

Formally Secure Compartmentalizing Compilation

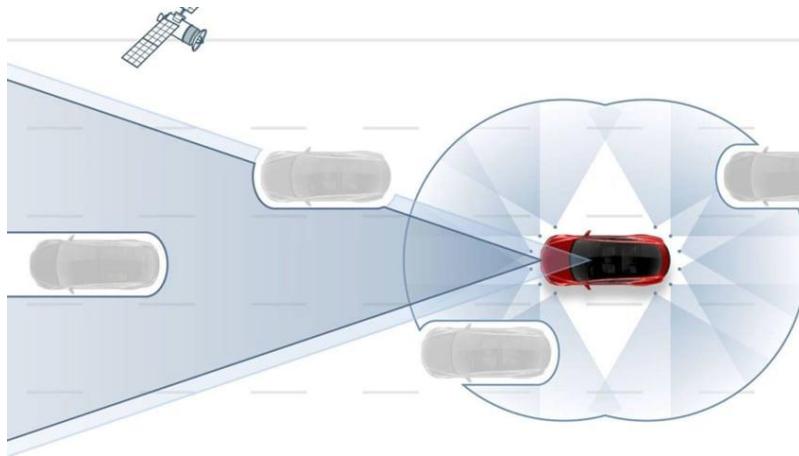
Cătălin Hritcu

Inria Paris

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

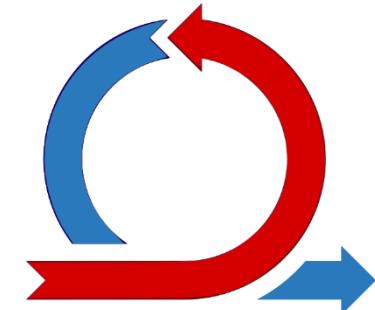
A visualization of binary code where the digits 0 and 1 are represented by green and red squares respectively, forming a grid pattern.

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



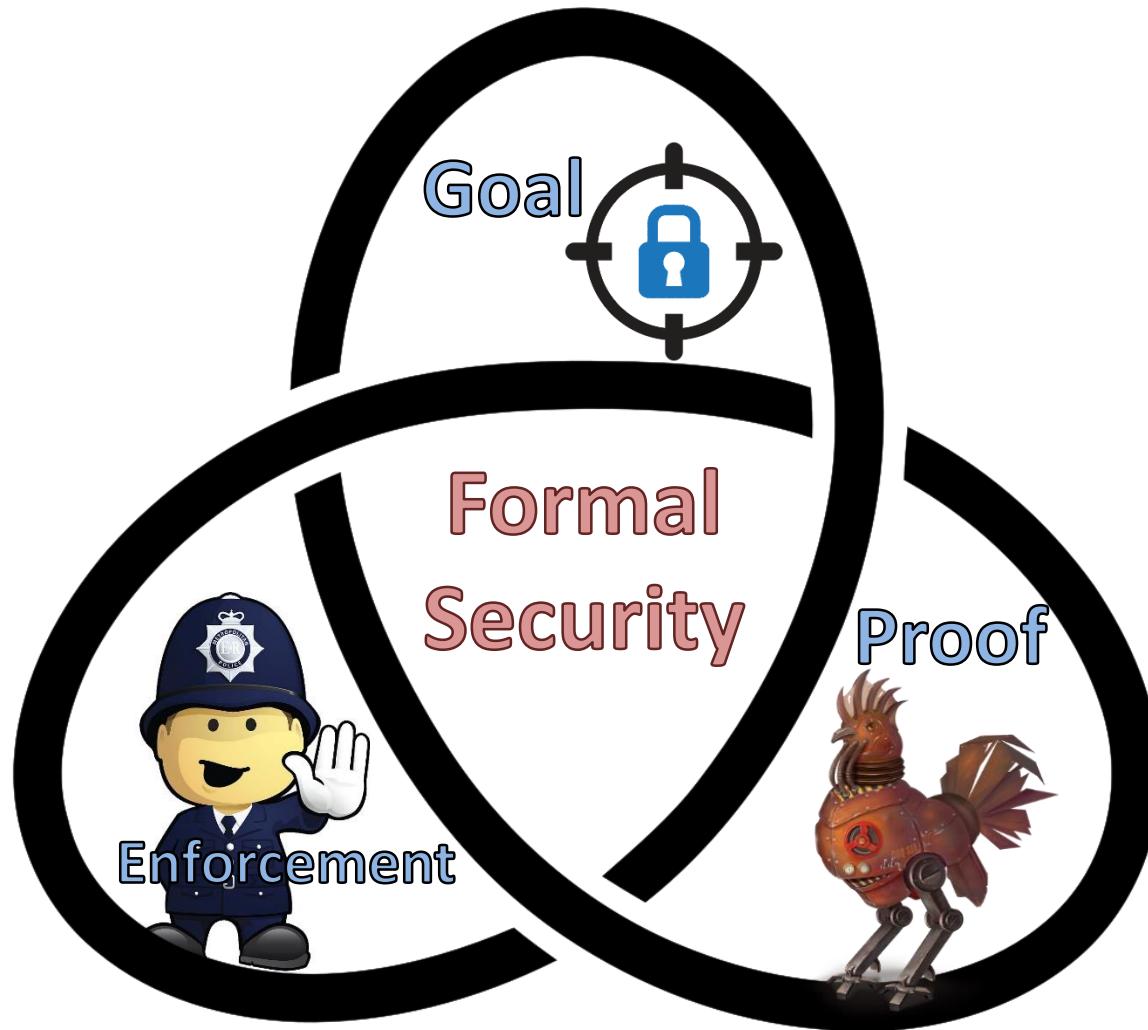
Need to break the exploitation cycle

- 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, there is an advertisement for "CALL OF DUTY®: MODERN WARFARE® - OPERATOR ENHANCED EDITION" with 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

ad.doubleclick.net

antrag zu -

Live

CALL OF DUTY®: MODERN WARFARE®
- OPERATOR ENHANCED EDITION

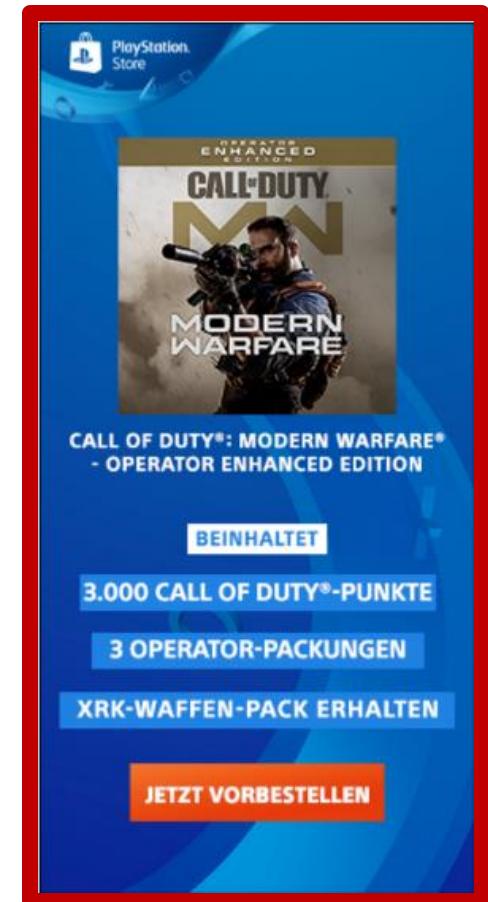
BEINHALTET

3.000 CALL OF DUTY®-PUNKTE
3 OPERATOR-PACKUNGEN
XRK-WAFFEN-PACK ERHALTEN

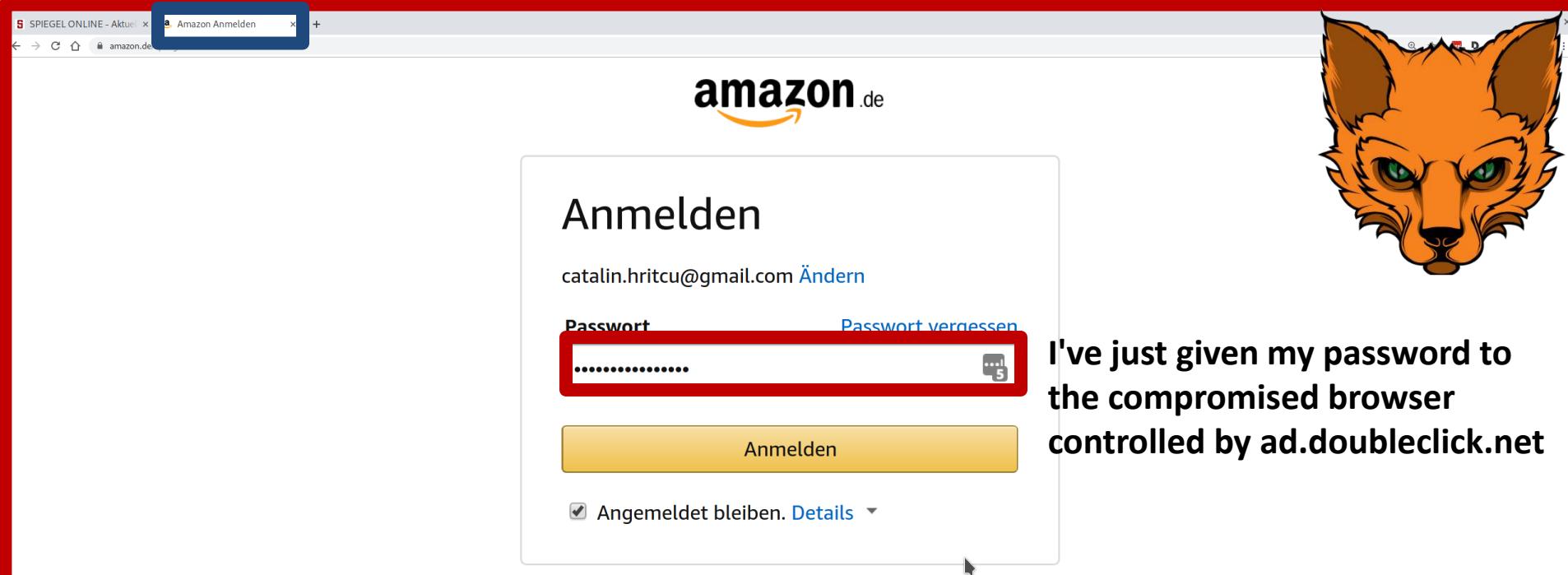
JETZT VORBESTELLEN

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



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 Livesticker +++ 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 mechanism
 - compartments coarse-grained
 - can compartmentalize tabs, but not secrets within a tab
 - compartments can't naturally interact
 - even for tabs this required big restructuring of web browsers

Fine-grained compartmentalization

The screenshot shows the SPIEGEL ONLINE homepage. The top navigation bar includes the logo, search bar, and login link. Below the header, there's a menu with categories like Politik, Meinung, Wirtschaft, Panorama, Sport, Kultur, Netzwerk, Wissenschaft, and more. The date 19. Oktober 2019 is displayed, along with a stock market tick (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 Bercow lässt Änderungsantrag zu - Brexit-Entscheidung könnte vertagt werden

Live

spiegel.de

A large image of a man in a suit speaking, identified as the Speaker of the House of Commons, John Bercow.

doubleclick.net

The right side of the page contains a red-bordered box with the text "doubleclick.net" and a large, grainy, black-and-white image, likely representing a placeholder or a broken image.

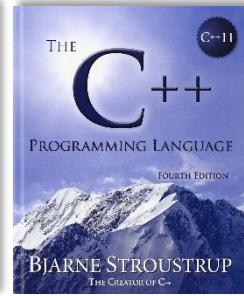
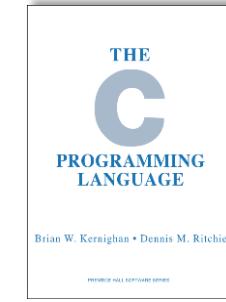
Fine-grained compartmentalization

The screenshot shows the SPIEGEL ONLINE login page (spiegel.de/meinspiegel/login.html). The page is divided into several compartments:

- Header Compartment:** Contains the SPIEGEL ONLINE logo, a search bar, and an "Anmelden" button.
- Navigation Compartment:** Includes a menu icon, navigation links like Politik, Meinung, Wirtschaft, Panorama, Sport, Kultur, Netzwelt, Wissenschaft, and a "mehr ▾" link.
- Main Content Compartment (Left):** A "MEIN SPIEGEL" section with a "Nachrichten > Mein SPIEGEL" breadcrumb. It contains a login form with fields for "Benutzername oder E-Mail-Adresse" (catalin.hritcu@gmail.com) and "Passwort" (redacted by a red box). Below the form is the text: "Spiegel.de password is still protected".
- Main Content Compartment (Center):** A large red box covers the central content area, obscuring the URL "facebook.com".
- Main Content Compartment (Right):** Contains a "Meine Dienste" section with links: "Mein Börsendepot | Unternehmen", "Meine Abos", and "Newsletter verwalten".
- Footer Compartment:** At the bottom left, there's a link "Oder sind Sie neu hier? Registrieren Sie sich jetzt kostenlos bei Mein". At the bottom right, there's a "Mein Konto" link.

Source language compartments

- Mozilla Firefox mostly implemented in C/C++
- Programming languages like C/C++, Java, F*, ... 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)
- **In the source language fine-grained compartments are easy to define and can naturally interact**



Source language compartments

compartment C₁ {

private var x;

private procedure p() {

x := get_counter();

x := password; ←not allowed

}

}

compartment C₂ {

private var counter;

private var password;

public procedure get_counter() {

counter := counter + 1;

return counter;

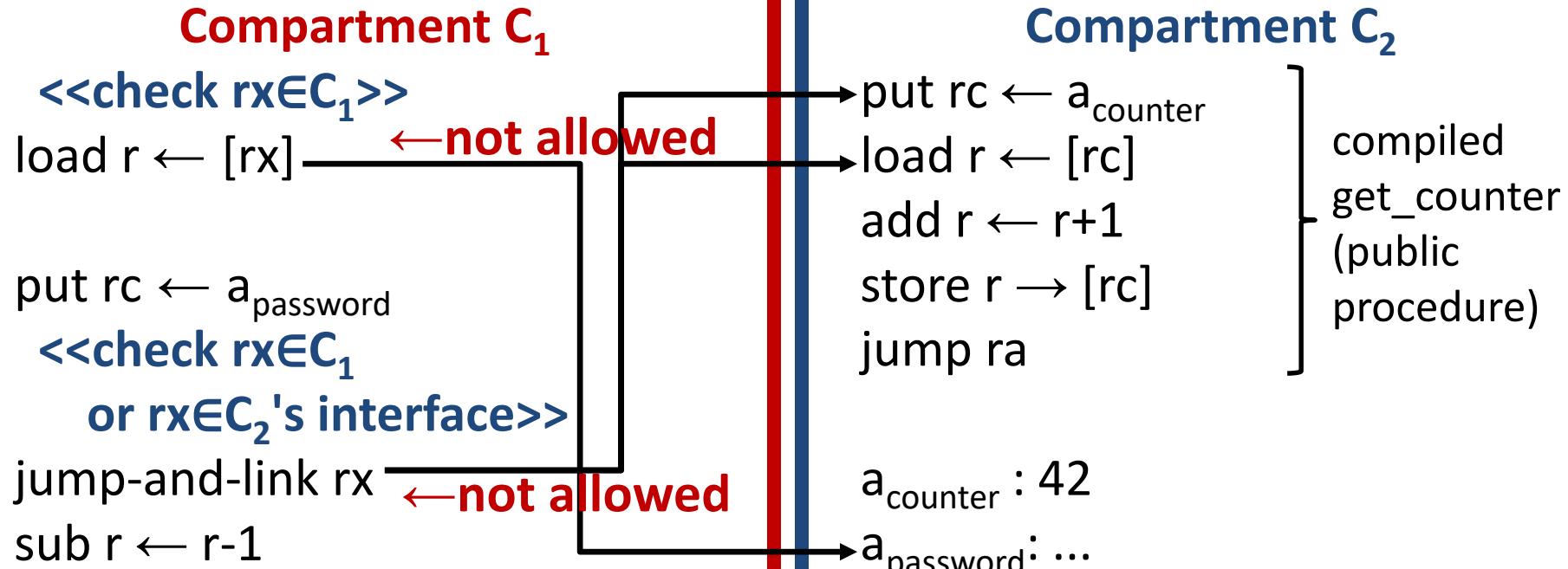
}

}

Abstractions lost during compilation

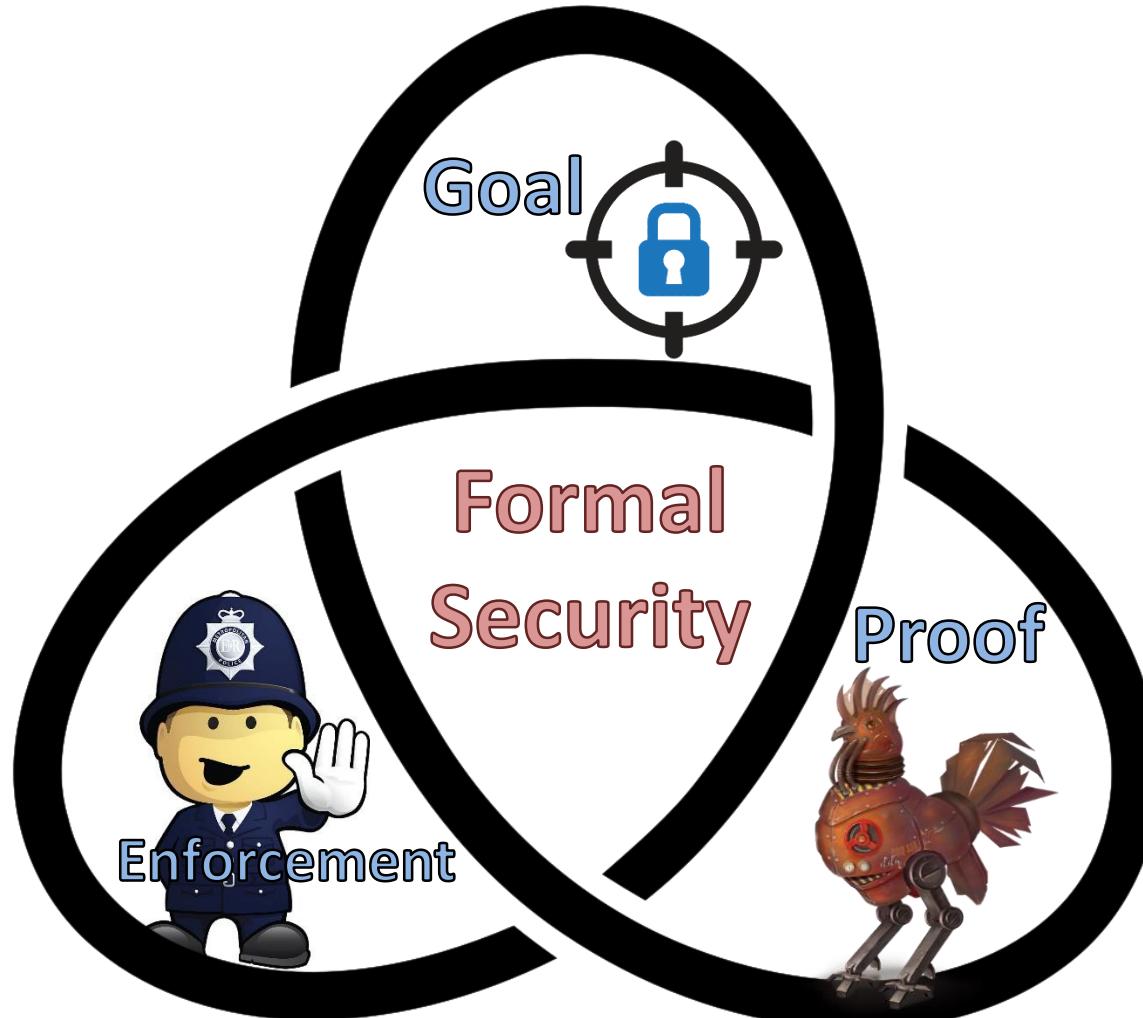
- Computers don't run C/C++, Java, or F*
 - Compiler translates Firefox from C/C++ to machine code instructions
- All compartmentalization abstractions lost during compilation
 - no procedures, no interfaces, no classes, no objects, no modules, ...
- Secure compilation
 - preserve abstractions through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

Machine-code level



Securely enforcing source abstractions is challenging!

Formally Secure Compartmentalizing Compilation





1. Security Goal



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

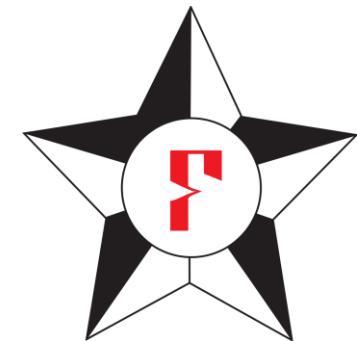
This is not just hypothetical!



Firefox

**Mozilla shipping EverCrypt
verified crypto library**

(also used by Microsoft, Linux, ...)

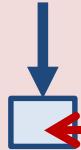


[POPL'16,'17,'18,'20,
ICFP'17,'19, ESOP'19,
CPP'18, SNAPL'17]

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

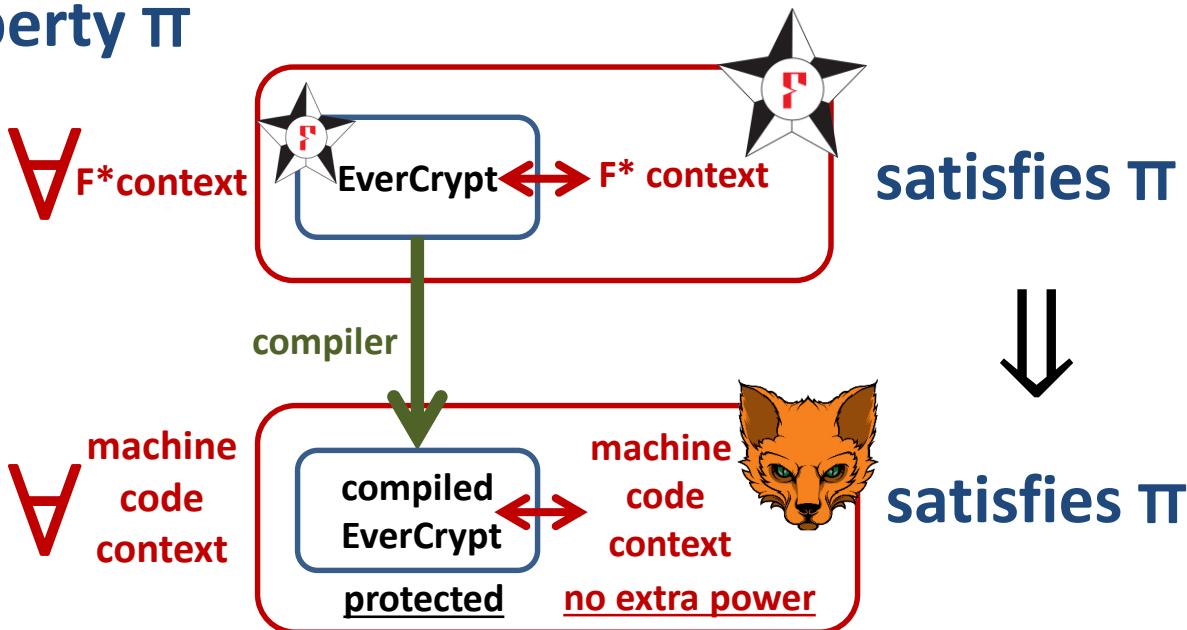
**20.000.000 lines
+ external libraries
all unverified**

**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 Π



Where "security property" can e.g., be
safety or integrity or **confidentiality** [CSF'19]

Π = "EverCrypt's private key is not leaked"

Extra challenges for our real security definition

[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



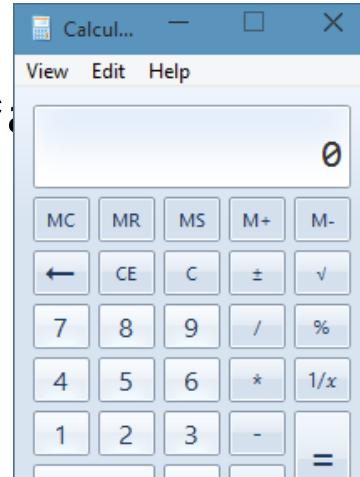
Compartment 5



Formalizing security of mitigations is hard

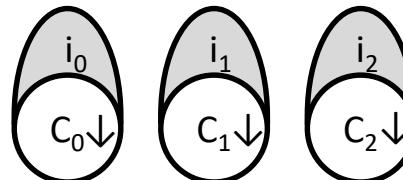
- We want source-level security reasoning principles
 - easier to reason about security in the source language if application is compartmentalized
- ... even in the presence of undefined behavior
 - can't be expressed at all by source language semantics!
 - what does the following program do?

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



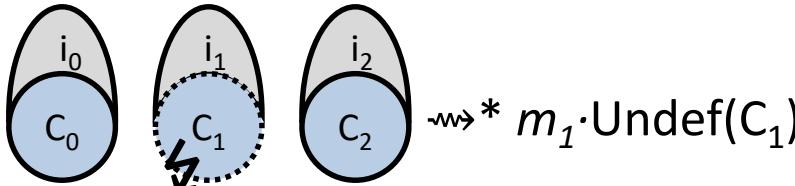
Compartmentalizing compilation should ...

- **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

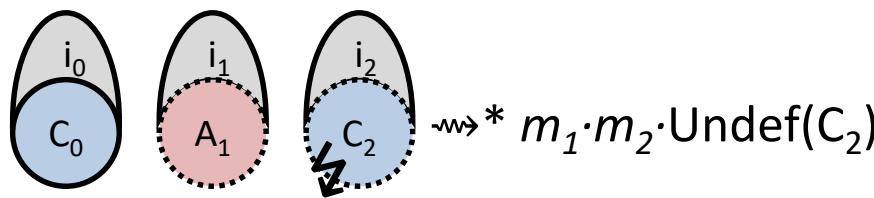
Security definition: If  $\rightsquigarrow m$ then

\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

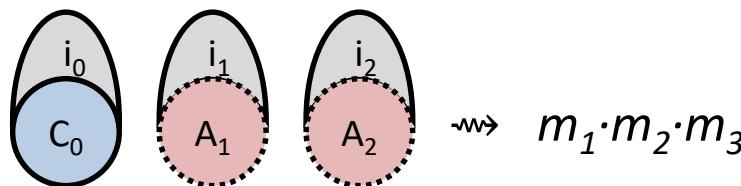
(1)



(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

2. Security Enforcement

Prototype compartmentalizing compilation chain

```
compartment C2 {  
    private var counter;  
    private var password;  
    public procedure get_counter() {  
        counter := counter + 1;  
        return counter;  
    }  
}
```



Buffers, procedures, compartments

Intermediate language with built-in compartmentalization

Bare-bone machine

Machine code

Hardware-accelerated enforcement

+Software enforcement

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

Software-fault isolation

Compartment C₁

<<check rx∈C₁>>

load r ← [rx] ←

put rc ← a_{password}

<<check rx∈C₁ ← not enough

or rx∈C₂'s interface>>

jump-and-link rx

sub r ← r-1

Compartment C₂

a₁: put rc ← a_{counter}

a₂: load r ← [rc]

a₃: add r ← r+1

a₄: store r → [rc]

a₅: jump ra

a_{counter} : 42

a_{password}: ...

Idea: rewrite C₁'s (& C₂'s) code to insert all the required checks

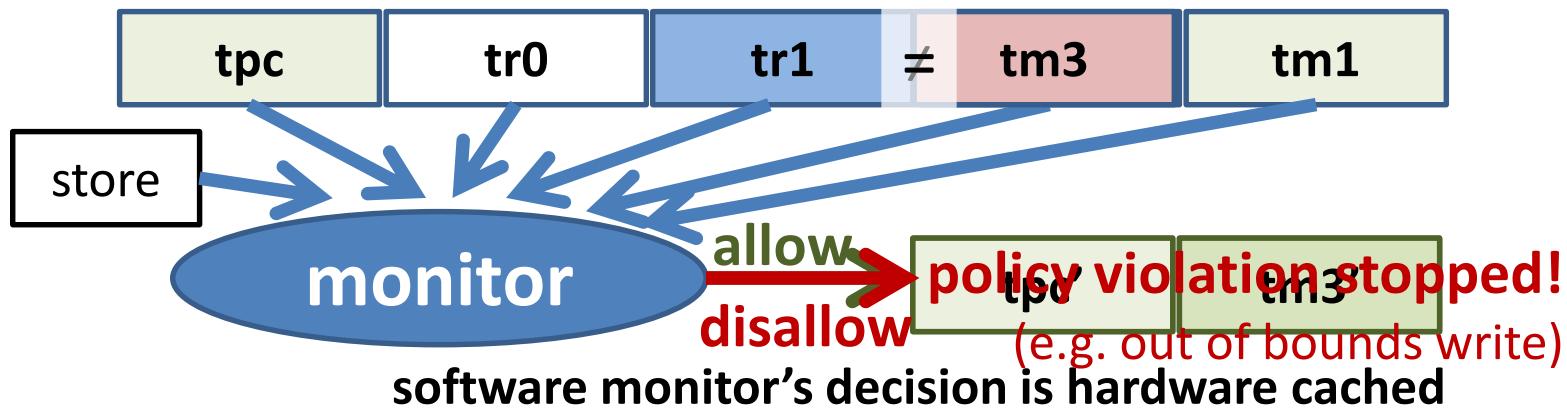
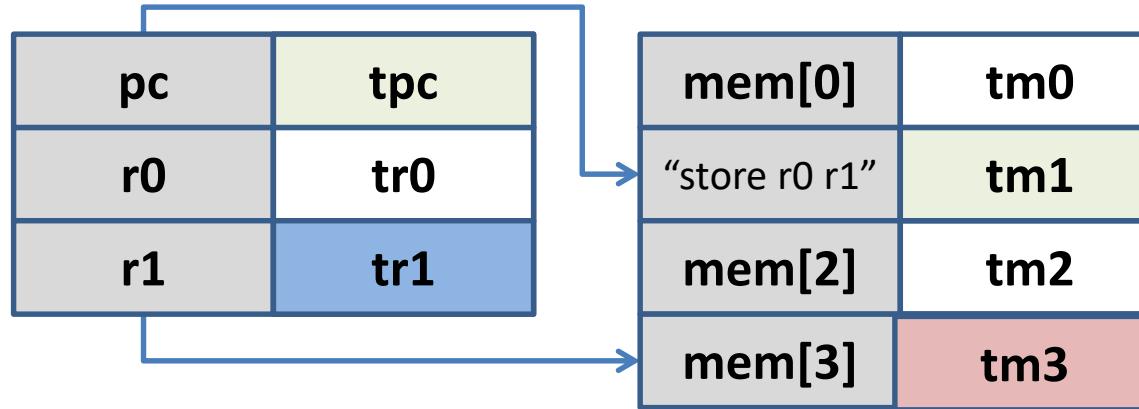
Challenges: checks complicated (uncircumventable, efficient)



Micro-Policies

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

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



Compartmentalization micro-policy



Compartment C_1

load r \leftarrow [rx]

put rc \leftarrow a_{password}

not
allowed

pc@ C_1
jump-and-link rx

sub r \leftarrow r-1 @NoEntry

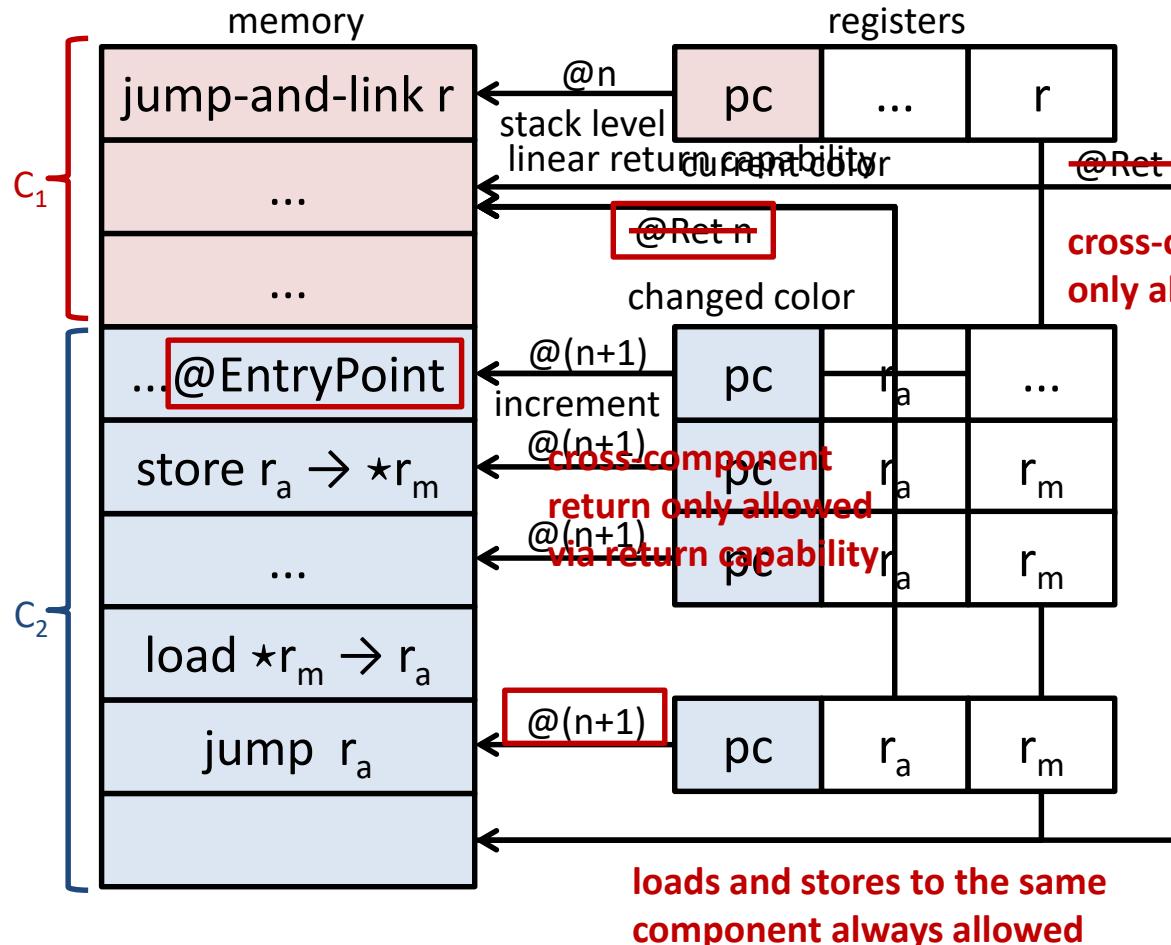
Compartment C_2

a₁: put rc \leftarrow a_{counter} @EntryPoint
 a₂: load r \leftarrow [rc] @NoEntry
 a₃: add r \leftarrow r+1 @NoEntry
 a₄: store r \rightarrow [rc] @ ...
 a₅: jump ra \leftarrow pc@ C_2

a_{counter}: 42
 a_{password}: ...

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

3. Security Proof



- **Proving mathematically that a compartmentalizing compilation chain achieves the security goal**
 - formally verifying the security of the whole compilation chain
 - such proofs **very difficult and tedious**
 - wrong conjectures survived for decades; 250pg for toy compiler
 - we propose a **more scalable proof technique**
 - focus on **machine-checked proofs** in the Coq proof assistant
 - **Proof-of-concept formally secure compilation chain in Coq**

Verified



Compartmentalized
unsafe source



Buffers, procedures, components
interacting via **strictly enforced interfaces**

generic proof technique

20K lines of Coq, mostly proofs

Compartmentalized
abstract machine



Simple RISC abstract machine with
build-in compartmentalization

Micro-policy
machine



Tag-based reference monitor enforcing:
- component separation
- procedure call and return discipline
(linear capabilities / linear entry points)

software fault isolation

Bare-bone
machine

Inline reference monitor enforcing:
- component separation
- procedure call and return discipline
(program rewriting, shadow call stack)

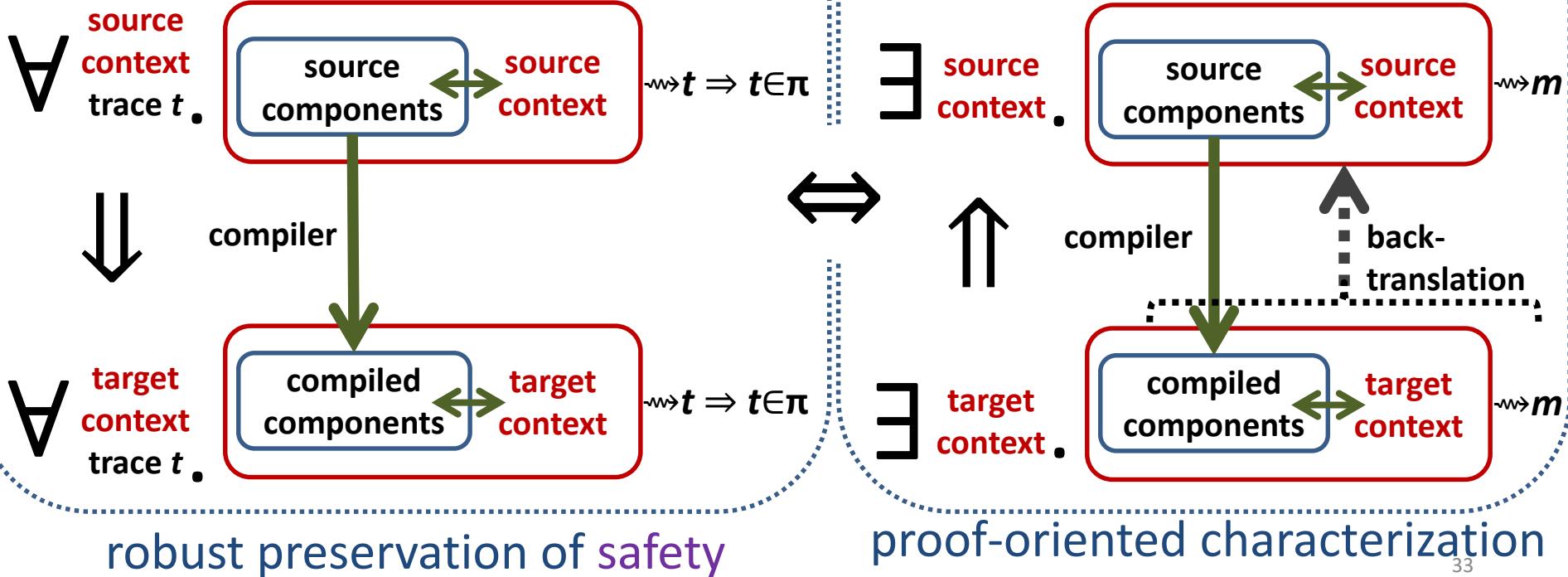
Systematically tested (with QuickChick)



We reduce our proof goal to a variant of:

Robust Safety Preservation

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

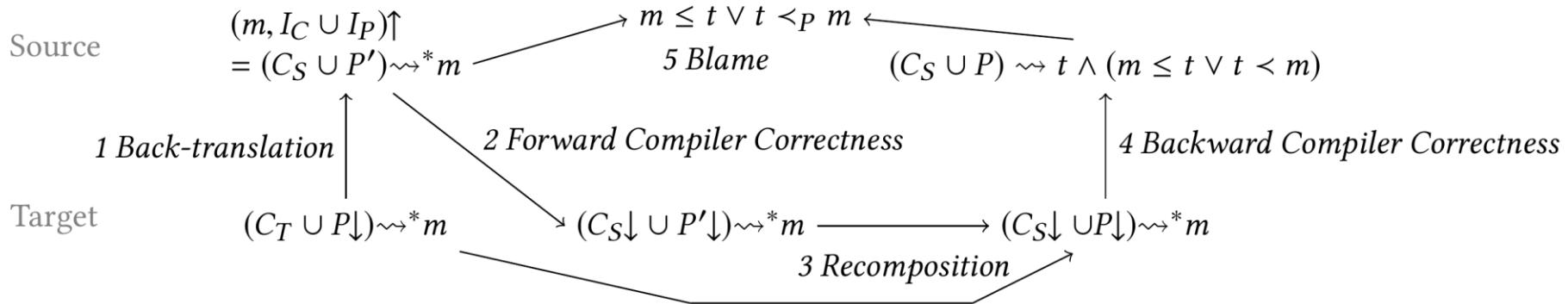


Scalable proof technique

(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



Summary

Compartmentalizing compilation is an important security defense in practice



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

- first definition supporting **mutually distrustful components** and **dynamic compromise**



2. Enforcement: protect abstractions all the way down

- **software fault isolation** or **tag-based architecture**



3. Proof: verify security of entire compilation chain

- **scalable proof technique machine-checked in Coq**



Making this more practical ... next steps:

- Scale formally secure compilation chain to **C language**
 - allow **pointer passing** (capabilities for fine-grained memory sharing)
 - eventually support enough of C to **measure and lower overhead**
 - check whether hardware support (tagged architecture) is faster
- Extend all this to **dynamic component creation**
 - rewind to when compromised component was created
- ... and **dynamic privileges**
 - capabilities, dynamic interfaces, history-based access control, ...
- From robust safety to **hypersafety (confidentiality)** [CSF'19]
- Secure compilation of EverCrypt, miTLS, ...

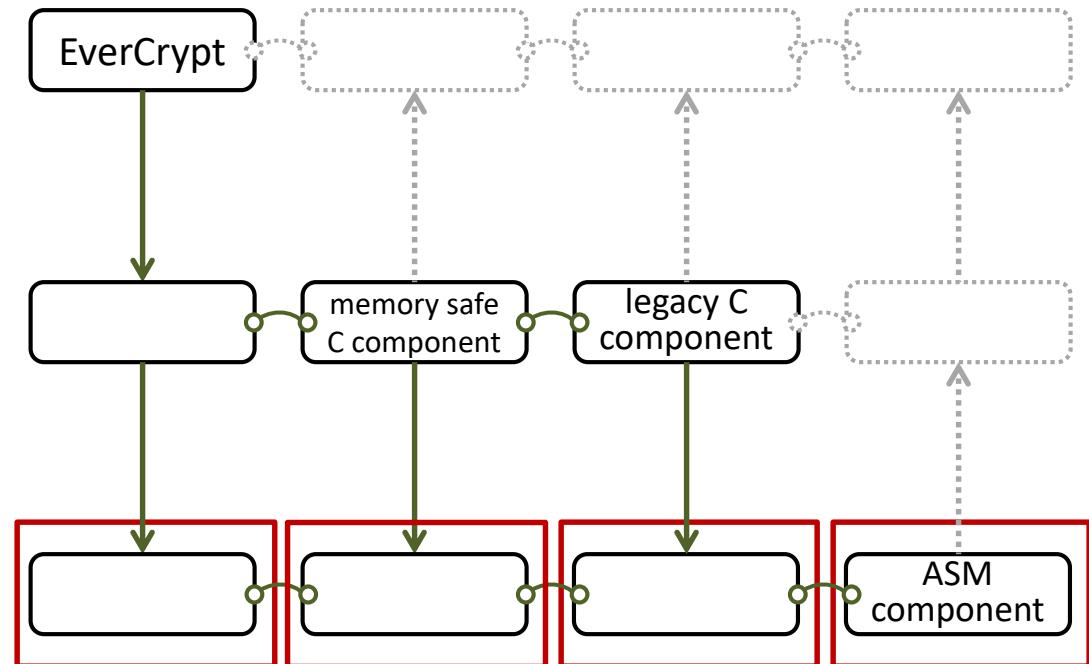
My dream: secure compilation at scale



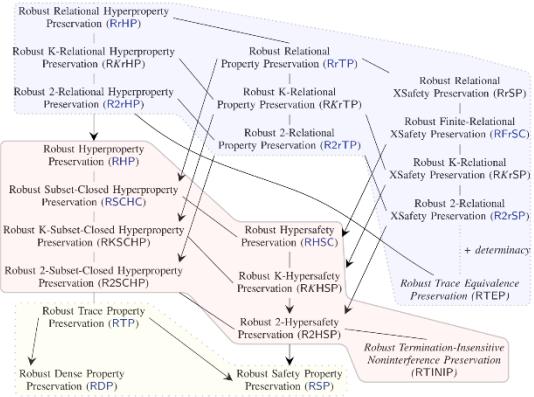
language

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



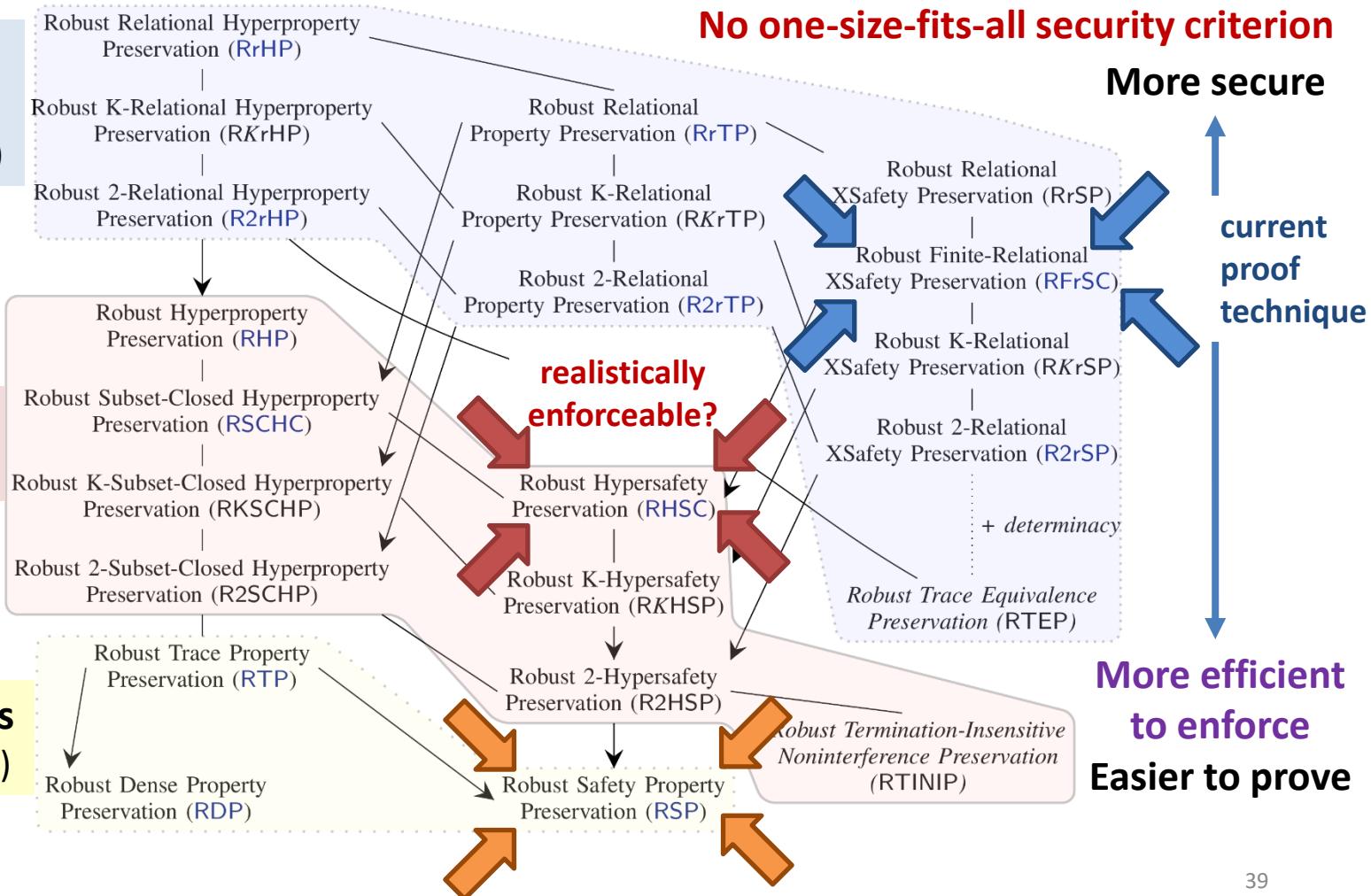
Marco
Patrignani
Stanford
& CISPA

Going beyond Robust Preservation of Safety [CSF'19]

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

hyperproperties
(noninterference)
+ data confidentiality

trace properties
(safety & liveness)
only integrity



Summary

Compartmentalizing compilation is an important security defense in practice



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

- first definition supporting **mutually distrustful components** and **dynamic compromise**



2. Enforcement: protect abstractions all the way down

- **software fault isolation** or **tag-based architecture**



3. Proof: verify security of entire compilation chain

- **scalable proof technique machine-checked in Coq**

