

# Formally Secure Compartmentalizing Compilation

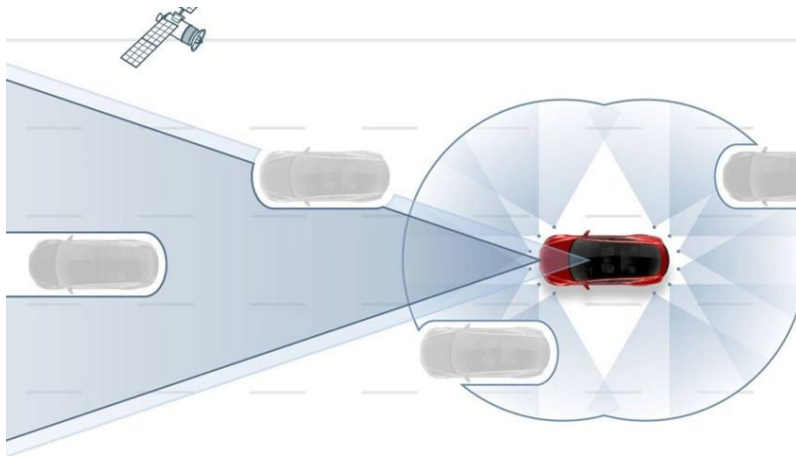
**Cătălin Hrițcu**

**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

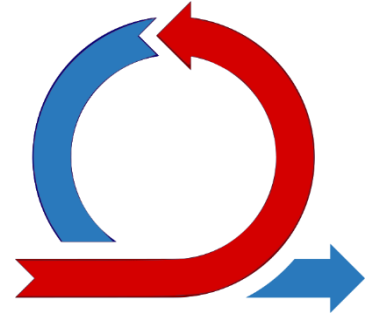


## Hackers Remotely Kill a Jeep on the Highway—With Me in It



# 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 image shows a screenshot of a web browser displaying the Spiegel.de website. The browser's address bar shows 'spiegel.de'. The website header includes the 'SPIEGEL ONLINE' logo and a search bar. A red box highlights the PlayStation Store logo in the top right corner. A large white box with black text is overlaid on the page, stating: 'Browser gets its input from the internet: a webpage (spiegel.de)'. Below this, another white box with black text says: '300+ resources loaded: html, image files, javascript, styles, ...'. A third white box with black text says: 'from 25+ different internet servers'. A fourth white box with black text says: '4 are clearly for ads:'. Below this, a red box highlights the URL 'ad.doubleclick.net'. A list of other ad-related URLs follows: '- ad.yieldlab.net', '- amazon-adsystem.com', and '- adalliance.io'. In the bottom right corner, there is a blue promotional banner for 'CALL OF DUTY: MODERN WARFARE\* OPERATOR ENHANCED EDITION' with a 'JETZT VORBESTELLEN' button. A 'Live' indicator is visible on the page. The background of the browser shows a news article with a photo of a man in a suit.

SPIEGEL ONLINE SPIEGEL

Suche Anmelden

PlayStation Store

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

CALL OF DUTY: MODERN WARFARE\* OPERATOR ENHANCED EDITION

BEINHALTE

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

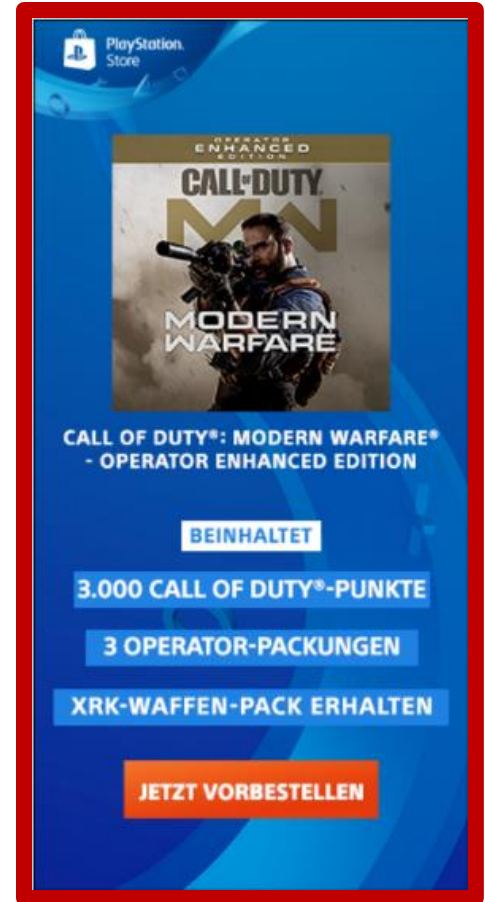
JETZT VORBESTELLEN

Live

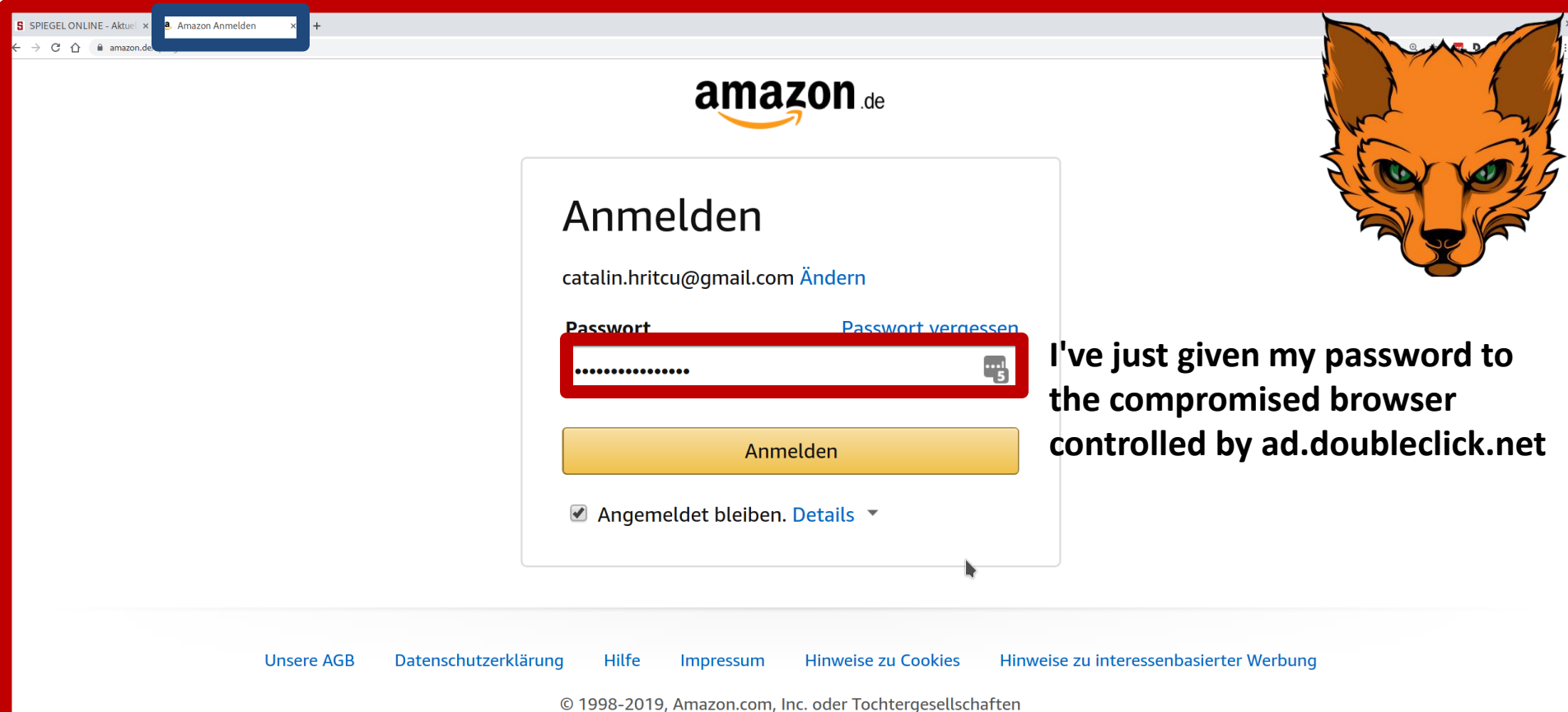
5

# Malicious server can hack the browser

- send it an image that **looks like an ad**
- **pecially 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



The image shows a browser window with the Amazon.de login page. The browser's address bar shows 'amazon.de'. The login form is titled 'Anmelden' and contains the following elements:

- Email: catalin.hritcu@gmail.com [Ändern](#)
- Password field: A red box highlights the password input field, which contains a masked password (dots) and a visibility toggle icon.
- Buttons: A yellow 'Anmelden' button and a checkbox for 'Angemeldet bleiben' with a [Details](#) link.

In the top right corner of the browser window, there is a cartoon illustration of a fox's head with green eyes.

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

At the bottom of the page, there are links for 'Unsere AGB', 'Datenschutzerklärung', 'Hilfe', 'Impressum', 'Hinweise zu Cookies', and 'Hinweise zu interessensbasierter Werbung'. The footer text reads: '© 1998-2019, Amazon.com, Inc. oder Tochtergesellschaften'.

# Compartmentalization can help

**compromised compartment 1**

**amazon.de password is still secure!**

**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 image shows a browser window with the Spiegel.de website. The browser's address bar shows 'spiegel.de'. The website header includes the 'SPIEGEL ONLINE' logo, the URL 'spiegel.de', a search icon, and an 'Anmelden' button. A navigation menu lists categories like 'Politik', 'Meinung', 'Wirtschaft', 'Panorama', 'Sport', 'Kultur', 'Netzwelt', and 'Wissenschaft'. The date '19. Oktober 2019' and stock market information 'DAX 12.633,60' are visible. Below the header, there are three advertisement tiles. The first tile is for 'adalliance.io' and features a man with a yellow banana. The second tile is for 'Hauswert-Rechner' and features a house with a red outline. The third tile is for 'Gleitsichtbrille' and features a man with glasses. A large red-bordered box labeled 'doubleclick.net' is overlaid on the right side of the page, containing a dense field of small, illegible text. Below the advertisements, a news article is visible with the headline 'Parlamentspräsident Bercow lässt Änderungsantrag zu - Brexit-Entscheidung könnte vertagt werden' and a 'Live' indicator. The article features a video player showing a man in a suit speaking.

# Fine-grained compartmentalization

The image shows a screenshot of the Spiegel.de login page with several blue-bordered boxes highlighting different sections. A red-bordered box highlights a Facebook.com advertisement. A red-bordered box highlights a password protection message. A red-bordered box highlights a list of services. A red-bordered box highlights a 'Mein Konto' section.

Mein SPIEGEL - SPIEGEL x +  
spiegel.de/meinspiegel/login.html

**SPIEGEL ONLINE** SPIEGEL **spiegel.de** **Anmelden**

☰ Menü | Politik Meinung Wirtschaft Panorama Sport Kultur Netzwelt Wissenschaft mehr ▾

**MEIN SPIEGEL** Schlagzeilen | DAX 12.894,51 | Abo

Nachrichten > Mein SPIEGEL

**Login** **spiegel.de**

Benutzername oder E-Mail-Adresse  
catalin.hritcu@gmail.com

Passwort  
.....

**Spiegel.de password is still protected**

**facebook.com**

**Meine Dienste**

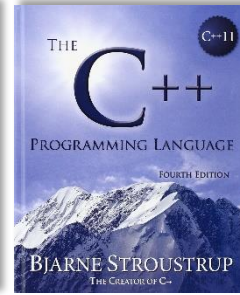
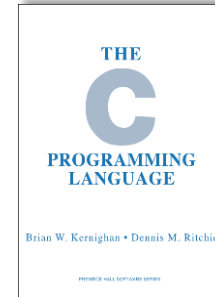
- Mein Börsendepot | Unternehmen
- Meine Abos
- Newsletter verwalten

Oder sind Sie neu hier?  
Registrieren Sie sich jetzt kostenlos bei Mein

**Mein Konto**

# 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  $C_1$  {**

**private var x;**

**private procedure p() {**

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

**x := password; ←not allowed**

**}**

**}**

**compartment  $C_2$  {**

**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

## Compartment $C_1$

<<check  $rx \in C_1$ >>

load  $r \leftarrow [rx]$  ← not allowed

put  $rc \leftarrow a_{\text{password}}$

<<check  $rx \in C_1$   
or  $rx \in C_2$ 's interface>>

jump-and-link  $rx$  ← not allowed

sub  $r \leftarrow r-1$

## Compartment $C_2$

put  $rc \leftarrow a_{\text{counter}}$

load  $r \leftarrow [rc]$

add  $r \leftarrow r+1$

store  $r \rightarrow [rc]$

jump  $ra$

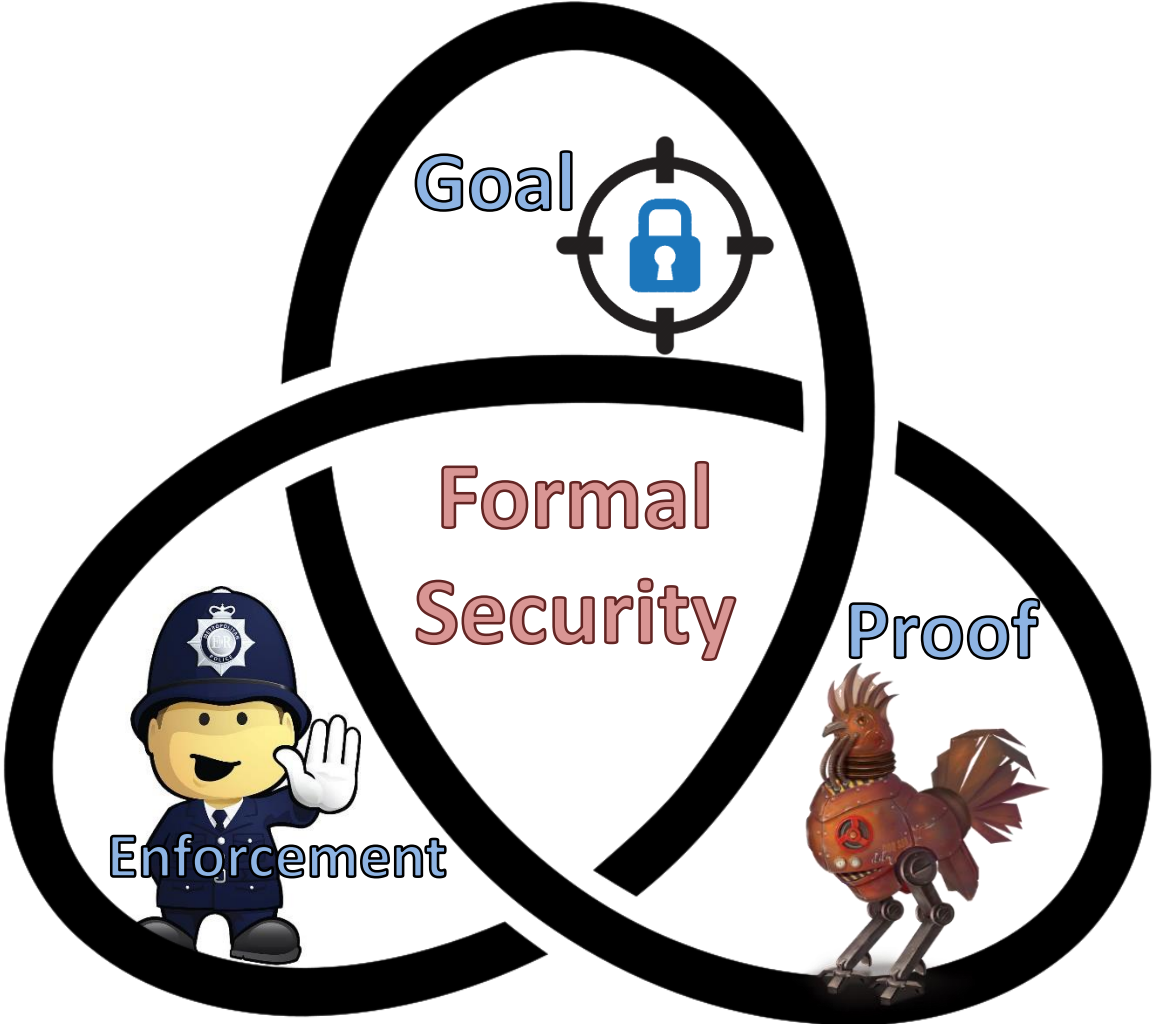
$a_{\text{counter}} : 42$

$a_{\text{password}} : \dots$

compiled  
get\_counter  
(public  
procedure)

**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

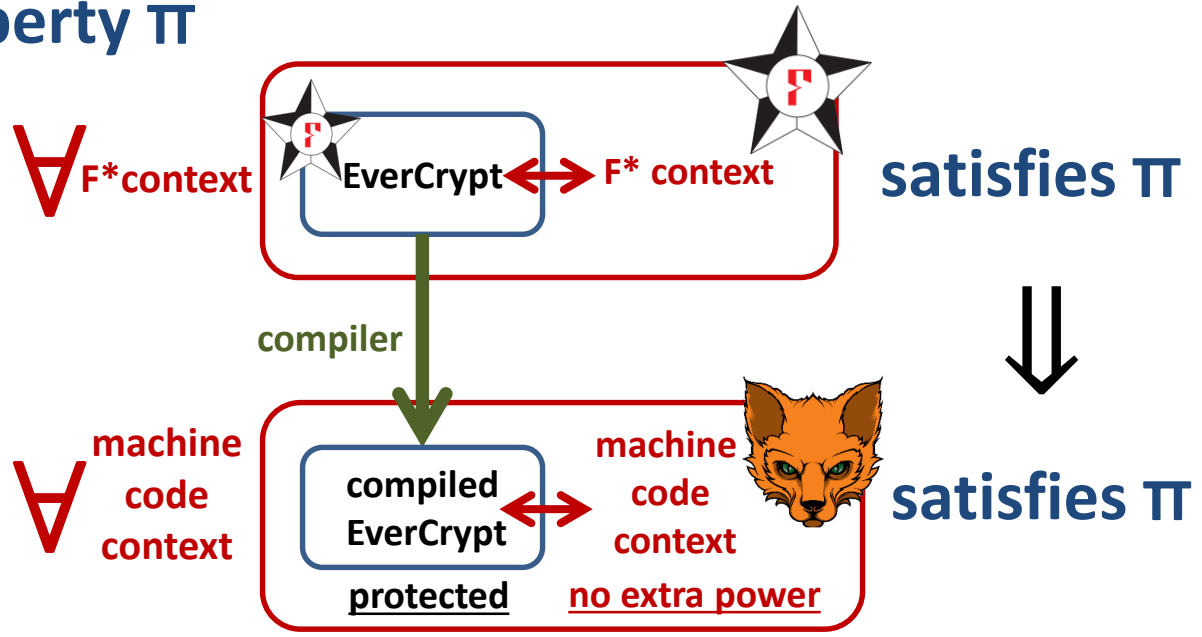
20.000.000 lines  
+ external libraries  
all unverified

**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  $\pi$



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

$\pi$  = "EverCrypt's private key is not leaked"

# Extra challenges for our real security definition

## [CSF'16, CCS'18]

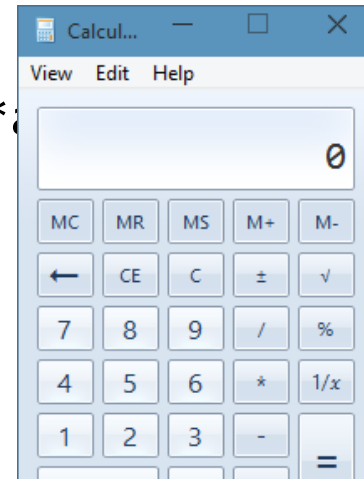
- Program split into **many mutually distrustful compartments**
- **We don't know which compartments will be compromised**
  - every compartment should be protected from all the others
- **We don't know when a compartment will be compromised**
  - every compartment should receive protection until compromised



# Formalizing security of mitigations is hard

- We want **source-level security reasoning principles**
  - easier to **reason about security in the source language** if and 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 **
    char c[12];
    strcpy(c, argv[1]);
    return 0;
}
```

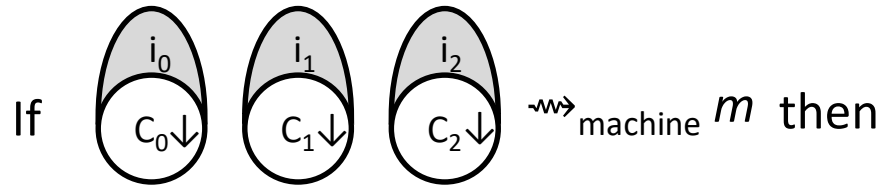


# Compartmentalizing compilation should ...

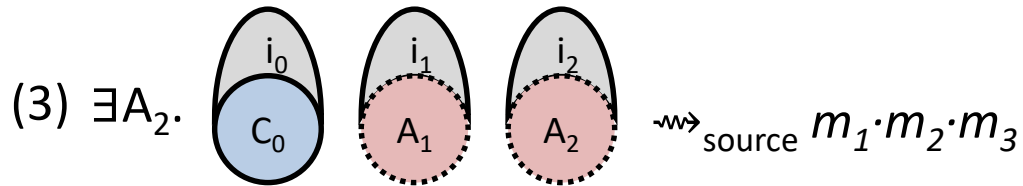
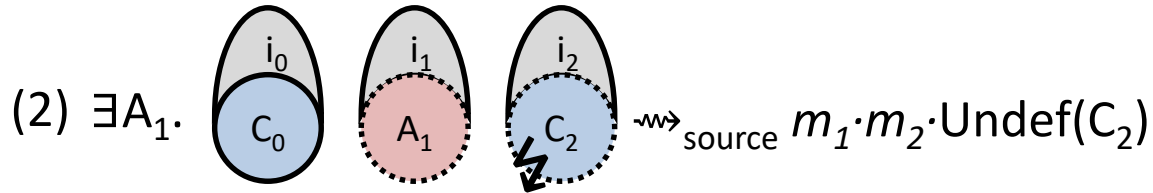
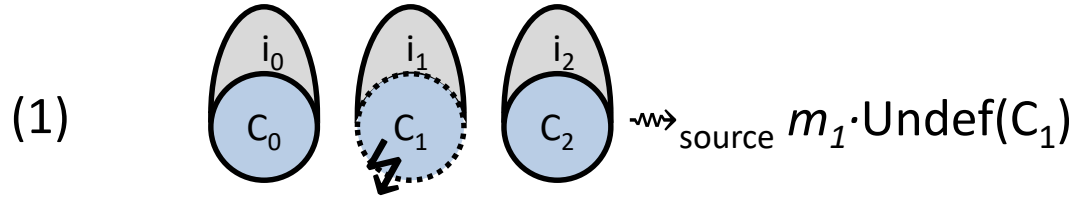
- **Restrict spatial scope** of undefined behavior
  - **mutually-distrustful components**
    - each component protected from all the others
- **Restrict temporal scope** of undefined behavior
  - **dynamic compromise**
    - each component gets guarantees as long as it has not encountered undefined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

# Security

**definition:**



$\exists$  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



**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;  
  }  
}
```



**Compartmentalized source language** 

Buffers, procedures, compartments

**Compartmentalized intermediate language** 

Intermediate language with built-in compartmentalization

 **Programmable tagged architecture** 

Hardware-accelerated enforcement

**Bare-bone machine**

Machine code

+Software enforcement

# Software-fault isolation

## Compartment $C_1$

**<<check  $rx \in C_1$ >>**

load  $r \leftarrow [rx]$  ←

put  $rc \leftarrow a_{\text{password}}$

**<<check  $rx \in C_1 \leftarrow$  not enough  
or  $rx \in C_2$ 's interface>>**

jump-and-link  $rx$  —

sub  $r \leftarrow r-1$

## Compartment $C_2$

$a_1$ : put  $rc \leftarrow a_{\text{counter}}$

$a_2$ : load  $r \leftarrow [rc]$

$a_3$ : add  $r \leftarrow r+1$

$a_4$ : store  $r \rightarrow [rc]$

$a_5$ : jump ra

$a_{\text{counter}} : 42$

$a_{\text{password}} : \dots$

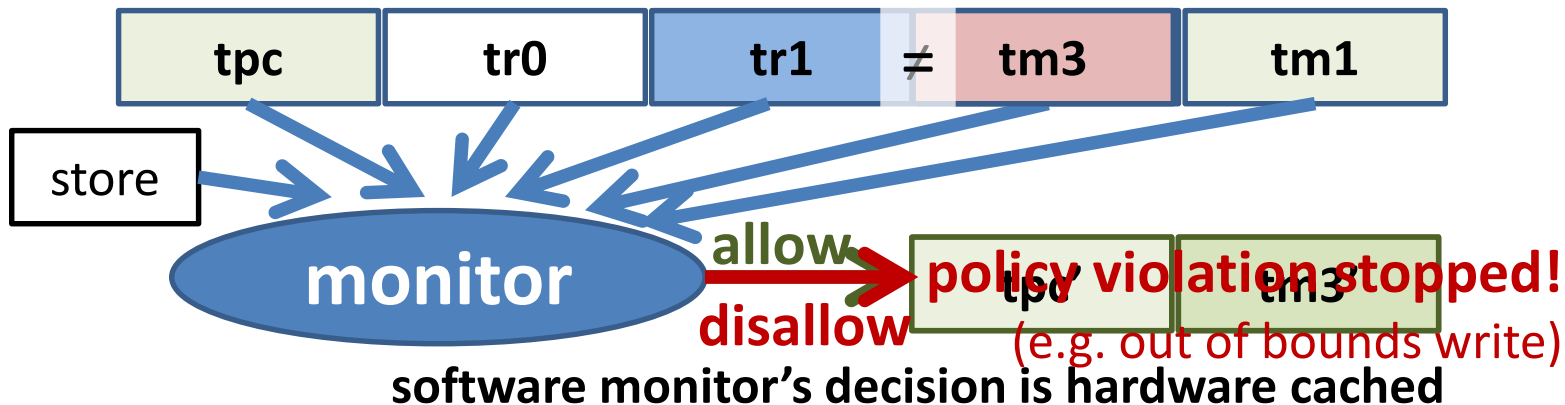
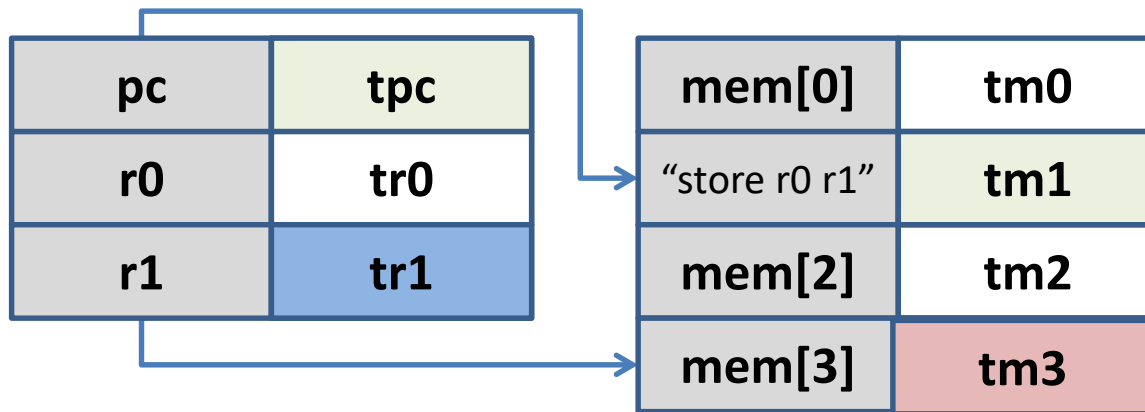
**Idea: rewrite  $C_1$ 's (&  $C_2$ 's) code to insert all the required checks**

**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  $rc \leftarrow a_{\text{password}}$

jump-and-link  $rx \leftarrow pc@C_1$

sub  $r \leftarrow r-1$  @NoEntry

not  
allowed

## Compartment $C_2$

$a_1$ : put  $rc \leftarrow a_{\text{counter}} @\text{EntryPoint}$

$a_2$ : load  $r \leftarrow [rc]$  @NoEntry

$a_3$ : add  $r \leftarrow r+1$  @NoEntry

$a_4$ : store  $r \rightarrow [rc]$  @ ...

$a_5$ : jump  $ra \leftarrow pc@C_2$

$a_{\text{counter}} : 42$

$a_{\text{password}} : \dots$

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

# 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**

**software fault isolation**

**Micro-policy  
machine** 

**Tag-based reference monitor enforcing:**

- component separation
- procedure call and return discipline  
(linear capabilities / linear entry points)

**Bare-bone  
machine**

**Inline reference monitor enforcing:**

- component separation
- procedure call and return discipline  
(program rewriting, shadow call stack)

**Systematically tested (with QuickChick)**



# 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



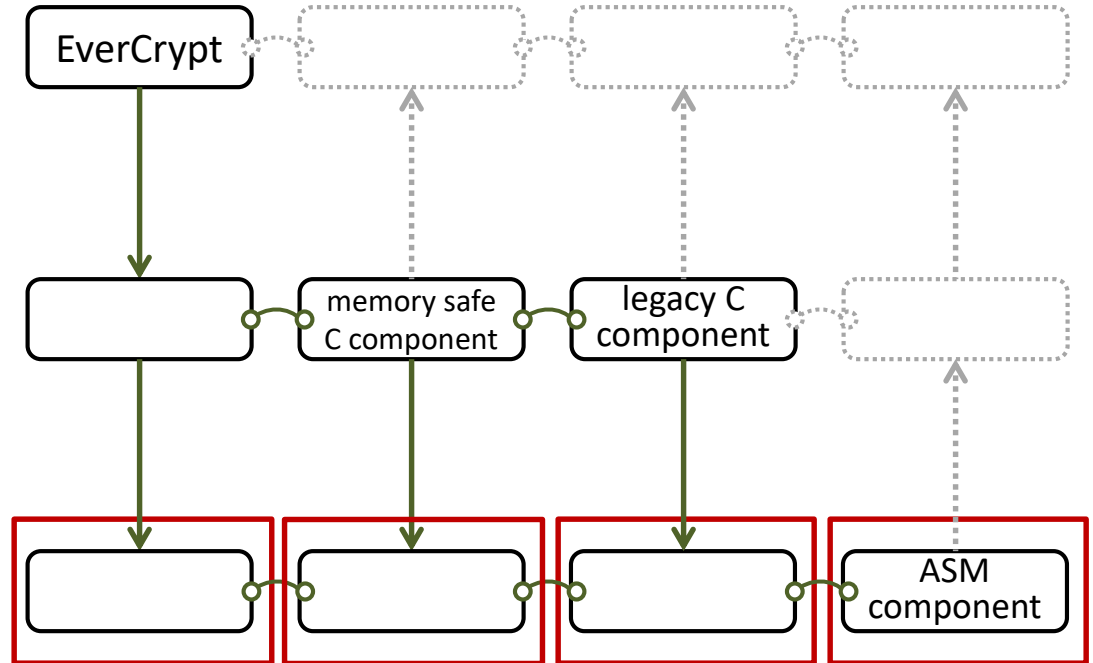
language

**C language**

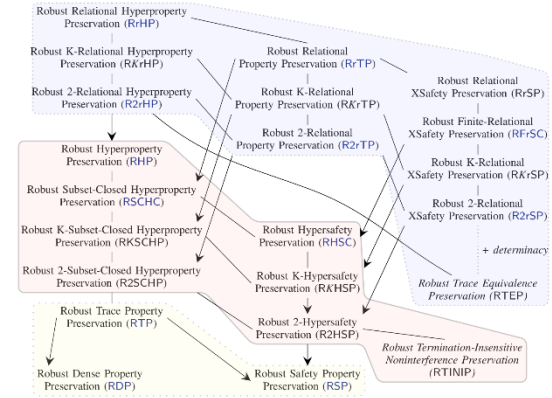
+ components  
+ memory safety

**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



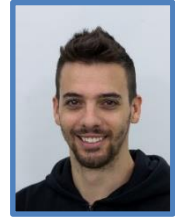
**Deepak Garg**  
MPI-SWS



**Cătălin Hrițcu**  
Inria Paris

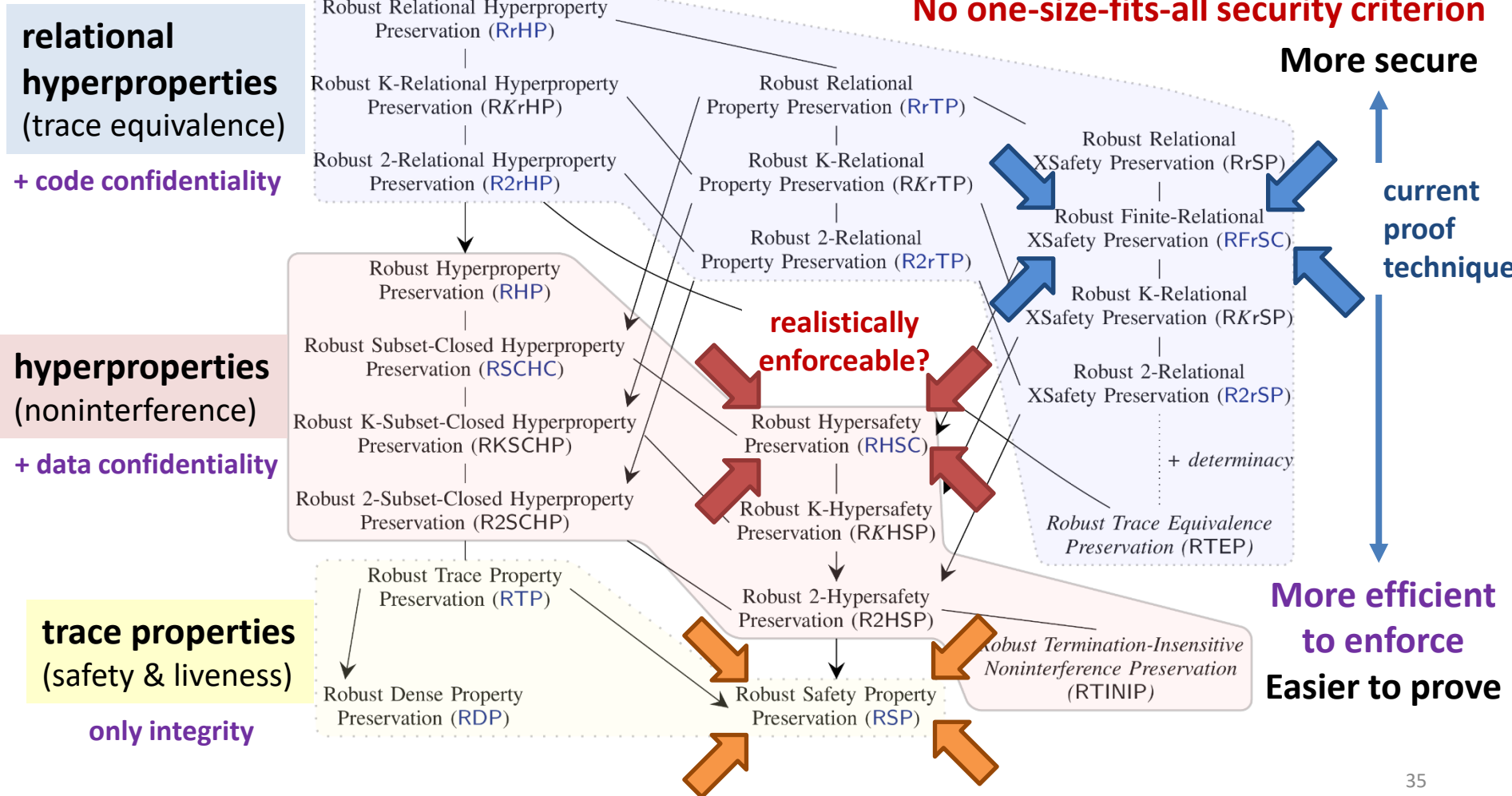


**Jérémy Thibault**  
Inria Paris



**Marco Patrignani**  
Stanford & CISA

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



# 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**

