often contradictory, leading to confusion and a growing body of work that cannot be compared. The lack of a systematic perspective leads to a mismatch between what software compartmentalization needs to progress towards the mainstream, and the focus and framing of research efforts: most do not tackle compartmentalization's key aspects as a whole and thus produce solutions that cannot be relevant to making it a mainstream practice.

Recognizing these challenges, we propose a unified model providing a consistent framework for defining, understanding, and implementing compartmentalization. This model comes with a comprehensive taxonomy that allows to classify compartmentalization strategies based on their policy definition methods, abstractions, and mechanisms, providing a basis for systematic evaluation and comparison.

We validate our model and taxonomy by systematizing 211 research and 61 mainstream software systems implementing compartmentalization. Doing so, this study provides unique insights into where mainstream efforts have

[1]This work was primarily done while Hugo Lefeuvre was affiliated with the University of Manchester and SCI Semiconductor, and Nathan Dautenhahn with Rice University (with minor contributions at Riverside Research).

(1)

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Potential obstacle #1: **Lack of awareness**?

"Out of the 13 groups of compartmentalized applications we constituted, ***8 are exclusively authored by academics or security professionals****"*

SoK: Software Compartmentalization

Hugo Lefeuvre[†1], Nathan Dautenhahn[‡1], David Chisnall[∞], Pierre Olivier[▽]
[†]*The University of British Columbia,* [‡]*Serenitix,* [∞]*SCI Semiconductor,* [▽]*The University of Manchester*

Abstract—**Decomposing large systems into smaller components with limited privileges has long been recognized as an effective means to minimize the impact of exploits. Despite historical roots, demonstrated benefits, and a plethora of research efforts in academia and industry, the compartmentalization of software is still not a mainstream practice. This paper investigates why, and how this status quo can be improved. Noting that existing approaches are fraught with inconsistencies in terminology and analytical methods, we propose a unified model for the systematic analysis, comparison, and directing of compartmentalization approaches. We use this model to review 211 research efforts and analyze 61 mainstream compartmentalized systems, confronting them to understand the limitations of both research and production works. Among others, our findings reveal that mainstream efforts largely rely on manual methods, custom abstractions, and legacy mechanisms, poles apart from recent research. We conclude with recommendations: compartmentalization should be solved holistically; progress is needed towards simplifying the definition of compartmentalization policies; towards better challenging our threat models in the light of confused deputies and hardware limitations; as well as towards bridging the gaps we pinpoint between research and mainstream needs. This paper not only maps the historical and current landscape of compartmentalization, but also sets forth a framework to foster their evolution and adoption.**

1. Introduction

Despite decades of effort, vulnerabilities still plague software, and thwarting them remains a game of cat and mouse. The *principle of least privilege* (PoLP) [209] is the last line of defense when protections fail or when flaws are unknown. By granting each entity only the privileges needed, the PoLP ensures that a compromise of one part will not imply that of the whole. *Software compartmentalization* is a prominent implementation of the PoLP, in which developers divide a large program into smaller, lesser privileged components, to reduce the impact of potential security breaches.

Software compartmentalization inherits from a large body of work, starting with processes [93]; including OS models such as microkernels [106, 126, 150, 164, 258], security and separation kernels [56, 58, 205, 206, 247], or capability OSes [83, 125]; all the way to fine-grain application compartmentalization in the 2000s [147, 181, 232, 238] following qmail [68], Postfix [121], or OpenSSH [198]. Its promises are plenty: containing memory safety issues [72, 202], untrusted third parties [54, 181], or unsafe parts of safe languages [62, 149, 174, 201, 228], isolating cryptographic secrets [167, 171, 232] or shadow stacks [74], thwarting supply-chain attacks [111, 235] and side-channels [137, 176, 182], or providing fault resilience [164, 184].

Despite longstanding recognition within the academic sphere and proven effectiveness in seminal industry projects, the adoption of compartmentalization techniques in mainstream software remains inconsistent: compartmentalizing is still far from being a common engineering practice. Take as example the isolation of cryptographic secrets, a practice long advocated by the community [65, 70, 73, 87, 115–117, 140, 158, 165, 167, 211, 226, 232] but without viable adoption by leading cryptography libraries. At a time of growing threats, we are missing out on the security benefits compartmentalization can bring. This paper investigates the reasons behind this status quo, and how to improve it.

Research speculated that this situation is due to a lack of automation [65, 253], limitations of mechanisms [190, 259], excessive performance overheads [188, 232], a lack of strong security guarantees [129, 160], among others. All are likely part of the problem, and through decades of research progress was made on every front. Today the community lacks a global overview of this progress. Designing for or retrofitting compartmentalization is often hampered by inconsistencies in the understanding and application of its concepts. Existing models and terminologies are numerous, ad-hoc, and often contradictory, leading to confusion and a growing body of work that cannot be compared. The lack of a systematic perspective leads to a mismatch between what software compartmentalization needs to progress towards the mainstream, and the focus and framing of research efforts: most do not tackle compartmentalization's key aspects as a whole and thus produce solutions that cannot be relevant to making it a mainstream practice.

Recognizing these challenges, we propose a unified model providing a consistent framework for defining, understanding, and implementing compartmentalization. This model comes with a comprehensive taxonomy that allows to classify compartmentalization strategies based on their policy definition methods, abstractions, and mechanisms, providing a basis for systematic evaluation and comparison.

We validate our model and taxonomy by systematizing 211 research and 61 mainstream software systems implementing compartmentalization. Doing so, this study provides unique insights into where mainstream efforts have

[1]This work was primarily done while Hugo Lefeuvre was affiliated with the University of Manchester and SCI Semiconductor, and Nathan Dautenhahn with Rice University (with minor contributions at Riverside Research).

(1)

We barely teach it at the university, not part of common "software design patterns"

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Potential obstacle #1: **Lack of awareness**?

*"Out of the 13 groups of compartmentalized applications we constituted, **8 are exclusively authored by academics or security professionals**"*

We barely teach it at the university, not part of common "software design patterns"

SoK: Software Compartmentalization

Hugo Lefeuvre[†1], Nathan Dautenhahn[‡1], David Chisnall[∞], Pierre Olivier[▽]
[†]The University of British Columbia, [‡]Serenitix, [∞]SCI Semiconductor, [▽]The University of Manchester

***Abstract*—Decomposing large systems into smaller components with limited privileges has long been recognized as an effective means to minimize the impact of exploits. Despite historical roots, demonstrated benefits, and a plethora of research efforts in academia and industry, the compartmentalization of software is still not a mainstream practice. This paper investigates why, and how this status quo can be improved. Noting that existing approaches are fraught with inconsistencies in terminology and analytical methods, we propose a unified model for the systematic analysis, comparison, and directing of compartmentalization approaches. We use this model to review 211 research efforts and analyze 61 mainstream compartmentalized systems, confronting them to understand the limitations of both research and production works. Among others, our findings reveal that mainstream efforts largely rely on manual methods, custom abstractions, and legacy mechanisms, poles apart from recent research. We conclude with recommendations: compartmentalization should be solved holistically; progress is needed towards simplifying the definition of compartmentalization policies; towards better challenging our threat models in the light of confused deputies and hardware limitations; as well as towards bridging the gaps we pinpoint between research and mainstream needs. This paper not only maps the historical and current landscape of compartmentalization, but also sets forth a framework to foster their evolution and adoption.**

1. Introduction

Despite decades of effort, vulnerabilities still plague software, and thwarting them remains a game of cat and mouse. The *principle of least privilege* (PoLP) [209] is the last line of defense when protections fail or when flaws are unknown. By granting each entity only the privileges needed, the PoLP ensures that a compromise of one part will not imply that of the whole. *Software compartmentalization* is a prominent implementation of the PoLP, in which developers divide a large program into smaller, lesser privileged components, to reduce the impact of potential security breaches.

Software compartmentalization inherits from a large body of work, starting with processes [93]; including OS models such as microkernels [106, 126, 150, 164, 258], security and separation kernels [56, 58, 205, 206, 247], or capability OSes [83, 125]; all the way to fine-grain application compartmentalization in the 2000s [147, 181, 232, 238] following qmail [68], Postfix [121], or OpenSSH [198]. Its promises are plenty: containing memory safety issues [72, 202], untrusted third parties [54, 181], or unsafe parts of safe languages [62, 149, 174, 201, 228], isolating cryptographic secrets [167, 171, 232] or shadow stacks [74], thwarting supply-chain attacks [111, 235] and side-channels [137, 176, 182], or providing fault resilience [164, 184].

Despite longstanding recognition within the academic sphere and proven effectiveness in seminal industry projects, the adoption of compartmentalization techniques in mainstream software remains inconsistent: compartmentalizing is still far from being a common engineering practice. Take as example the isolation of cryptographic secrets, a practice long advocated by the community [65, 70, 73, 87, 115–117, 140, 158, 165, 167, 211, 226, 232] but without viable adoption by leading cryptography libraries. At a time of growing threats, we are missing out on the security benefits compartmentalization can bring. This paper investigates the reasons behind this status quo, and how to improve it.

Research speculated that this situation is due to a lack of automation [65, 253], limitations of mechanisms [190, 259], excessive performance overheads [188, 232], a lack of strong security guarantees [129, 160], among others. All are likely part of the problem, and through decades of research progress was made on every front. Today the community lacks a global overview of this progress. Designing for or retrofitting compartmentalization is often hampered by inconsistencies in the understanding and application of its concepts. Existing models and terminologies are numerous, ad-hoc, and often contradictory, leading to confusion and a growing body of work that cannot be compared. The lack of a systematic perspective leads to a mismatch between what software compartmentalization needs to progress towards the mainstream, and the focus and framing of research efforts: most do not tackle compartmentalization's key aspects as a whole and thus produce solutions that cannot be relevant to making it a mainstream practice.

Recognizing these challenges, we propose a unified model providing a consistent framework for defining, understanding, and implementing compartmentalization. This model comes with a comprehensive taxonomy that allows to classify compartmentalization strategies based on their policy definition methods, abstractions, and mechanisms, providing a basis for systematic evaluation and comparison.

We validate our model and taxonomy by systematizing 211 research and 61 mainstream software systems implementing compartmentalization. Doing so, this study provides unique insights into where mainstream efforts have

[1]This work was primarily done while Hugo Lefeuvre was affiliated with the University of Manchester and SCI Semiconductor, and Nathan Dautenhahn with Rice University (with minor contributions at Riverside Research).

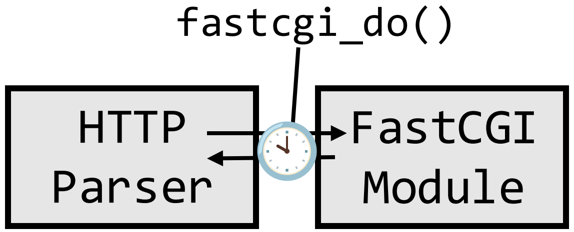
(1)

Yes, but...

(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

Potential obstacle #2: **Performance**?

Potential obstacle #2: **Performance**?



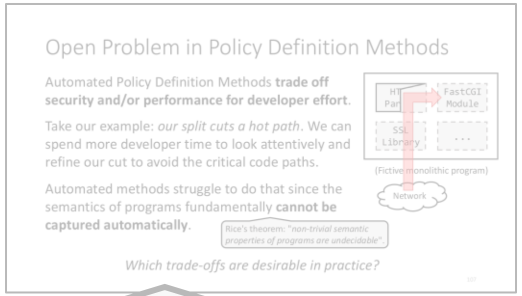
Compartmentalization has an **impact on performance**

- Crossing protection domains or IPC is not free (among others)
- This cost varies based on *where* crossings are in the control flow

This makes compartmentalization more complex and limits the extent to which one can compartmentalize

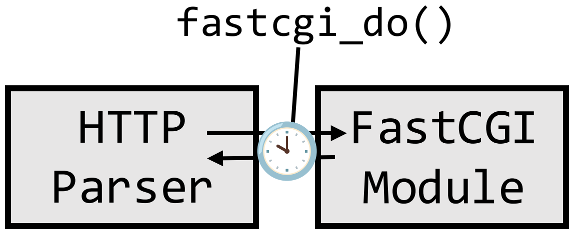
Critical stance: **in most cases, this is not a problem.**

- The community is overly focused on performance
- The cost is reasonable in most cases if the compartmentalization is done correctly
- In fact, many performance-sensitive programs are compartmentalized!



Remember our policy definition challenge...

Potential obstacle #2: **Performance**?



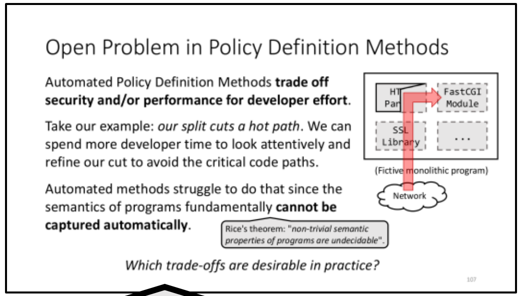
Compartmentalization has an **impact on performance**

- Crossing protection domains or IPC is not free (among others)
- This cost varies based on *where* crossings are in the control flow

This makes compartmentalization more complex and limits the extent to which one can compartmentalize

Critical stance: **in most cases, this is not a problem.**

- The community is overly focused on performance
- The cost is reasonable in most cases if the compartmentalization is done correctly
- In fact, many performance-sensitive programs are compartmentalized!



Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.

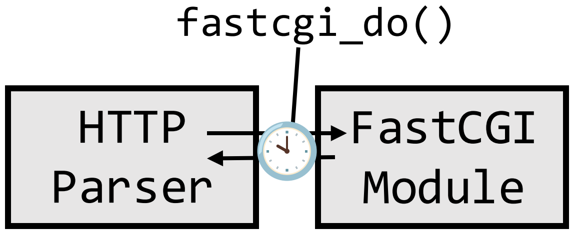
Rice's theorem: "*non-trivial semantic properties of programs are undecidable*".

Which trade-offs are desirable in practice?

107

Remember our policy definition challenge...

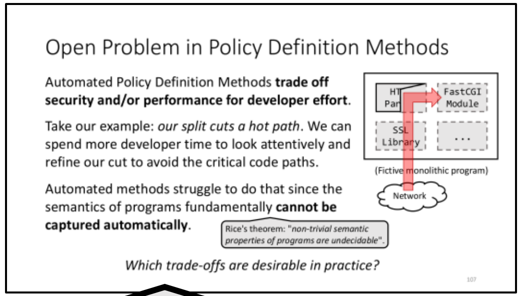
Potential obstacle #2: **Performance**?



Compartmentalization has an **impact on performance**

- Crossing protection domains or IPC is not free (among others)
- This cost varies based on *where* crossings are in the control flow

This makes compartmentalization more complex and limits the extent to which one can compartmentalize



Open Problem in Policy Definition Methods

Automated Policy Definition Methods **trade off security and/or performance for developer effort**.

Take our example: *our split cuts a hot path*. We can spend more developer time to look attentively and refine our cut to avoid the critical code paths.

Automated methods struggle to do that since the semantics of programs fundamentally **cannot be captured automatically**.

Rice's theorem: *"non-trivial semantic properties of programs are undecidable"*.

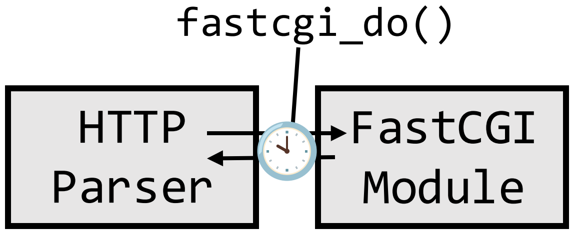
Which trade-offs are desirable in practice?

Remember our policy definition challenge...

Critical stance: **in most cases, this is not a problem.**

- The community is overly focused on performance
- The cost is reasonable in most cases if the compartmentalization is done correctly
- In fact, many performance-sensitive programs are compartmentalized!

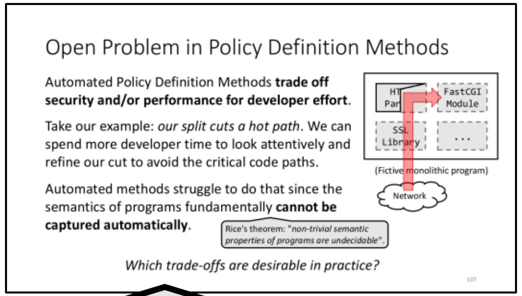
Potential obstacle #2: **Performance**?



Compartmentalization has an **impact on performance**

- Crossing protection domains or IPC is not free (among others)
- This cost varies based on *where* crossings are in the control flow

This makes compartmentalization more complex and limits the extent to which one can compartmentalize

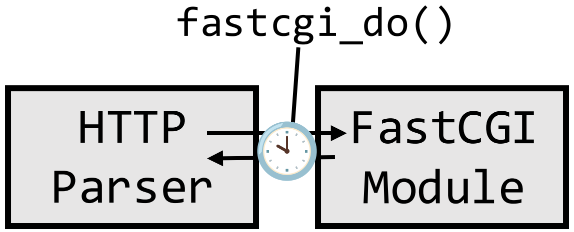


Remember our policy definition challenge...

Critical stance: **in most cases, this is not a problem.**

- The community is overly focused on performance
- The cost is reasonable in most cases if the compartmentalization is done correctly
- In fact, many performance-sensitive programs are compartmentalized!

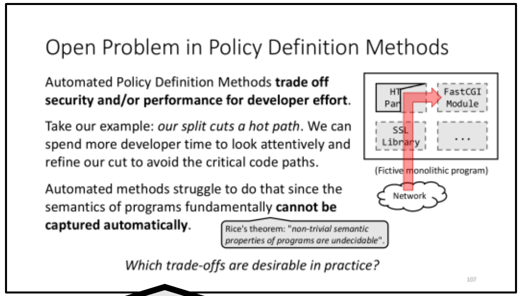
Potential obstacle #2: **Performance**?



Compartmentalization has an **impact on performance**

- Crossing protection domains or IPC is not free (among others)
- This cost varies based on *where* crossings are in the control flow

This makes compartmentalization more complex and limits the extent to which one can compartmentalize



Remember our policy definition challenge...

Critical stance: **in most cases, this is not a problem.**

- The community is overly focused on performance
- The cost is reasonable in most cases if the compartmentalization is done correctly
- In fact, many performance-sensitive programs are compartmentalized!

Yes, but...

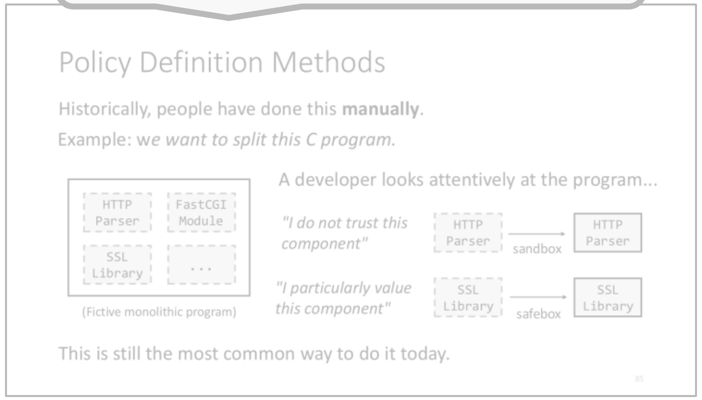
Potential obstacle #3: **Complexity**!

Potential obstacle #3: **Complexity**!

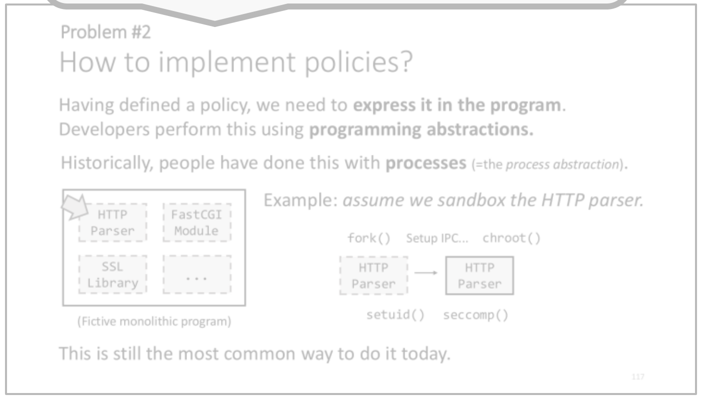
The historical approach to compartmentalizing software is **too complex**.

Going back to our example:

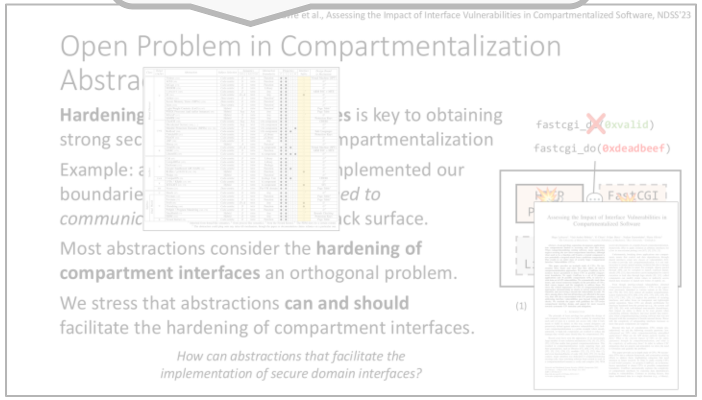
Determining boundaries



Implementing them with low-level process APIs



Securing them with ad-hoc audits

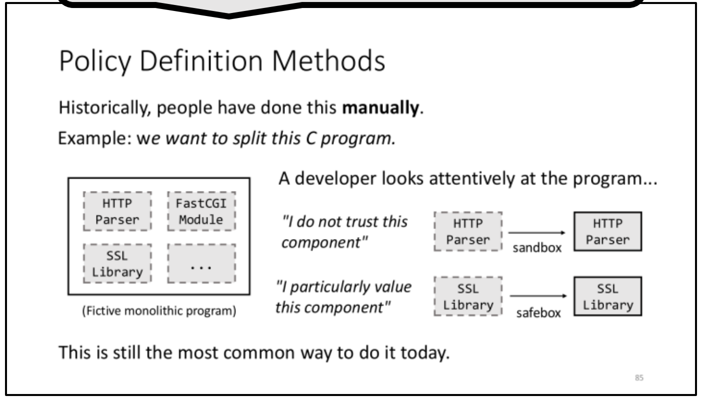


Potential obstacle #3: **Complexity**!

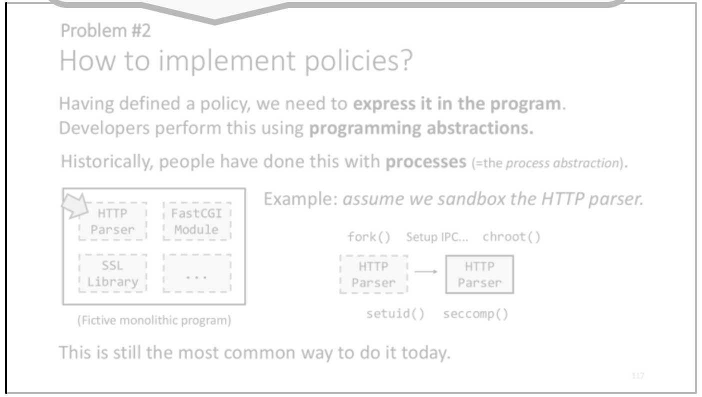
The historical approach to compartmentalizing software is **too complex**.

Going back to our example:

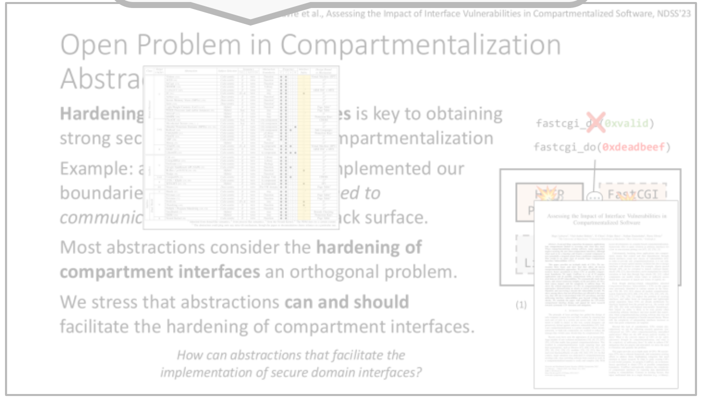
Determining boundaries



Implementing them with low-level process APIs



Securing them with ad-hoc audits

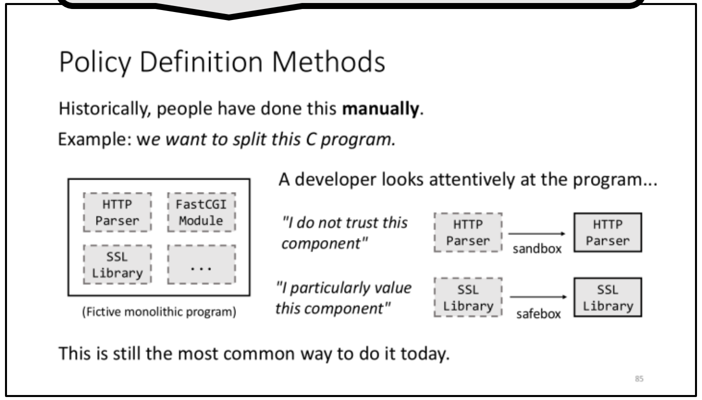


Potential obstacle #3: **Complexity**!

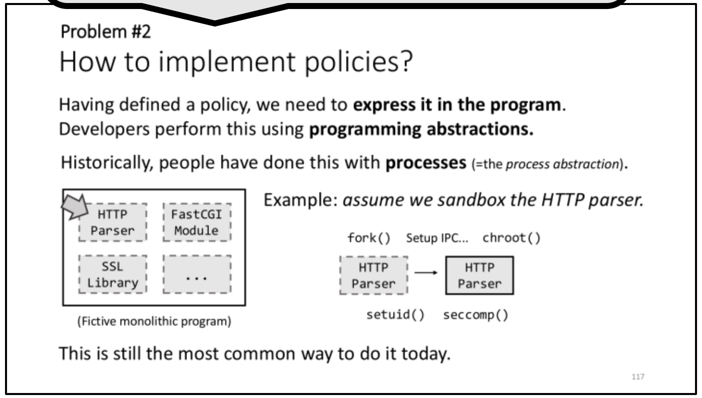
The historical approach to compartmentalizing software is **too complex**.

Going back to our example:

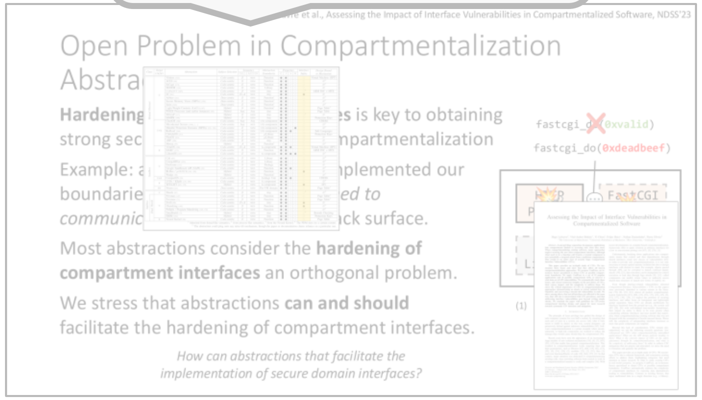
Determining boundaries



Implementing them with low-level process APIs



Securing them with ad-hoc audits

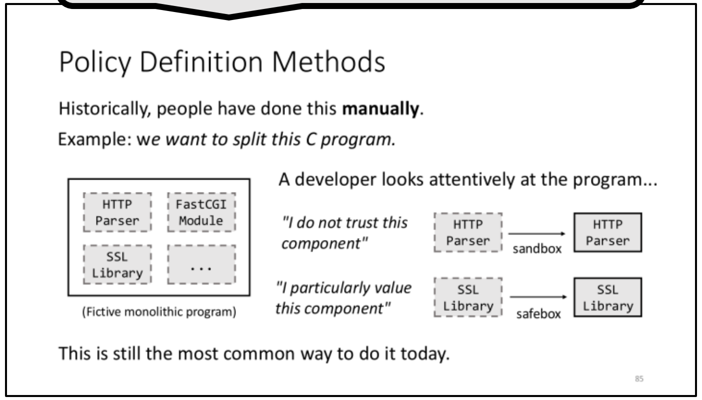


Potential obstacle #3: **Complexity**!

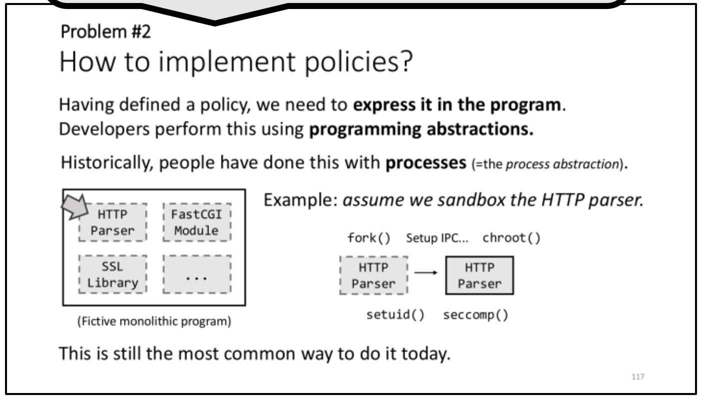
The historical approach to compartmentalizing software is **too complex**.

Going back to our example:

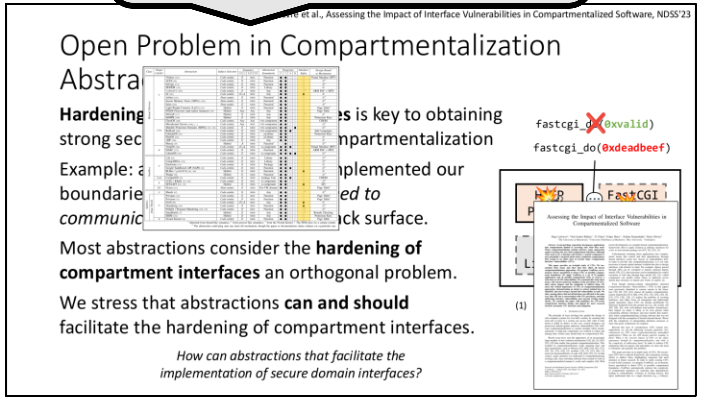
Determining boundaries



Implementing them with low-level process APIs



Securing them with ad-hoc audits



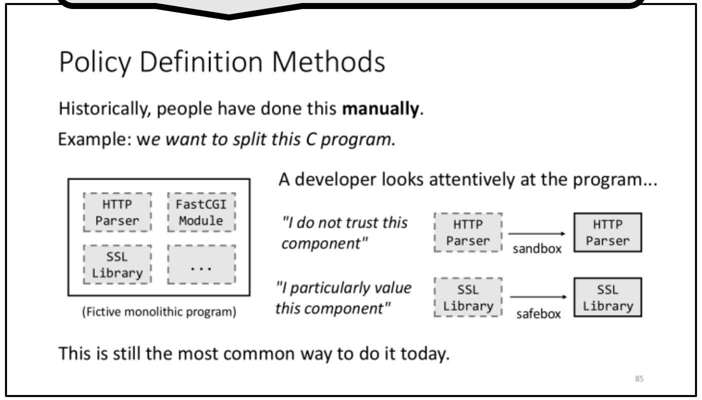
This approach is ad-hoc, costly in developer time, and requires a lot of expertise.

Potential obstacle #3: **Complexity**!

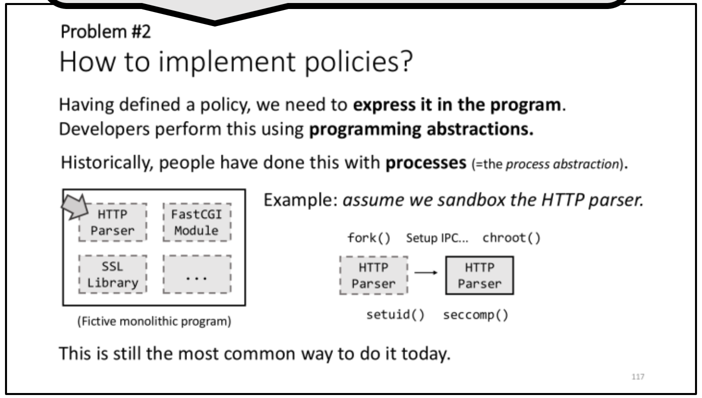
The historical approach to compartmentalizing software is **too complex**.

Going back to our example:

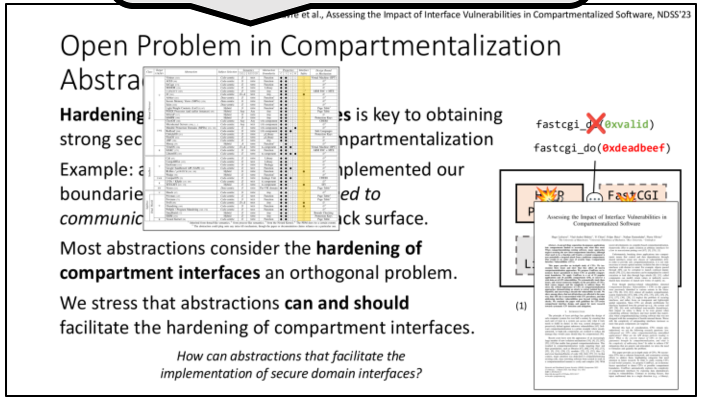
Determining boundaries



Implementing them with low-level process APIs



Securing them with ad-hoc audits

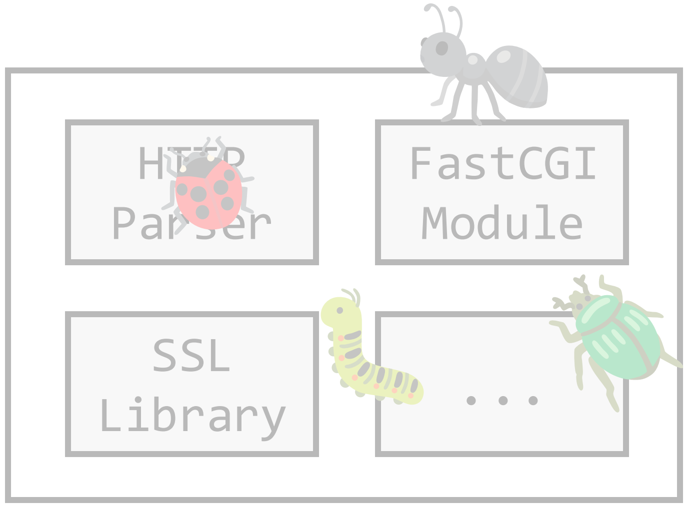


This approach is ad-hoc, costly in developer time, and requires a lot of expertise.

Potential obstacle #3: **Complexity**!

The historical approach to compartmentalizing software is **too complex**.

Worse yet, it considers compartmentalization as a one-off operation, **which it isn't**.



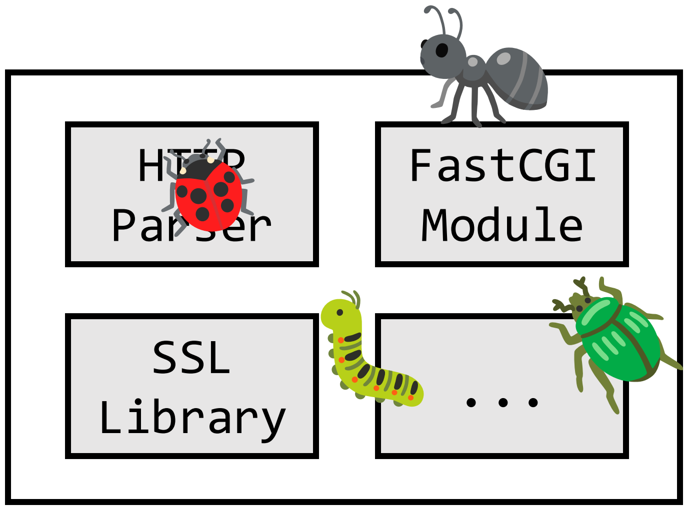
Consider debugging:

- *There will be bugs to fix*.
- Bugs now occur across security boundaries.
- This impacts the debugging experience but the **debugging ecosystem is not there**.

Potential obstacle #3: **Complexity**!

The historical approach to compartmentalizing software is **too complex**.

Worse yet, it considers compartmentalization as a one-off operation, **which it isn't**.



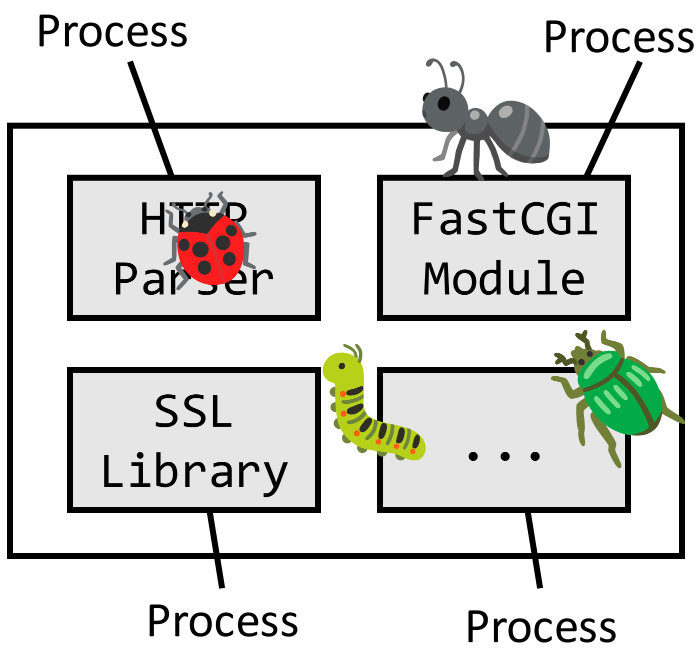
<u>Consider debugging:</u>

- *There will be bugs to fix*.
- Bugs now occur across security boundaries.
- This impacts the debugging experience but the **debugging ecosystem is not there**.

Potential obstacle #3: **Complexity**!

The historical approach to compartmentalizing software is **too complex**.

Worse yet, it considers compartmentalization as a one-off operation, **which it isn't**.



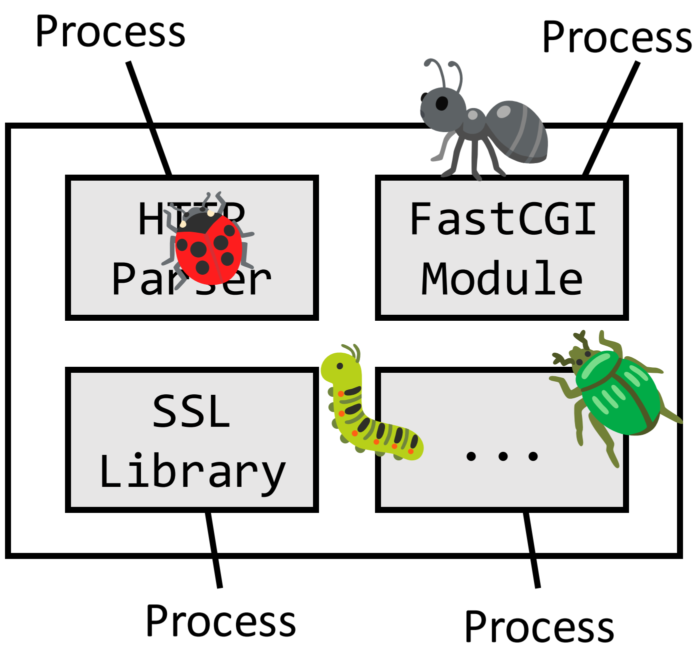
Consider debugging:

- *There will be bugs to fix*.
- Bugs now occur across security boundaries.
- This impacts the debugging experience but the **debugging ecosystem is not there**.

Potential obstacle #3: **Complexity**!

The historical approach to compartmentalizing software is **too complex**.

Worse yet, it considers compartmentalization as a one-off operation, **which it isn't**.



Consider debugging:

- *There will be bugs to fix*.
- Bugs now occur across security boundaries.
- This impacts the debugging experience but the **debugging ecosystem is not there**.

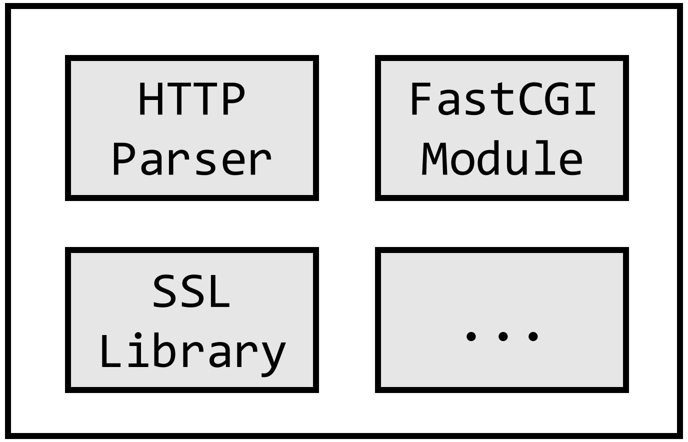
Potential obstacle #3: **Complexity**!

The historical approach to compartmentalizing software is **too complex**.

Worse yet, it considers compartmentalization as a one-off operation, **which it isn't**.

Consider maintenance!

- Developers will need to maintain the partitioned program.
- Over time, the **partitioning strategy may need to change**.
- Again, there is historically **no tooling** to handle that safely and avoid regressions.



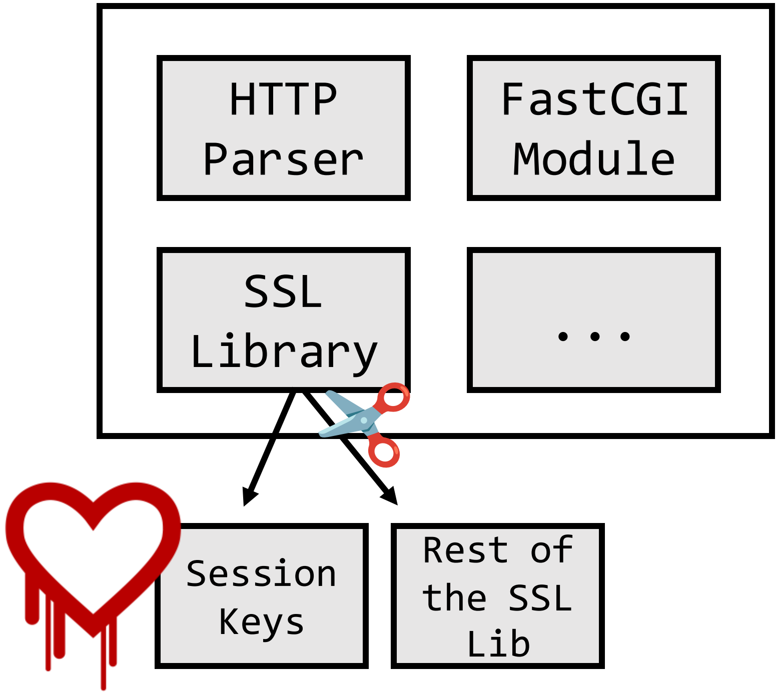
Potential obstacle #3: **Complexity**!

The historical approach to compartmentalizing software is **too complex**.

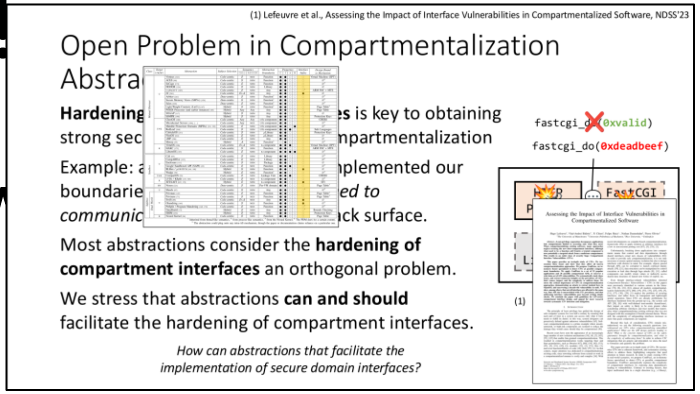
Worse yet, it considers compartmentalization as a one-off operation, **which it isn't**.

Consider maintenance!

- Developers will need to maintain the partitioned program.
- Over time, the **partitioning strategy may need to change**.
- Again, there is historically **no tooling** to handle that safely and avoid regressions.



Potential obstacle #3: **Complexity!**

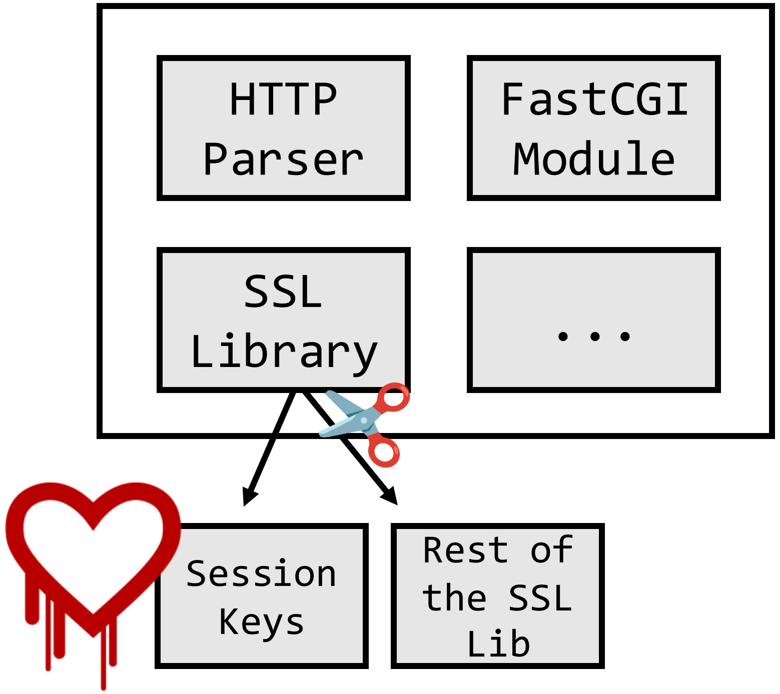


The historical approach to compartmentalizing softw

Worse yet, it considers compartmentalization as a one-off operation, **which it isn't**.

<u>Consider maintenance!</u>

- Developers will need to maintain the partitioned program.
- Over time, the **partitioning strategy may need to change**.
- Again, there is historically **no tooling** to handle that safely and avoid regressions.



Potential obstacle #4: **The lack of a systematic approach**?

Potential obstacle #4: **The lack of a systematic approach**?

The historical approach to compartmentalization does not scale:

- Historically, compartmentalization is deeply **custom to each program**
- Compartmentalize program A, *then start all over again with B*

Can we make compartmentalization efforts reusable?

- Can we compartmentalize component *C*, and reuse that component compartmentalized everywhere?
- Analogy to the "software crisis" from the 1960s:
  - *The answer to "we are not producing enough software" is to build software from reusable components*[1]

(1) https://www.cs.dartmouth.edu/~doug/components.txt

Potential obstacle #4: **The lack of a systematic approach**?

The historical approach to compartmentalization does not scale:

- Historically, compartmentalization is deeply **custom to each program**
- Compartmentalize program A, *then start all over again with B*

Can we make compartmentalization efforts reusable?

- Can we compartmentalize component *C*, and reuse that component compartmentalized everywhere?
- Analogy to the "software crisis" from the 1960s:
  - *The answer to "we are not producing enough software" is to build software from reusable components*[1]

(1) `https://www.cs.dartmouth.edu/~doug/components.txt`

Potential obstacle #4: **The lack of a systematic approach**?

The historical approach to compartmentalization does not scale:

- Historically, compartmentalization is deeply **custom to each program**
- Compartmentalize program A, *then start all over again with B*

Can we make compartmentalization efforts reusable?

- Can we compartmentalize component *C*, and reuse that component compartmentalized everywhere?
- Analogy to the "software crisis" from the 1960s:
  - *The answer to "we are not producing enough software" is to build software from reusable components*[1]

(1) `https://www.cs.dartmouth.edu/~doug/components.txt`

A golden age

It is the right *time* to push compartmentalization everywhere

(1)

- More than ever **we need more secure software**

> The level of **cyber threat keeps on increasing** (...). Today, ANSSI assesses that attackers linked to China, Russia and cybercrime pose the **greatest threat to the most critical networks and to the French ecosystem in a systemic way**.

*emphasis mine

(1) `https://www.cert.ssi.gouv.fr/uploads/CERTFR-2024-CTI-002.pdf`

It is the right *time* to push compartmentalization everywhere

- More than ever **we need more secure software**
- Compartmentalization **absolutely fits the needs**

(1)

The level of **cyber threat keeps on increasing** (…).
Today, ANSSI assesses that attackers linked to
… d cybercrime pose the **greatest**
… **st critical networks and to the**
… **m in a systemic way**.

*emphasis mine

(2)

Secure Hardware Foundation: Incorporate architectural features that **enable fine-grained memory protection**, such as those described by Capability Hardware Enhanced RISC Instructions (CHERI)

Co-signed by, among others, UK, NL, DE, NO

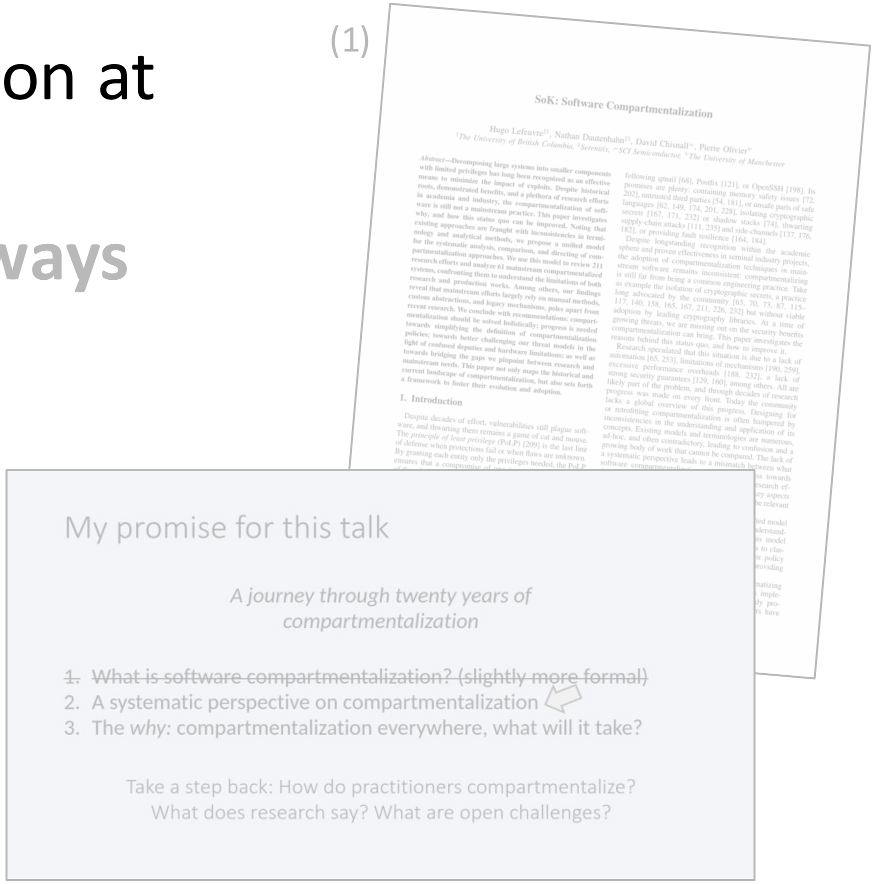
(2) https://www.cisa.gov/sites/default/files/2023-10/SecureByDesign_1025_508c.pdf

(1) https://www.cert.ssi.gouv.fr/uploads/CERTFR-2024-CTI-002.pdf

It is the right *time* to push compartmentalization everywhere

- The challenges to push compartmentalization at scale are **not new**:
  - Research has **explored them in many ways** and come up with **solutions that work** (second part of this talk!)
  - These advances are just asking to be **developed and mainstreamed**

(1)



(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

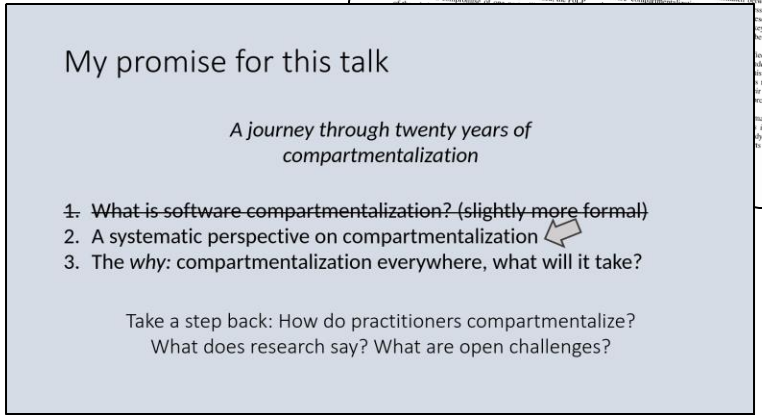
It is the right *time* to push compartmentalization everywhere

- The challenges to push compartmentalization at scale are **not new**:
  - Research has **explored them in many ways** and come up with **solutions that work** (second part of this talk!)
  - These advances are just asking to be **developed and mainstreamed**

(1)

SoK: Software Compartmentalization

Hugo Lefeuvre[†1], Nathan Dautenhahn[‡1], David Chisnall[≃], Pierre Olivier[▽]
[†]The University of British Columbia, [‡]Serenitix, [≃]SCI Semiconductor, [▽]The University of Manchester



(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

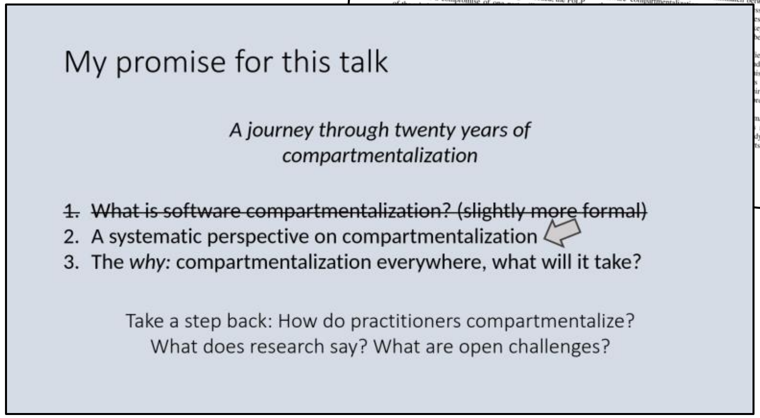
It is the right *time* to push compartmentalization everywhere

- The challenges to push compartmentalization at scale are **not new**:
  - Research has **explored them in many ways** and come up with **solutions that work** (second part of this talk!)
  - These advances are just asking to be **developed and mainstreamed**

(1)

SoK: Software Compartmentalization

Hugo Lefeuvre, Nathan Dautenhahn, David Chisnall, Pierre Olivier



(1) Lefeuvre et al., SoK: Software Compartmentalization, S&P 2025

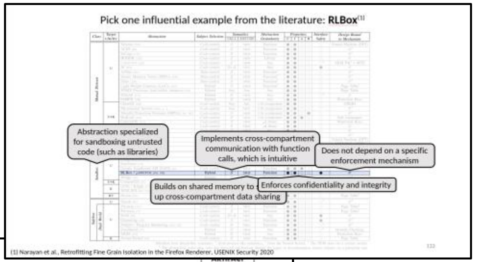
The missing frameworks are emerging

Take RLBox, discussed earlier in the talk

- Productized in Firefox for library isolation
- You can use it: https://github.com/PLSysSec/rlbox

Recently: increasing interest in coming together to create industry standards for compartmentalization

- The *Open Robust Compartmentalization Alliance* was just accepted as Linux Foundation project: https://github.com/ORCA-LF/governance
- Aiming to foster the adoption and standardization of compartmentalization practices



(1)


Retrofitting Fine Grain Isolation in the Firefox Renderer

Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm, Sorin Lerner, Hovav Shacham, Deian Stefan

Abstract

1 Introduction

USENIX Association — 29th USENIX Security Symposium 699

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Render, USENIX Security 2020

The missing frameworks are emerging

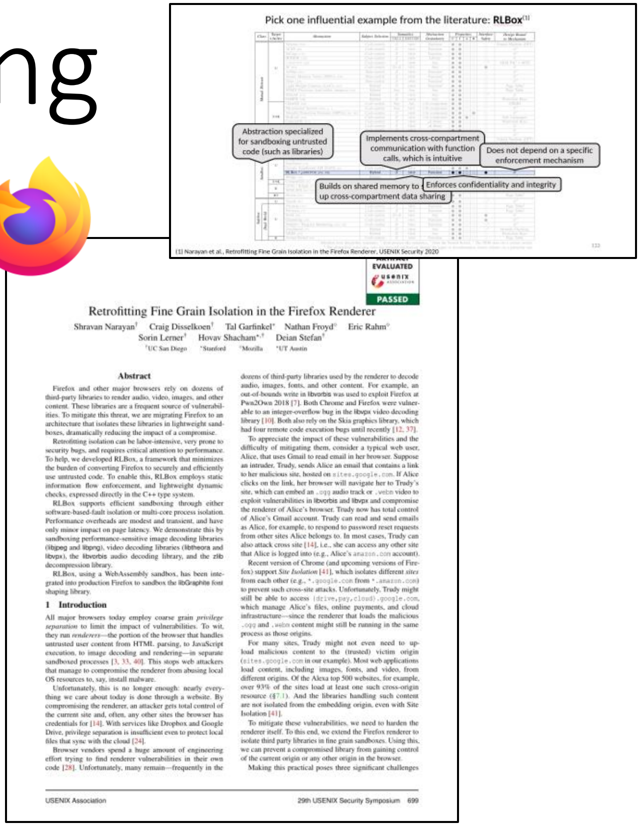
Take RLBox, discussed earlier in the talk

- Productized in Firefox for library isolation
- You can use it: https://github.com/PLSysSec/rlbox

Recently: increasing interest in coming together to create industry standards for compartmentalization

- The *Open Robust Compartmentalization Alliance* was just accepted as Linux Foundation project: https://github.com/ORCA-LF/governance
- Aiming to foster the adoption and standardization of compartmentalization practices

(1)

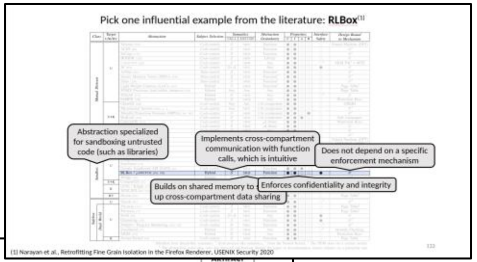


(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Render, USENIX Security 2020

The missing frameworks are emerging

Take RLBox, discussed earlier in the talk

- Productized in Firefox for library isolation
- You can use it: https://github.com/PLSysSec/rlbox



(1)

Retrofitting Fine Grain Isolation in the Firefox Renderer

Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm, Sorin Lerner, Hovav Shacham, Deian Stefan

Abstract

1 Introduction

USENIX Association — 29th USENIX Security Symposium 699

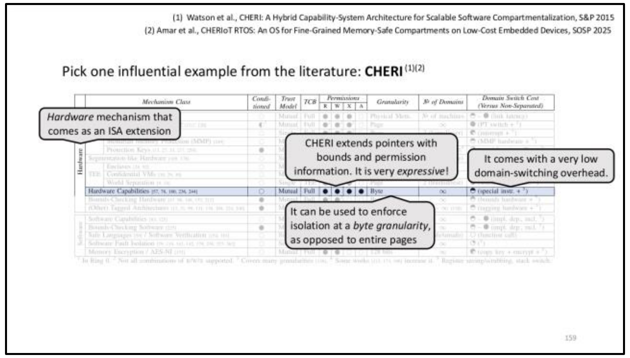
Recently: increasing interest in coming together to create industry standards for compartmentalization

- The *Open Robust Compartmentalization Alliance* was just accepted as Linux Foundation project: https://github.com/ORCA-LF/governance
- Aiming to foster the adoption and standardization of compartmentalization practices

(1) Narayan et al., Retrofitting Fine Grain Isolation in the Firefox Render, USENIX Security 2020

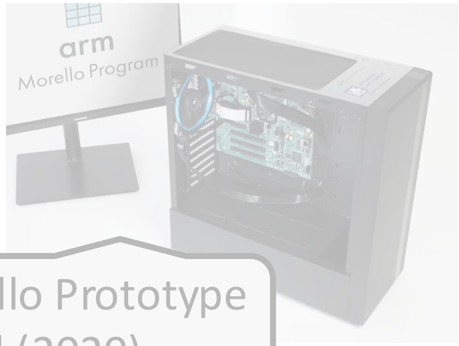
Join us! Still in early stage, you can send me a message.

The hardware ecosystem is emerging

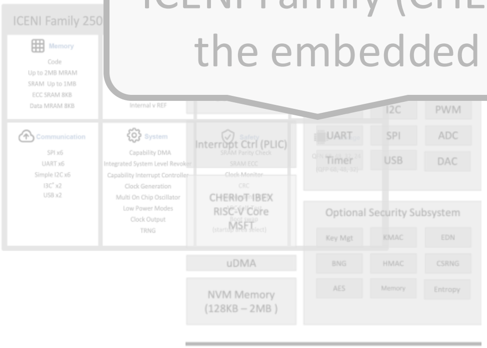


Take CHERI, discussed earlier as well.

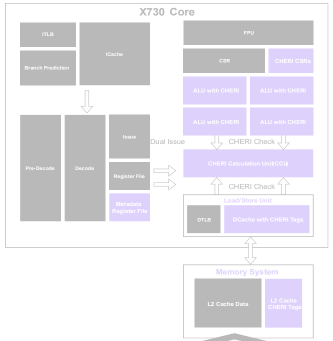
It is not a research prototype. You can use it!



ARM Morello Prototype Board (2020)



ICENI Family (CHERIoT): CHERI for the embedded world (2025)



Codasip X730 (2024)

Recently: creation of the *CHERI Alliance* to support industry standards and foster adoption

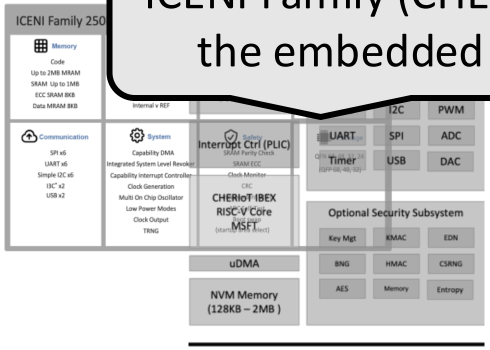
- *Join the CHERI Alliance!* https://cheri-alliance.org/

The hardware ecosystem is emerging

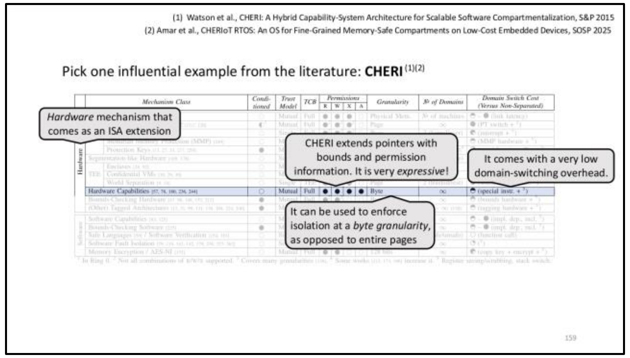
Take CHERI, discussed earlier as well.

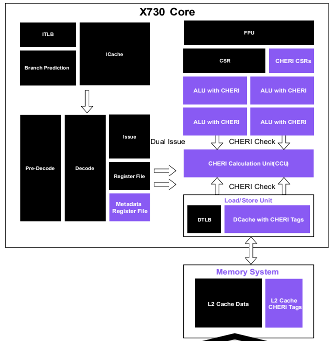
It is not a research prototype. You can use it!

ARM Morello Prototype Board (2020)



ICENI Family (CHERIoT): CHERI for the embedded world (2025)





Codasip X730 (2024)

Recently: creation of the *CHERI Alliance* to support industry standards and foster adoption

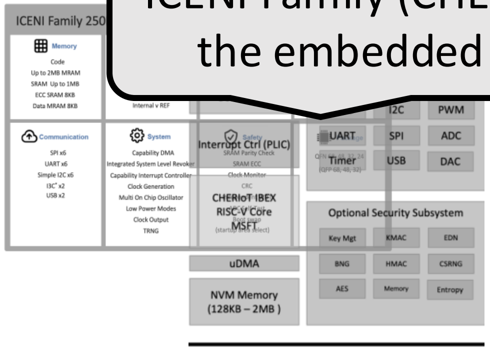
- *Join the CHERI Alliance!* https://cheri-alliance.org/

The hardware ecosystem is emerging

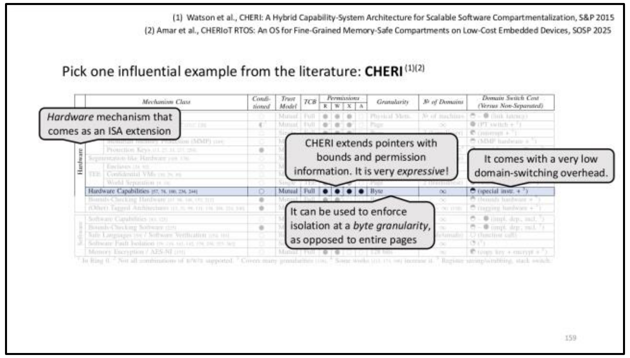
Take CHERI, discussed earlier as well.

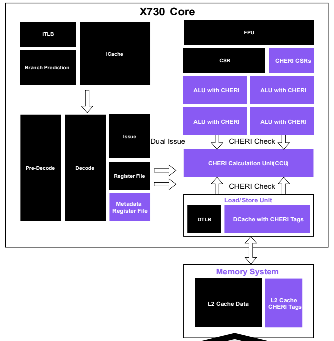
It is not a research prototype. You can use it!

ARM Morello Prototype Board (2020)



ICENI Family (CHERIoT): CHERI for the embedded world (2025)



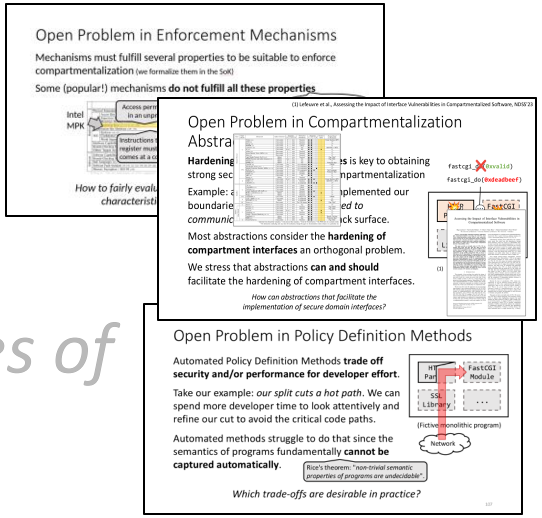


Codasip X730 (2024)

Recently: creation of the *CHERI Alliance* to support industry standards and foster adoption

- *Join the CHERI Alliance!* https://cheri-alliance.org/

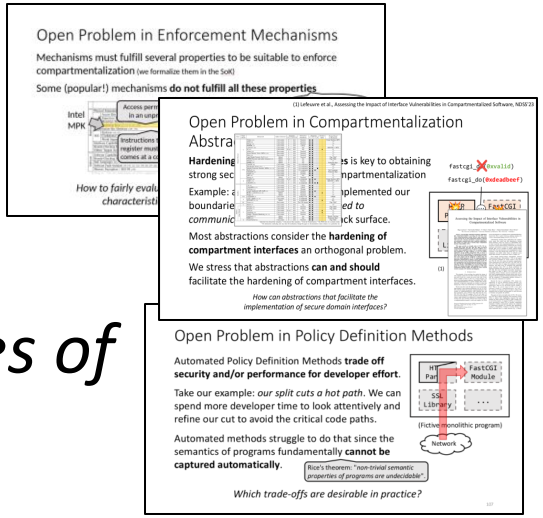
Software compartmentalization everywhere: *what will it take?*



Yes, there are many interesting research challenges left.

But for the better part, it is about *transitioning two decades of research into the industry and open-source communities*

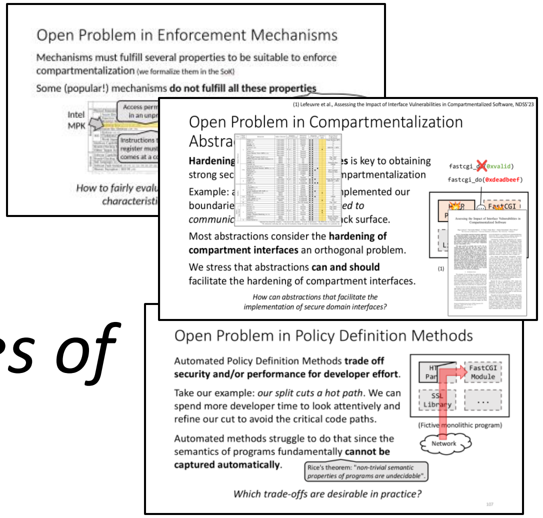
1. An effort on **industry-grade compartmentalization frameworks** to lower the bar to compartmentalize software
2. An effort on using these frameworks to **compartmentalize vulnerable components in a re-usable fashion**: can we do this under the umbrella of Linux distributions?
3. Supporting the push towards **emerging security-oriented hardware**: integrate CHERI in your hardware products!



Yes, there are many interesting research challenges left.

But for the better part, it is about *transitioning two decades of research into the industry and open-source communities*

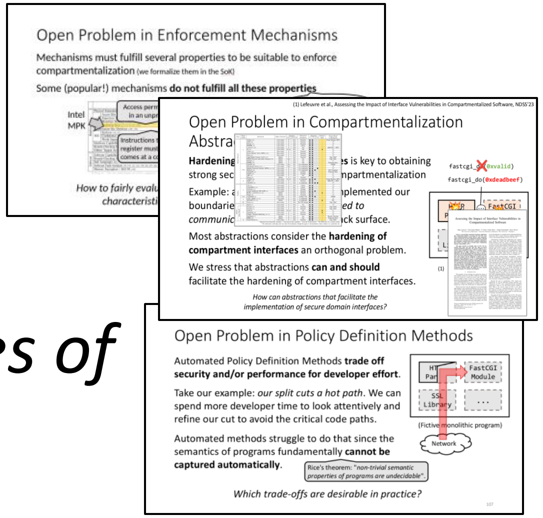
1. An effort on **industry-grade compartmentalization frameworks** to lower the bar to compartmentalize software
2. An effort on using these frameworks to **compartmentalize vulnerable components in a re-usable fashion**: can we do this under the umbrella of Linux distributions?
3. Supporting the push towards **emerging security-oriented hardware**: integrate CHERI in your hardware products!



Yes, there are many interesting research challenges left.

But for the better part, it is about *transitioning two decades of research into the industry and open-source communities*

1. An effort on **industry-grade compartmentalization frameworks** to lower the bar to compartmentalize software
2. An effort on using these frameworks to **compartmentalize vulnerable components in a re-usable fashion**: can we do this under the umbrella of Linux distributions?
3. Supporting the push towards **emerging security-oriented hardware**: integrate CHERI in your hardware products!



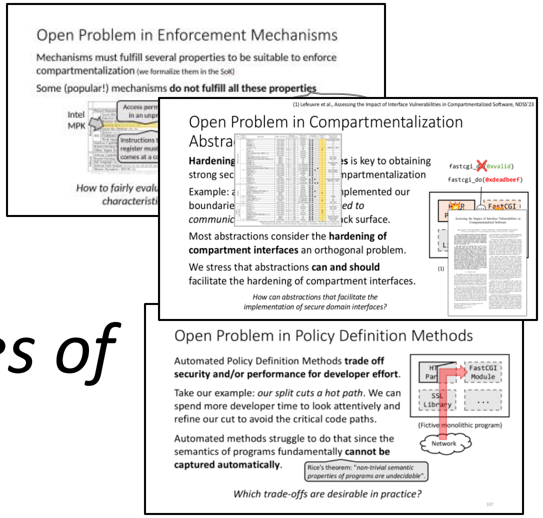
Yes, there are many interesting research challenges left.

But for the better part, it is about *transitioning two decades of research into the industry and open-source communities*

1. An effort on **industry-grade compartmentalization frameworks** to lower the bar to compartmentalize software
2. An effort on using these frameworks to **compartmentalize vulnerable components in a re-usable fashion**: can we do this under the umbrella of Linux distributions?
3. Supporting the push towards **emerging security-oriented hardware**: integrate CHERI in your hardware products!

Yes, there are many interesting research challenges left.

But for the better part, it is about *transitioning two decades of research into the industry and open-source communities*



1. An effort on **industry-grade compartmentalization frameworks** to lower the bar to compartmentalize software
2. An effort on using these frameworks to **compartmentalize vulnerable components in a re-usable fashion**: can we do this under the umbrella of Linux distributions?
3. Supporting the push towards **emerging security-oriented hardware**: integrate CHERI in your hardware products!

THE UNIVERSITY OF BRITISH COLUMBIA

Key Takeaways

1. Compartmentalization is a key practice to **improve the trustworthiness of software**
2. Compartmentalization, in its historical form, is **hindered by its complexity**
3. We live at a **golden age to solve this problem**. A call to the community: we can make software more trustworthy!

THE UNIVERSITY OF BRITISH COLUMBIA

Key Takeaways

1. Compartmentalization is a key practice to **improve the trustworthiness of software**
2. Compartmentalization, in its historical form, is **hindered by its complexity**
3. We live at a **golden age to solve this problem**. A call to the community: we can make software more trustworthy!

THE UNIVERSITY OF BRITISH COLUMBIA

Key Takeaways

1. Compartmentalization is a key practice to **improve the trustworthiness of software**
2. Compartmentalization, in its historical form, is **hindered by its complexity**
3. We live at a **golden age to solve this problem**. A call to the community: we can make software more trustworthy!

THE UNIVERSITY
OF BRITISH COLUMBIA

Key Takeaways

1. Compartmentalization is a key practice to **improve the trustworthiness of software**
2. Compartmentalization, in its historical form, is **hindered by its complexity**
3. We live at a **golden age to solve this problem**. A call to the community: we can make software more trustworthy!

THE UNIVERSITY OF BRITISH COLUMBIA

Sponsors:

Co-funded by the European Union

UK Research and Innovation

Collaborators:

Key Takeaways

1. Compartmentalization is a key practice to **improve the trustworthiness of software**
2. Compartmentalization, in its historical form, is **hindered by its complexity**
3. We live at a **golden age to solve this problem**. A call to the community: we can make software more trustworthy!

Questions?

Reach out: hugo.lefeuvre@ubc.ca