

# Formally Secure Compilation of Unsafe C Compartments

Cătălin Hrițcu, MPI-SP, Bochum

Joint work with

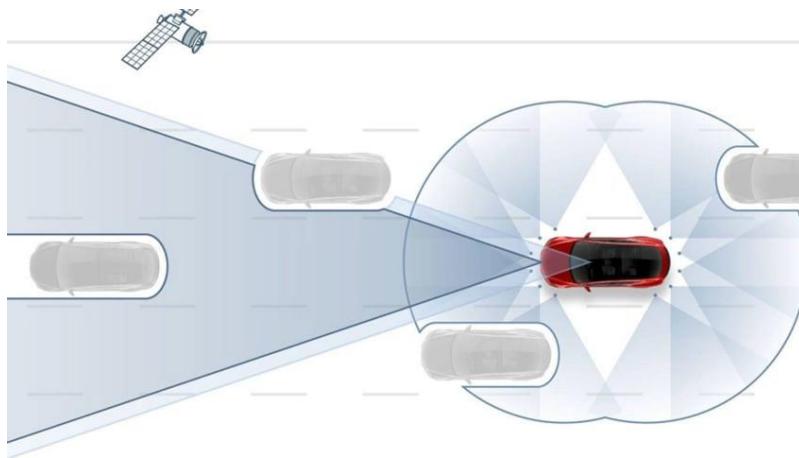
Carmine Abate, Cezar-Constantin Andrici, Arthur Azevedo de Amorim,  
Roberto Blanco, Ştefan Ciobâcă, Adrien Durier, **Akram El-Korashy**, Boris Eng,  
Ana Nora Evans, Guglielmo Fachini, **Deepak Garg**, Aïna Linn Georges, Théo Laurent,  
Guido Martínez, **Marco Patrignani**, Benjamin Pierce, Exequiel Rivas, Marco Stronati,  
Éric Tanter, Jérémie Thibault, Andrew Tolmach, Théo Winterhalter, ...

In part supported by ERC Starting Grant SECOMP

# We are increasingly reliant on computers



... trusting them with our ~~digital~~ lives



# Computers vulnerable to hacking

## Windows 10 zero-day exploit code released online

Security researcher 'SandboxEscaper' returns with new Windows LPE zero-day.



By Catalin Cimpanu for Zero Day | May 22,

## Heartbleed vulnerability may have been exploited months before patch [Updated]

Fewer servers now vulnerable, but the potential damage rises.

GOOGLE TECH ANDROID

## Google finds Android zero day that can take control of Pixel and Galaxy devices

Affecting devices from Samsung, Huawei, and Google itself

By Jon Porter | @JonPorty | Oct 4, 2019, 8:42am EDT

f t SHARE

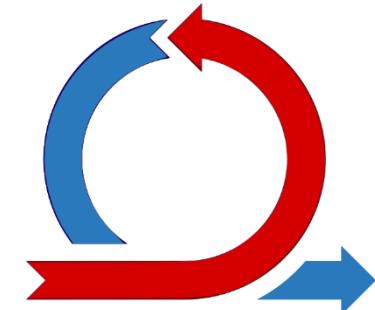
A visualization of binary code where the digits 0 and 1 are represented by green and red squares respectively, forming a grid pattern.

## Hackers Remotely Kill a Jeep on the Highway—With Me in It



# Need to break the exploitation cycle

- Once the stakes are high enough, **attackers will find a way to exploit *any* vulnerability**
- Weak security defenses** get deployed,



- We need a deeper understanding that we can use to build provably secure defenses**

- defenders find clever ways to "increase attacker effort"
  - **attackers find clever ways around them**

# Web browsers are frequently hacked

The screenshot shows a web browser window displaying a SPIEGEL ONLINE article. The page title is "Browser gets its input from the internet: a webpage (spiegel.de)". Below the title, a large heading reads "300+ resources loaded: html, image files, javascript, styles, ...". A red box highlights the URL "ad.doubleclick.net" in the list of loaded resources. The page content discusses the cost of photovoltaic systems. To the right, a sidebar for "CALL OF DUTY®: MODERN WARFARE® - OPERATOR ENHANCED EDITION" is shown, featuring a "JETZT VORBESTELLEN" button.

**Browser gets its input from the internet: a webpage (spiegel.de)**

**300+ resources loaded: html, image files, javascript, styles, ...**

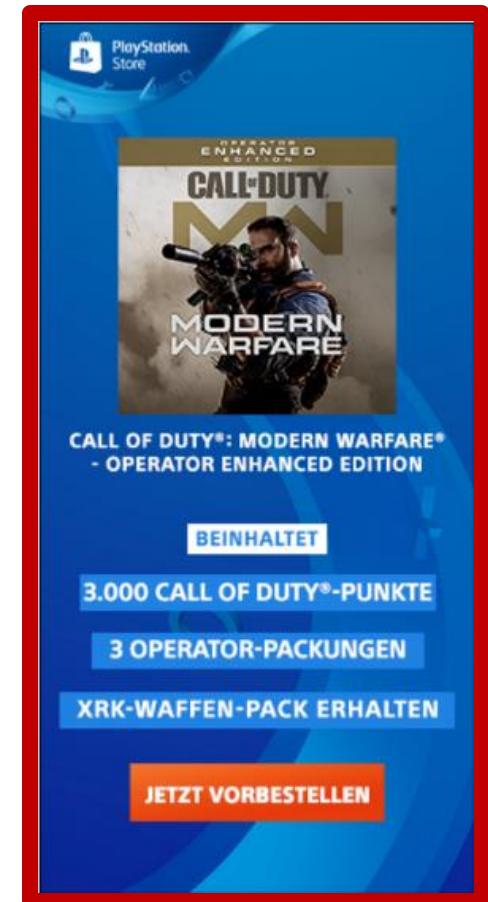
**from 25+ different internet servers**

**4 are clearly for ads:**

- ad.doubleclick.net
- ad.yieldlab.net
- amazon-adsystem.com
- adalliance.io

# Malicious server can hack the browser

- send it an image that **looks like an** ad
- **specially crafted to exploit a vulnerability** in the browser's image drawing engine
- **this compromises the whole browser**
  - i.e. gives server **complete control** over it
- **malicious server can now:**
  - steal the user's data
  - take control of the victim's computer
  - encrypt victim's data and ask for ransom



# Compromised browser can steal user's data

S SPIEGEL ONLINE - Aktuell x a Amazon Anmelden x +

amazon.de

## Anmelden

catalin.hritcu@gmail.com Ändern

Passwort Passwort vergessen

.....

Anmelden

Angemeldet bleiben. Details ▾

I've just given my password to the compromised browser controlled by ad.doubleclick.net

Unsere AGB

Datenschutzerklärung

Hilfe

Impressum

Hinweise zu Cookies

Hinweise zu interessenbasierter Werbung

© 1998-2019, Amazon.com, Inc. oder Tochtergesellschaften

# Compartmentalization can help

ONLINE - Aktuell x +

SPIEGEL

Brama Sport Kultur Netzwerk Wissenschaft mehr ▾

Schlagzeilen | DAX 12.633,60 | Abo

Bis zu 150 € sparen Will ich haben vodafone

Augsburg (M) FC Bayern (M) Bremen (M) 15.30 Hertha (M) 15.30 RB Leipzig (M) Wolfsburg (M)

+++ Brexit-Debatte im Livestream +++ Live  
"Johnson's Vorgehen gefährdet die Nation"

Das britische Unterhaus debattiert über Boris Johnsons Brexit-Deal. Dessen Parteikollege Oliver Letwin hält das Abkommen unverantwortlich und will mit einem Antrag die Abstimmung aufschieben. Die Live-News. Mit Max Hölzer mehr... [ Video | Forum ]

Die Lage am Samstag: Der Tag der Brexit-Entscheidung

+++ Livestream +++ Live  
Verfolgen Sie hier die Debatte im Unterhaus

Seit dem Morgen debattieren die britischen Parlamentarier, dabei wird es mitunter laut und emotional. Sehen Sie hier den Livestream aus dem Unterhaus. mehr...

SPIEGEL Hier finden Sie alle Artikel

[Facebook](#) [Twitter](#) [Instagram](#) [Newsletter](#)

# compromised compartment 1

Republikaner-Chef verurteilt Trumps Kurs in Syrien scharf

Amazon Anmelden x +

amazon.de

Anmelden

catalin.hritcu@gmail.com

amazon.de password is still secure!

Passwort

JETZT VORBESTELLEN

Anmelden

Angemeldet bleiben. Details ▾

# not compromised compartment 2

## Good news: browsers now compartmentalized!

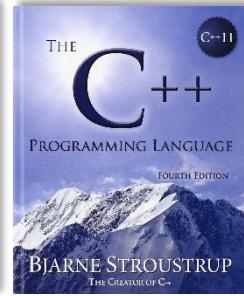
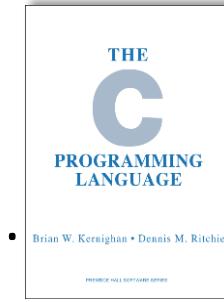
- each tab indeed started in separate compartment

## Bad news, so far:

- limited compartmentalization mechanisms
  - compartments coarse-grained, most often OS processes
    - can compartmentalize tabs, but not origins or resources within a tab
  - compartments can't naturally interact
    - even for tabs this required big restructuring of web browsers

# Source language compartments

- Mozilla Firefox mostly implemented in C/C++
- Programming languages like C/C++, Rust, Java, ... already provide **natural abstractions** for **fine-grained compartmentalization**:
  - procedures, interfaces, classes, objects, modules, libraries, ...
  - a **compartment** can be a library/module/class or even an object (e.g., an image or an origin)
- **In the source language fine-grained compartments are easy to define and can naturally interact**



# Source language compartments

(simple example in simplified source language)

**compartment C<sub>1</sub> {**

**private var x;**

**private procedure p() {**

**x := get\_counter();**

**x := password; ←not allowed**

**}**

**}**

**compartment C<sub>2</sub> {**

**private var counter;**

**private var password;**

**public procedure get\_counter() {**

**counter := counter + 1;**

**return counter;**

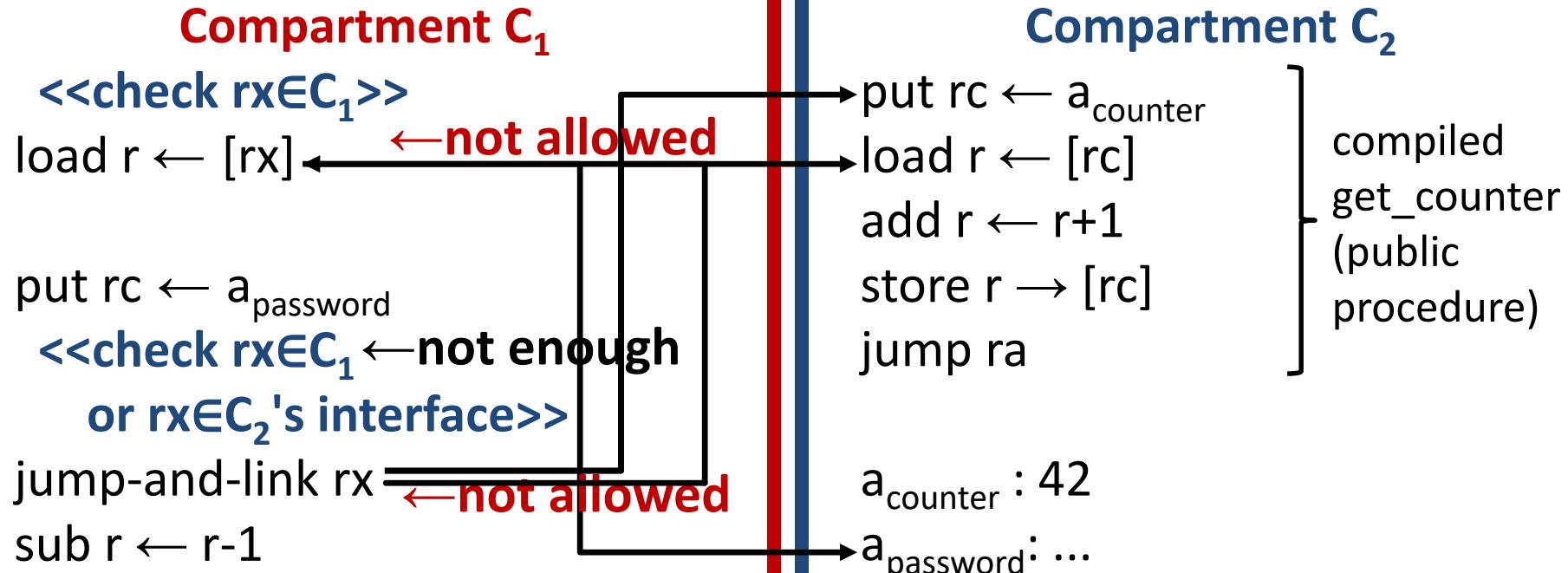
**}**

**}**

# Abstractions lost during compilation

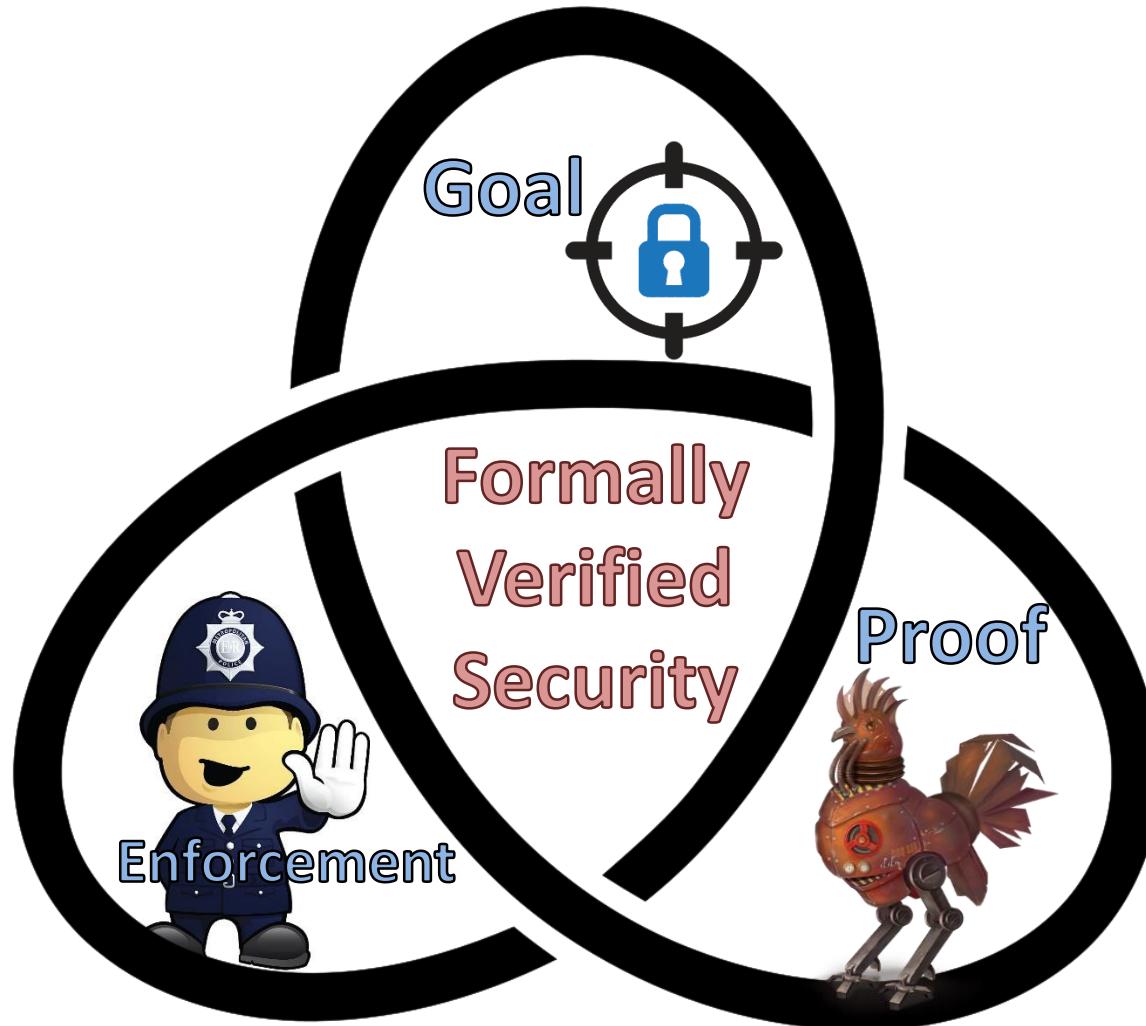
- Computers don't directly run C/C++, Java, Rust, or F\*
  - Compiler translates Firefox from C/C++/Rust to machine code instructions
- All high-level abstractions lost during compilation
  - no procedures, no interfaces, no classes, no objects, no modules, ...
- Secure compilation
  - preserve compartment abstractions through compilation, enforce them all the way down
- Shared responsibility of the whole compilation chain:
  - source language, compiler, operating system, and hardware
- Goal: secure compilation chain for compartmentalized code

# Machine-code level



**Securely enforcing source abstractions is challenging!**  
e.g. software checks complicated (uncircumventable, efficient)

# Formally Secure Compilation of C Compartments





# 1. Security Goal



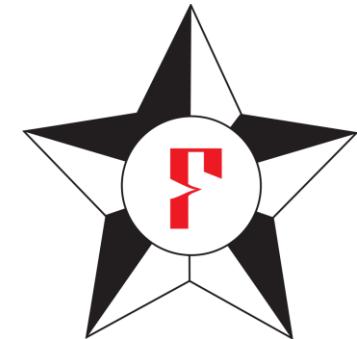
- What does it mean for a compilation chain for unsafe C compartments to be secure?
  - formal definition expressing end-to-end security guarantees
  - these guarantees were not understood before
- Will only show an easier definition
  - protecting 1 trusted compartment from 1 untrusted one
  - untrusted compartment arbitrary (e.g. compromised Firefox)
  - trusted compartment has no vulnerabilities

# This is not just hypothetical!



**Firefox**

**Mozilla shipping EverCrypt  
verified crypto library**  
**(also used by Microsoft, Linux, ...)**



[POPL'16, '17, '18, '20,  
ICFP'17, '19, ESOP'19,  
CPP'18, SNAPL'17]

**Formal verification milestone:**  
**40.000+ lines of highly-efficient code,**  
**mathematically proved to be free of vulnerabilities**  
**(and functionally correct and side-channel resistant)**

# Putting things into perspective

**EverCrypt**  
(verified in F\*)



**40.000 lines**



**Firefox**

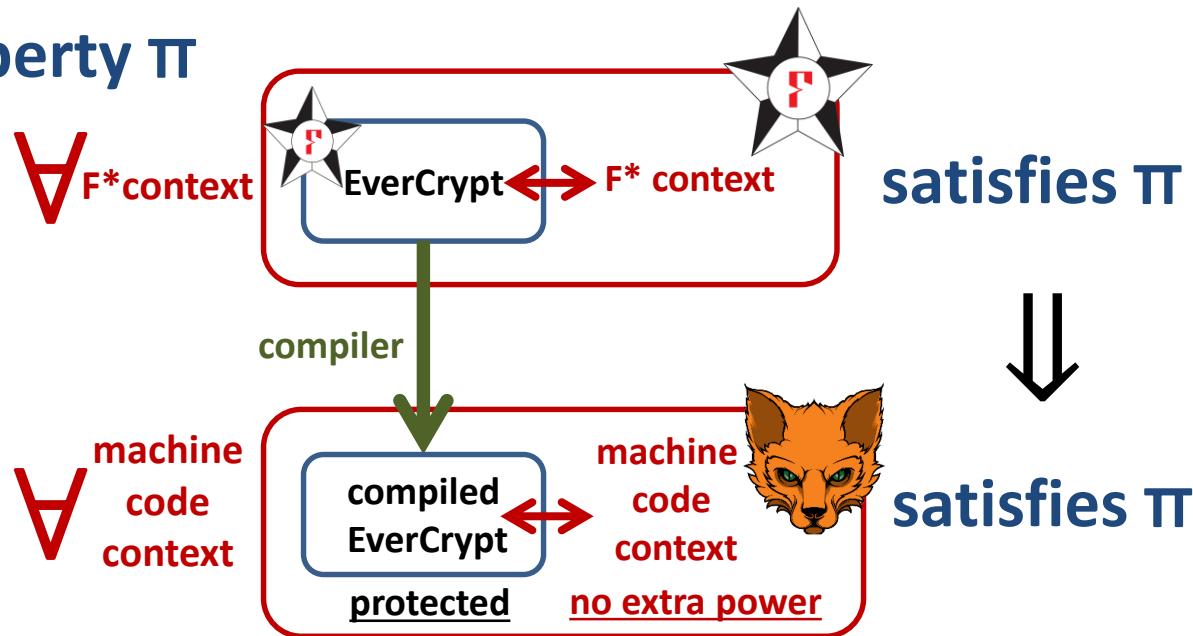
**20.000.000 lines**  
+ external libraries  
all unverified

**Without compartmentalization interoperability is insecure:  
if Firefox is compromised it can break security of verified code**

**What does secure compartmentalization mean in this setting?**

# Preserving security against adversarial contexts

$\forall$  security property  $\Pi$



Where "security property" can e.g., be **data confidentiality**

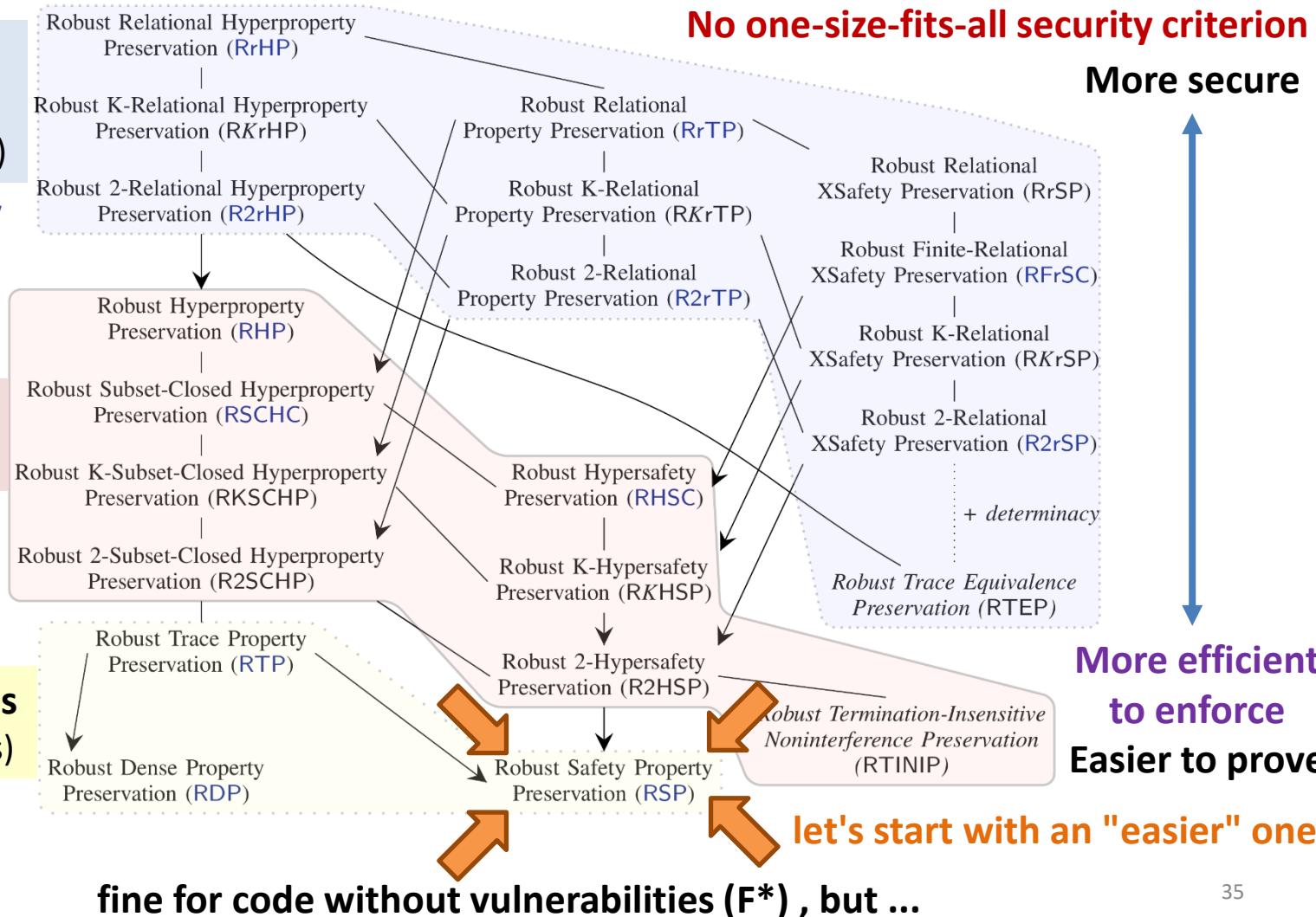
$\Pi$  = "private key is not leaked"

# Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

**relational  
hyperproperties**  
(trace equivalence)  
+ code confidentiality

**hyperproperties**  
(noninterference)  
+ data confidentiality

**trace properties**  
(safety & liveness)  
only integrity



# Extra challenges in defining secure compilation for unsafe C compartments [CSF'16, CCS'18]

- Program split into **many mutually distrustful compartments**
- **We don't know which compartments will be compromised**
  - every compartment should be protected from all the others
- **We don't know when a compartment will be compromised**
  - every compartment should receive protection until compromised

Compartment 1



Compartment 2



Compartment 3



Compartment 4



Compartment 5



# 2. Security Enforcement



CompCert C  
with compartments

large subset of the C language (ISO C 2011)

CompCert verified C compiler extended with compartments

CompCert RISC-V ASM  
with compartments

magically secure semantics for RISC-V ASM

Software-Fault Isolation

vanilla ASM

Done for simplified languages,  
yet to be ported to RISC-V

Micro-Policies: ASM  
with programmable tags

[POPL'14, S&P'15, ASPLOS'15,  
POST'18, CCS'18, CSF'23 subm.]

CHERI RISC-V  
capability machine

(inspiration for ARM Morello)

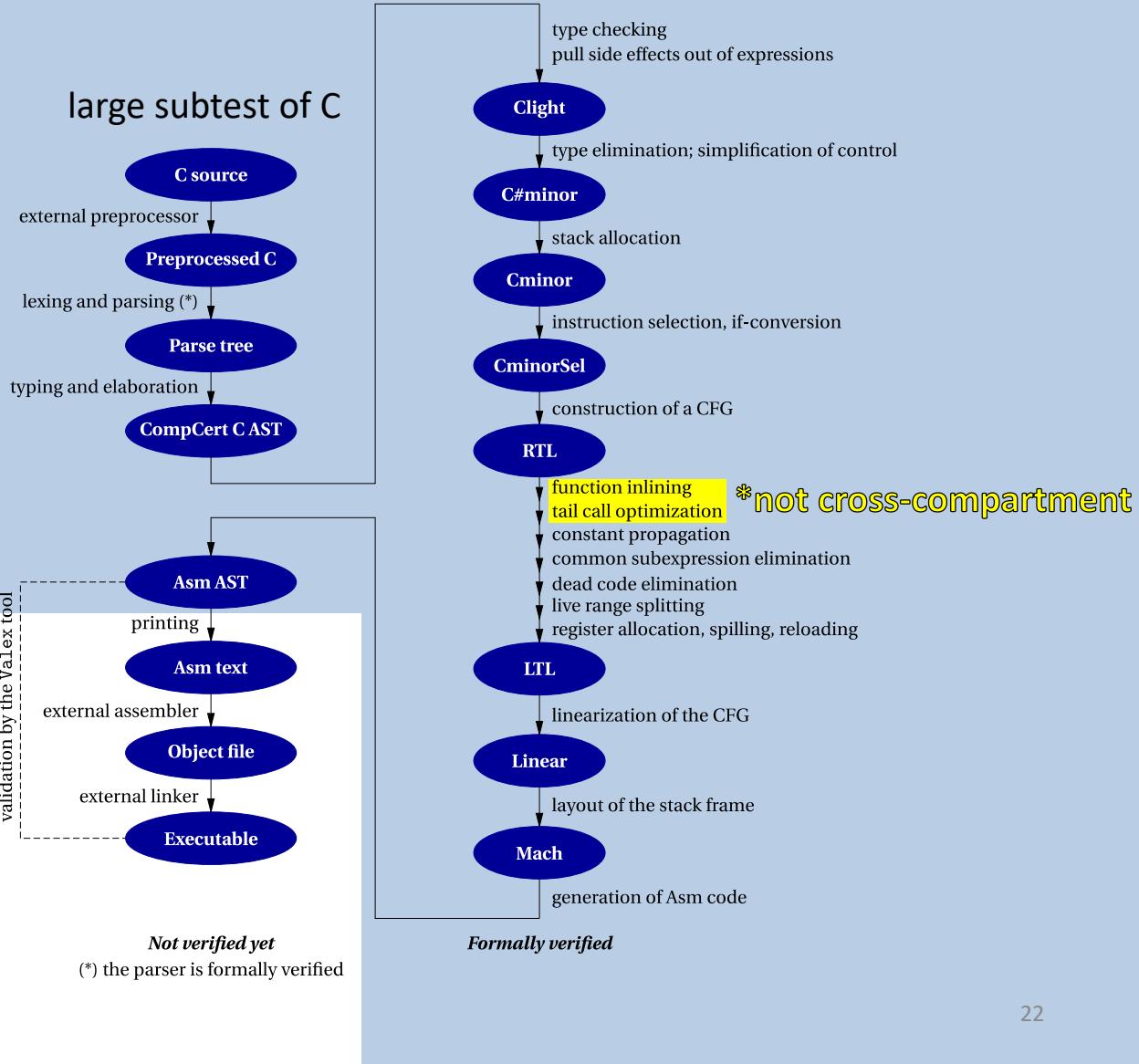
Hardware-accelerated enforcement

# CompCert extended with compartments

mutually distrustful,  
with clearly specified interfaces,  
interacting via procedure calls

all 15 verified compilation passes\*  
from Clight to RISC-V ASM  
(magically secure semantics)

compiler correctness proofs  
a lot of work, reusing for security



# Capabilities Backend



- Targeting the Cheri RISC-V capability machine
  - capabilities = unforgeable pointers with base and bounds
- Secure and efficient calling convention enforcing stack safety  
[Aïna Linn Georges et al, Le temps de cerises, OOPSLA 2022]
  - Uninitialized capabilities: cannot read memory before initializing
  - Directed capabilities: cannot access old stack frames
- Mutual distrustful compartments: capability-protected wrappers
  - on calls and returns clear registers and prevent passing capabilities between compartments
- Also investigating calling convention based solely on wrappers
  - no new kind of capability over what Cheri already provides
  - but more interesting stack layout (not a single contiguous block)

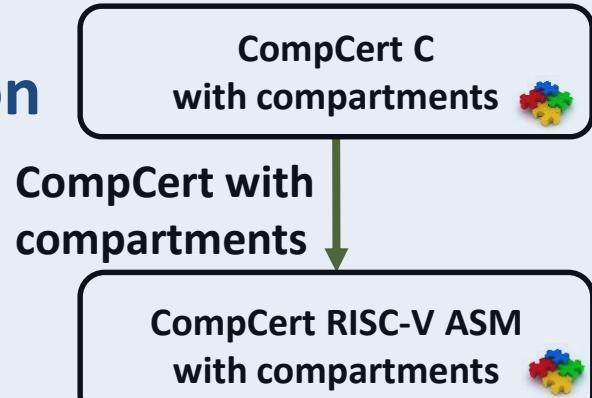
# 3. Security Proof



- **Proving mathematically that our compilation chain for C compartments achieves secure compilation**
  - such proofs generally **very difficult and tedious**
    - wrong conjectures survived for decades
    - 250 pages of proof on paper for toy compiler
  - we propose a **more scalable proof technique**
  - focus on **machine-checked proofs** in the Coq proof assistant
    - with **property-based testing** stopgap to find bugs early

# Testing and Proving secure compilation in Coq

## Verification



Scalable proof technique for secure compilation

- applied to simpler languages [CCS'18, CSF'22]
- currently porting to CompCert with compartments
- reuses our extended compiler correctness proof
- aiming to finish this in the next couple of months
  - milestone for secure compilation in terms of scale/realism

## Software-Fault Isolation



Done for simpler languages,  
yet to be ported to RISC-V

Next verification challenge

CHERI RISC-V capability machine

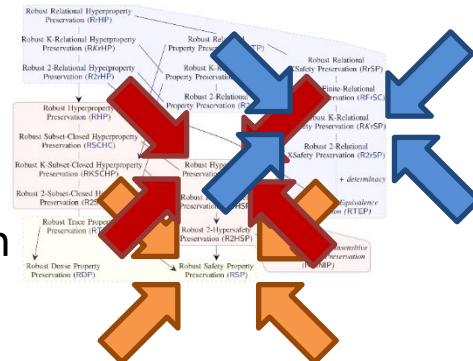


Systematic testing with QuickChick [POPL'17, ICFP'13, ITP'15, JFP'16]

# Future work: extending proof technique

- **Verifying backends more challenging**
  - can't hide all information about compartment's code (memory layout)
  - proof step inspired by full abstraction no longer works (recomposition)
- **Fine-grained dynamic memory sharing by capability passing**
  - already proved in Coq in simpler setting [Akram El-Korashy et al, CSF'22]
- **Beyond preserving safety against adversarial contexts**
  - towards preserving **hyperproperties** (data confidentiality)
  - even **relational hyperproperties** (observational equivalence)
    - secure compilation criteria strictly stronger than full abstraction
    - can do this for CompCert, but won't hold for backends

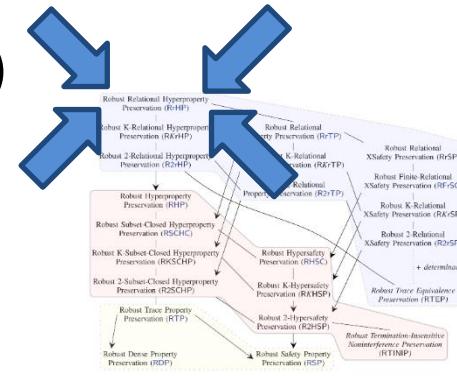
[Jérémie Thibault et al, CSF'19, ICFP'21 submission]



[CSF'19, ESOP'20,  
TOPLAS'21]

# Future work (continued)

- **Enforcement beyond preserving safety against adversarial contexts**
  - towards preserving **hyperproperties** (data confidentiality)
  - **challenging at the lowest level: [micro-architectural] side-channels attacks**
- **Dynamic component creation**
  - from code-based to data-based compartmentalization (e.g. browser tabs)
- **Dynamic privileges**
  - passing capabilities, dynamic interfaces, history-based access control, ...
- **Protecting higher-level abstractions (than those of the C lang.)**
  - **Securely Compiling Verified F\* Programs With IO**  
[Cezar-Constantin Andrici et al, ICFP'23 submission]
    - using reference monitoring and higher-order contracts
    - preserving **all relational hyperproperties** against adversarial contexts
    - first step towards formally secure F\*-OCaml interoperability



# Formally Secure Compilation of Unsafe C Compartments

## 1. Goal: formalize end-to-end security guarantees

- preserve properties **against adversarial contexts**
- we overcame additional challenges to support  
**mutually distrustful components** and **dynamic compromise**



## 2. Enforcement: protect abstractions all the way down

- **SFI** or **tagged architecture** or **capability machine**



## 3. Proof: verify security of our compilation chain

- **scalable proof technique** machine-checked in Coq
- applying it to **CompCert extended with compartments**



# Fine-grained compartmentalization

The screenshot shows the SPIEGEL ONLINE homepage. The top navigation bar includes the logo "SPIEGEL ONLINE", the URL "spiegel.de", a search icon, and a login button. Below the header, there's a menu with links to Politik, Meinung, Wirtschaft, Panorama, Sport, Kultur, Netzwerk, Wissenschaft, and more. The date "19. Oktober 2019" and a stock market tick "Schlagzeilen | DAX 12.633,60 | Abo" are also present. The main content area features several advertisements: one for "adalliance.io" showing a man with a yellow tool and a house with a red hexagon overlay; another for a "Gleitsichtbrille mit erweitertem Sehbereich"; and a third for "Was kostet Photovoltaik mit Stromspeicher?". Below these ads, a live video stream of a Brexit debate is shown, with a "Live" indicator. The video frame shows a man in a suit speaking.

SPIEGEL ONLINE - Aktuell +

spiegel.de

SPIEGEL ONLINE spiegel.de

Anmelden

Menü | Politik Meinung Wirtschaft Panorama Sport Kultur Netzwerk Wissenschaft mehr ▾

19. Oktober 2019

Schlagzeilen | DAX 12.633,60 | Abo

ANZEIGE

adalliance.io

Was kostet Photovoltaik mit Stromspeicher?  
Jetzt mit Stromspeicher dank Förderung & Energieverbrauch super rentabel!

Hauswert-Rechner: Wie viel ist Ihr Haus wert?  
Die Immobilienpreise sind auf Rekordhoch.  
Jetzt Preis ermitteln und zum Mega-Preis!

Gleitsichtbrille mit erweitertem Sehbereich  
Jetzt zum Sensationspreis von 109 € erhalten - bei über 550 Optikern

+++ Brexit-Debatte im Liveticker +++

Parlamentspräsident ~~Brexit~~ lässt Änderungsantrag zu - Brexit-Entscheidung könnte vertagt werden

spiegel.de Live

A man in a suit and tie speaking during a Brexit debate.

# Fine-grained compartmentalization

The screenshot shows a web browser window for 'Mein SPIEGEL - SPIEGEL' at [spiegel.de/meinspiegel/login.html](http://spiegel.de/meinspiegel/login.html). The page is divided into several compartments:

- Header Compartment:** Contains the 'SPIEGEL ONLINE' logo in red, the 'spiegel.de' logo in blue, a search icon, and a 'Anmelden' button.
- Navigation Bar:** Includes a menu icon, links for Politik, Meinung, Wirtschaft, Panorama, Sport, Kultur, Netzwelt, Wissenschaft, and 'mehr ▾', and links for Schlagzeilen, DAX 12.894,51, and Abo.
- Login Form Compartment:** Contains fields for 'Login' and 'spiegel.de', and a password field with placeholder text 'Benutzername oder E-Mail-Adresse' and value 'catalin.hritcu@gmail.com'. Below the form is a large text block: 'Spiegel.de password is still protected'.
- Facebook Integration Compartment:** A red-bordered box contains the 'facebook.com' logo over a gray noise background.
- My Services Compartment:** Contains a 'Meine Dienste' heading and links for 'Mein Börsendepot | Unternehmen', 'Meine Abos', and 'Newsletter verwalten'.
- Footer Compartment:** Contains links for 'Oder sind Sie neu hier?', 'Registrieren Sie sich jetzt kostenlos bei Mein', and 'Mein Konto'.

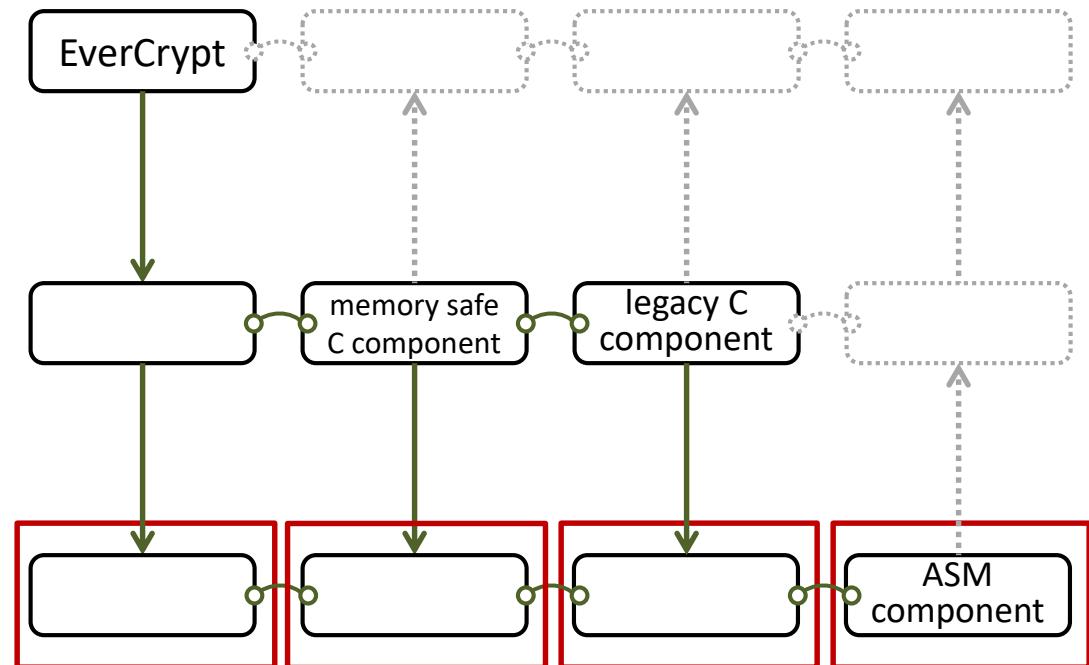
# My dream: secure compilation at scale



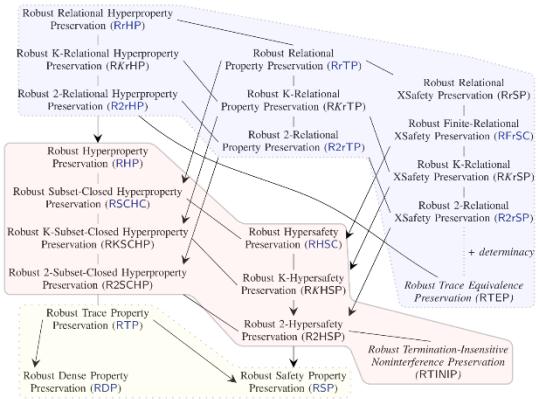
language

C language  
+ components  
+ memory safety

ASM language  
(RISC-V + micro-policies)



# Going beyond Robust Preservation of Safety



## Journey Beyond Full Abstraction (CSF 2019)



Carmine  
Abate  
Inria Paris



Rob  
Blanco  
Inria Paris



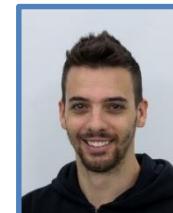
Deepak  
Garg  
MPI-SWS



Cătălin  
Hrițcu  
Inria Paris



Jérémie  
Thibault  
Inria Paris



Marco  
Patrignani  
Stanford  
& CISPA

# Going beyond Robust Preservation of Safety [CSF'19]

# relational hyperproperties (trace equivalence)

## + code confidentiality

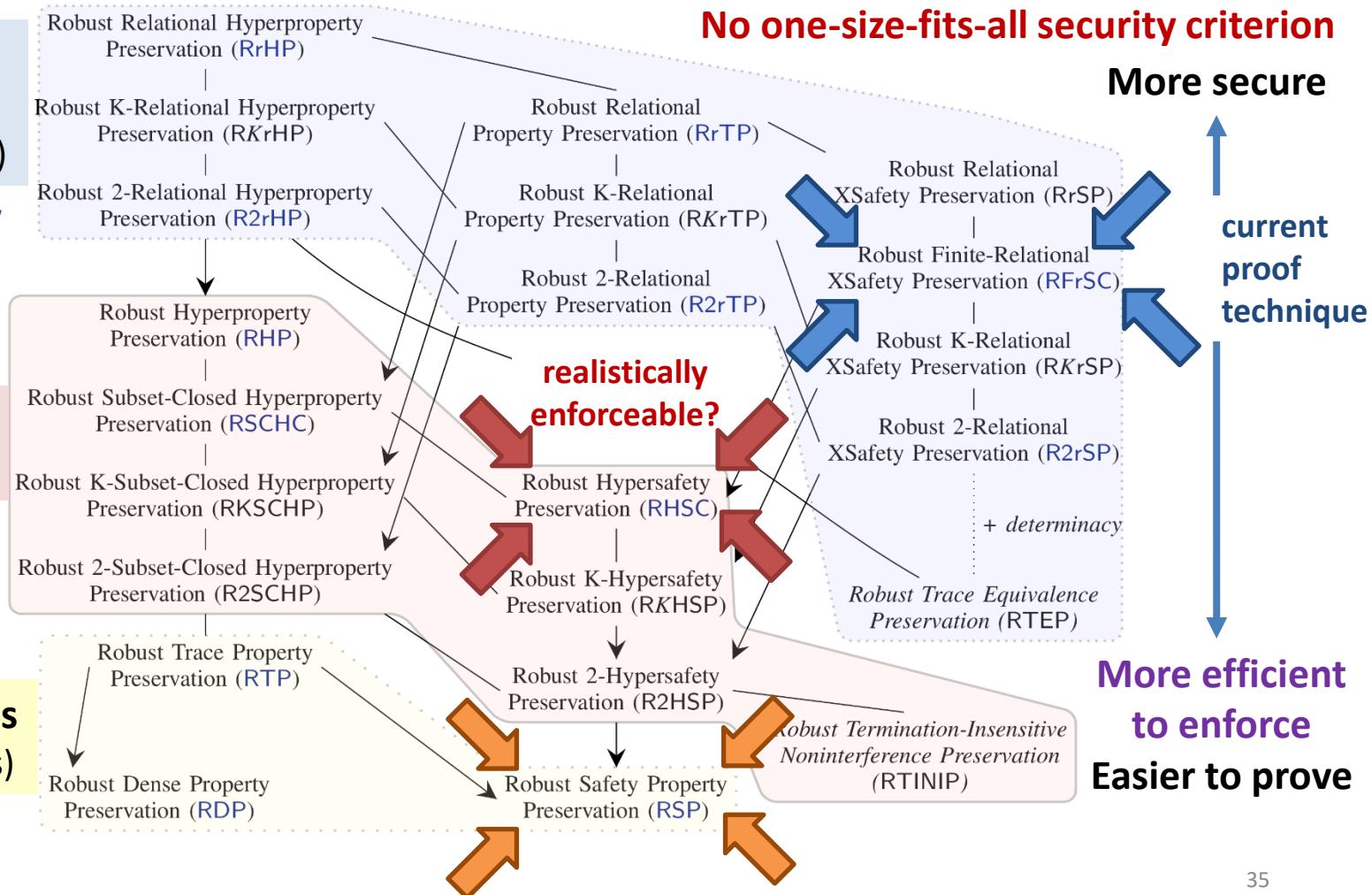
# hyperproperties

## (noninterference)

+ data confidentiality

## trace properties (safety & liveness)

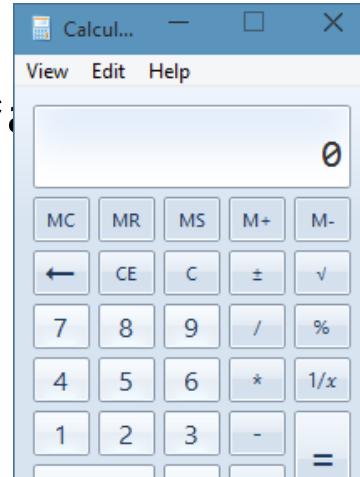
## only integrity



# Formalizing security of mitigations is hard

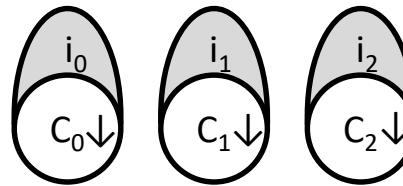
- We want source-level security reasoning principles
  - easier to reason about security in the source language if application is compartmentalized
- ... even in the presence of undefined behavior
  - can't be expressed at all by source language semantics!
  - what does the following program do?

```
#include <string.h>
int main (int argc, char **argv) {
    char c[12];
    strcpy(c, argv[1]);
    return 0;
}
```



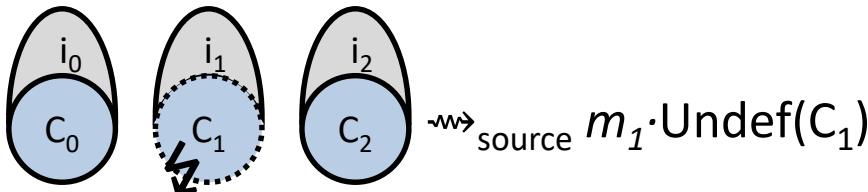
# Compartmentalizing compilation should ...

- **Restrict spatial scope** of undefined behavior
  - **mutually-distrustful components**
    - each component protected from all the others
- **Restrict temporal scope** of undefined behavior
  - **dynamic compromise**
    - each component gets guarantees as long as it has not encountered undefined behavior
    - i.e. the mere existence of vulnerabilities doesn't necessarily make a component compromised

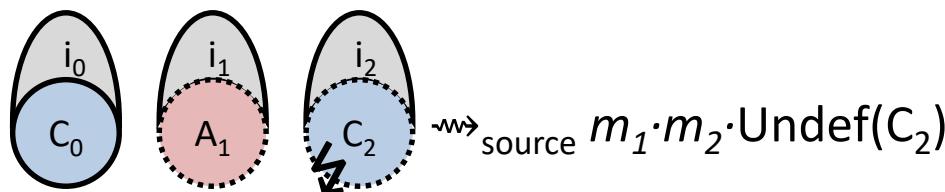
**Security definition:** If   $\rightsquigarrow$  machine  $m$  then

$\exists$  a sequence of component compromises explaining the finite trace  $m$  in the source language, for instance  $m=m_1 \cdot m_2 \cdot m_3$  and

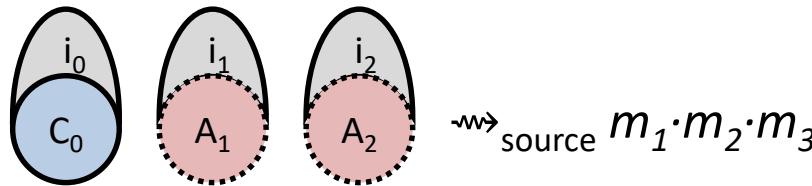
(1)



(2)  $\exists A_1 \cdot$



(3)  $\exists A_2 \cdot$



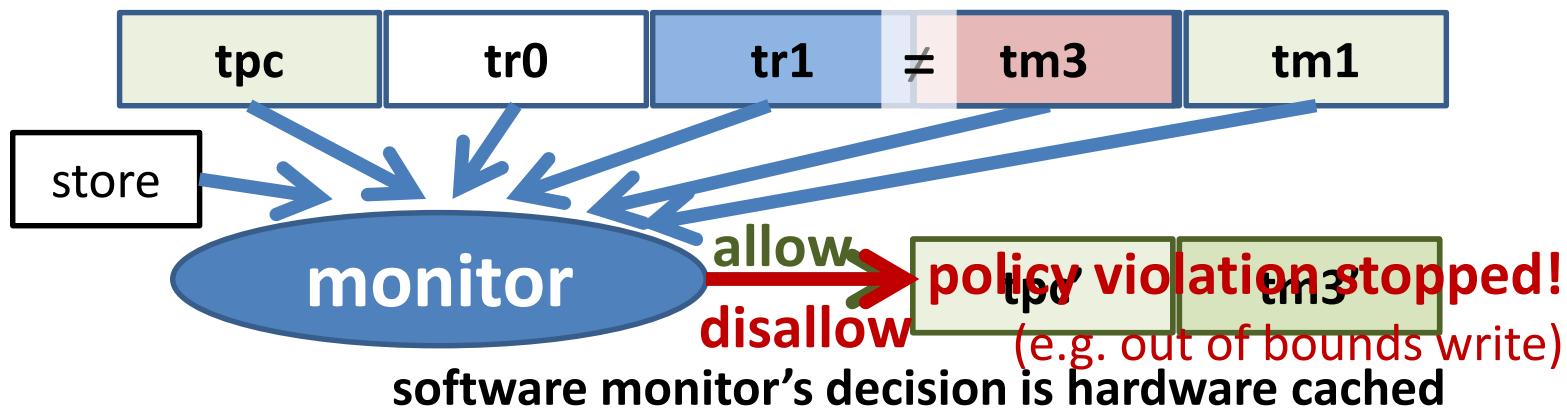
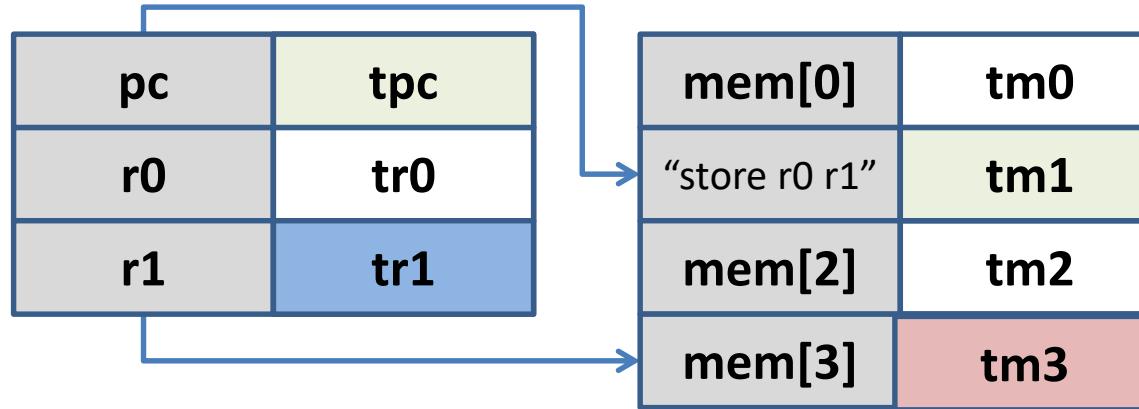
Finite trace  $m$  records which component encountered undefined behavior and allows us to rewind execution



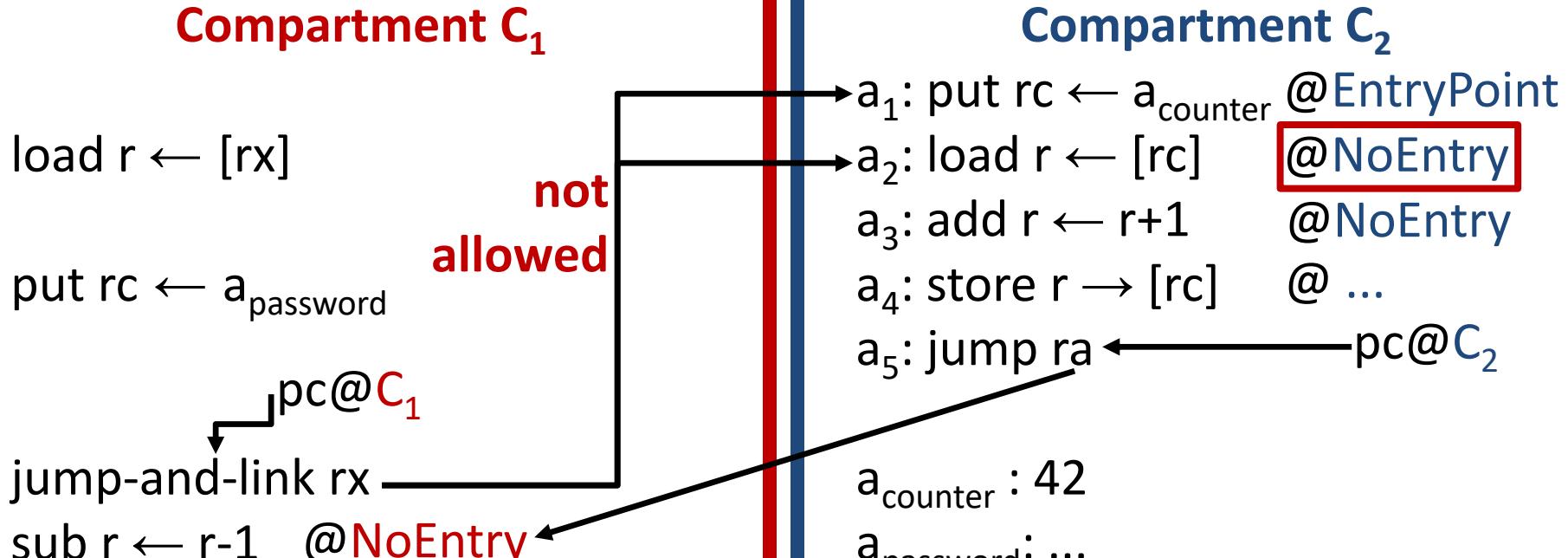
# Micro-Policies

[POPL'14, Oakland'15, ASPLOS'15, POST'18, CCS'18]

software-defined, hardware-accelerated, tag-based monitoring

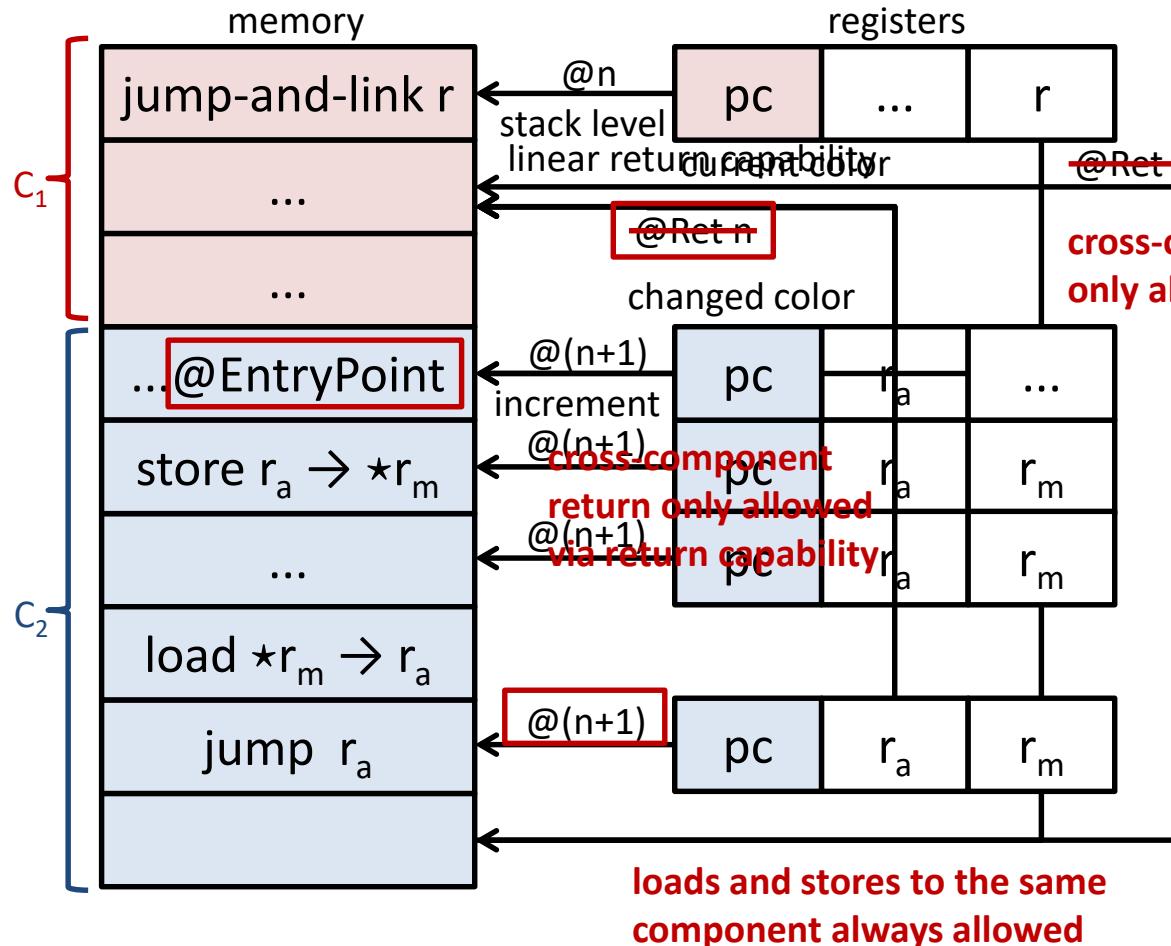


# Compartmentalization micro-policy



**Challenge:** making sure returns go to the right place

# Compartmentalization micro-policy (calls and returns)



invariant:  
at most one  
return capability  
per call stack level

cross-component call  
only allowed at EntryPoint

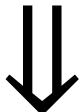
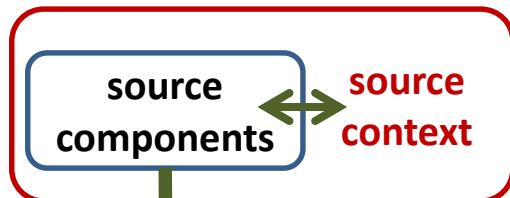
Enforcement quickly gets complicated

We reduce our proof goal to a variant of:

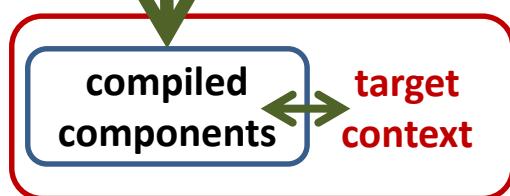
# Robust Safety Preservation

$\forall$  source components.  
 $\forall \pi$  safety property.

$\forall$  source context trace  $t$ .



compiler

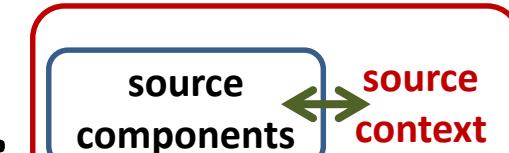


$\forall$  target context trace  $t$ .

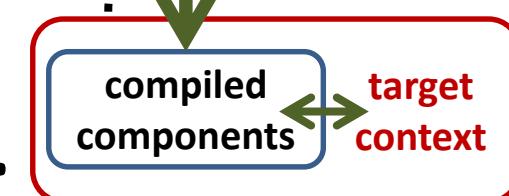
robust preservation of safety

$\forall$  source components.  
 $\forall$  (bad/attack) finite trace  $m$ .

$\exists$  source context.



compiler



$\exists$  target context.

proof-oriented characterization

$\leftrightarrow$

back-translation

# Scalable proof technique

(for our variant of Robust Safety Preservation)



1. back-translating finite trace prefix to whole source program
- 2+4. compiler correctness proof (à la CompCert) used as a black-box
- 3+5. also simulation proofs, but at a single level

