What is secure compilation?

Security goals and attacker models

Cătălin Hrițcu

Inria Paris

What is secure compilation?

Devising **"more secure" compilation chains**

Compiler can play an important role ...

... but so can the **linker, loader, runtime,**
operating system, hardware, ...

... and various security **enforcement mechanisms**

Many enforcement mechanisms

- **safer languages** (RUST, WASM)
- **static analysis** & **verification**
- **program transformation** & **instrumentation** (SFI)
- **information flow control** (static, dynamic, hybrid)
- **dynamic monitors**
- **memory protection** (virtual memory, MPX, SSM)
- **enclaves** (SGX, TrustZone)
- **capability machines** (CHERI)
- **tagged hardware** (MicroPolicies)

Security is **hard** to enforce, so we will discuss a lot about **how** in this seminar (including on Wednesday afternoon)

What are we trying to enforce?

diverse security goals

Against what kinds of attacks?

diverse attacker models

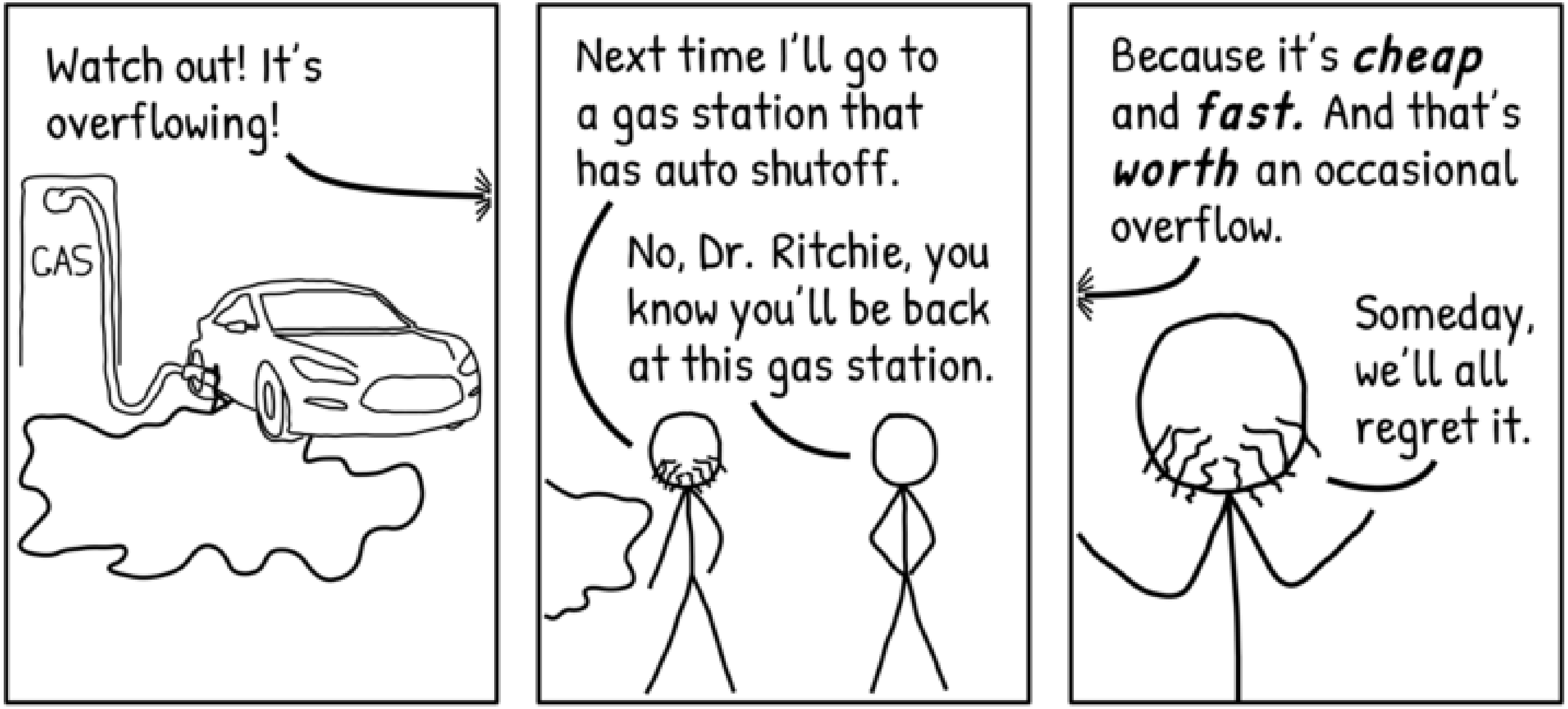
Safety in theory

Security goal:

- **Memory safety**
  - spatial and temporal memory violations lead to safe behavior (e.g. exception, termination)
- **Type safety**
  - e.g. invalid casts are safe
- **Less/no "undefined behavior"**

Attacker model:

- **Malicious inputs** 
  - tries to exploit lack of safety to take full control, mess with your data, obtain secrets, ...



Buffer Overflow.

icanbarelydraw.com

Safety in practice

Security goal:

- **Make exploits more difficult**
- Control-flow integrity
- Data-flow integrity
- Code-pointer integrity
- Stack protection
- Probabilistic guarantees
  (by randomization)
- ...

Attacker model:

1. **Attacker sends inputs**
   - exploiting safety vulnerability
2. **Attacker can access memory**
   - contiguous write,
   - arbitrary read, ...

... tries to:

- inject code or behavior,
- mess with your data,
- leak secrets, ...

Safety in practice

Security goal:

- **Limit attack damage**
  - only to the compromise of the **components** encountering undefined behavior (compartmentalization)

Attacker model:

1. **Attacker sends inputs**
   - exploiting safety vulnerability
2. **Attacker can access memory**
   - contiguous write,
   - arbitrary read, ...

... tries to:

- inject code or behavior,
- mess with your data,
- leak secrets, ...

Still, what are we trying to enforce?

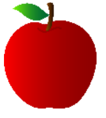
Security goal:

- **Integrity / encapsulation**
  - code, data, invariants
- **Confidentiality**
  - secrets don't get leaked
- **Availability**
  - no crashes or hangs (liveness)

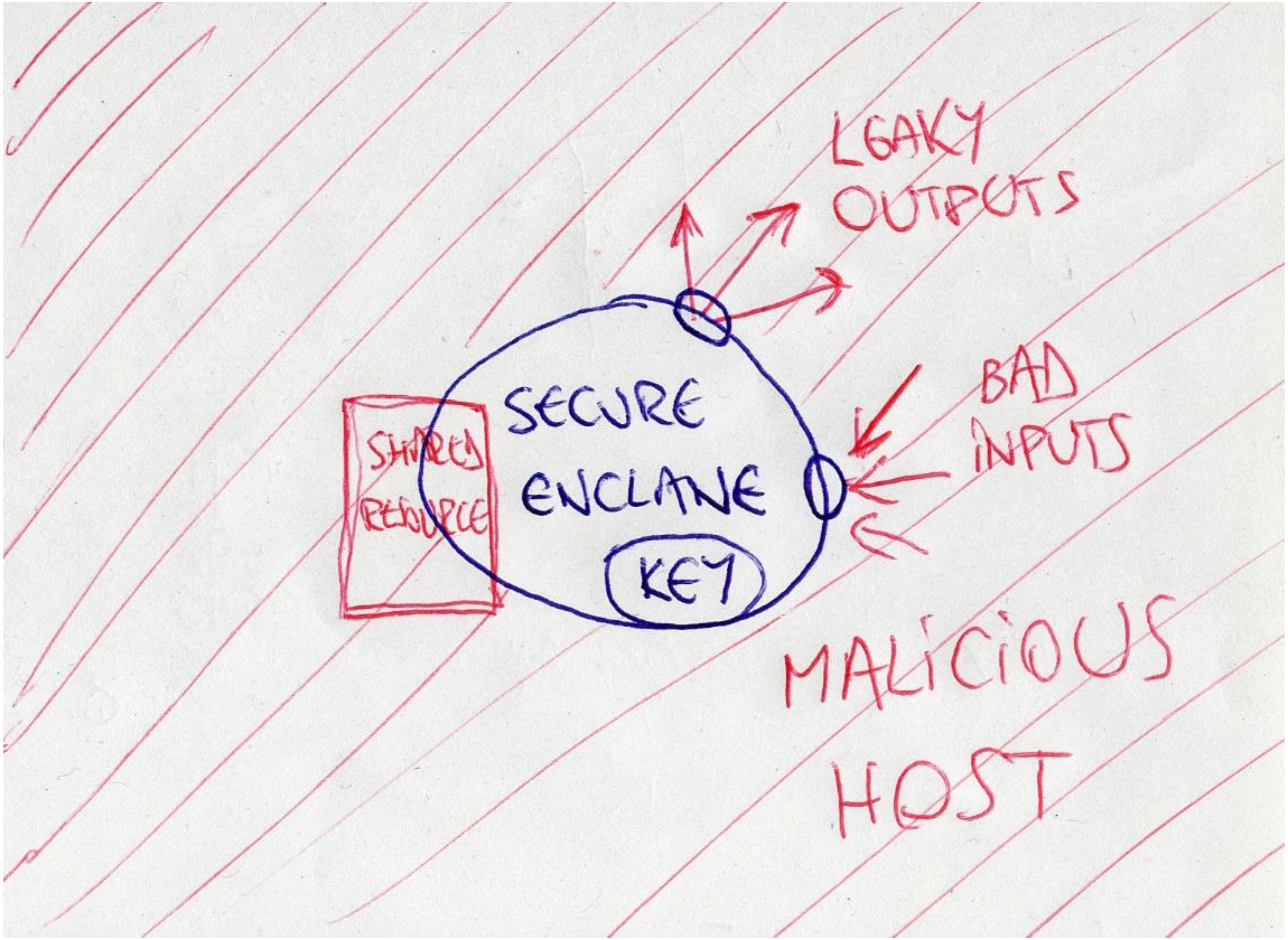
Attacker model:

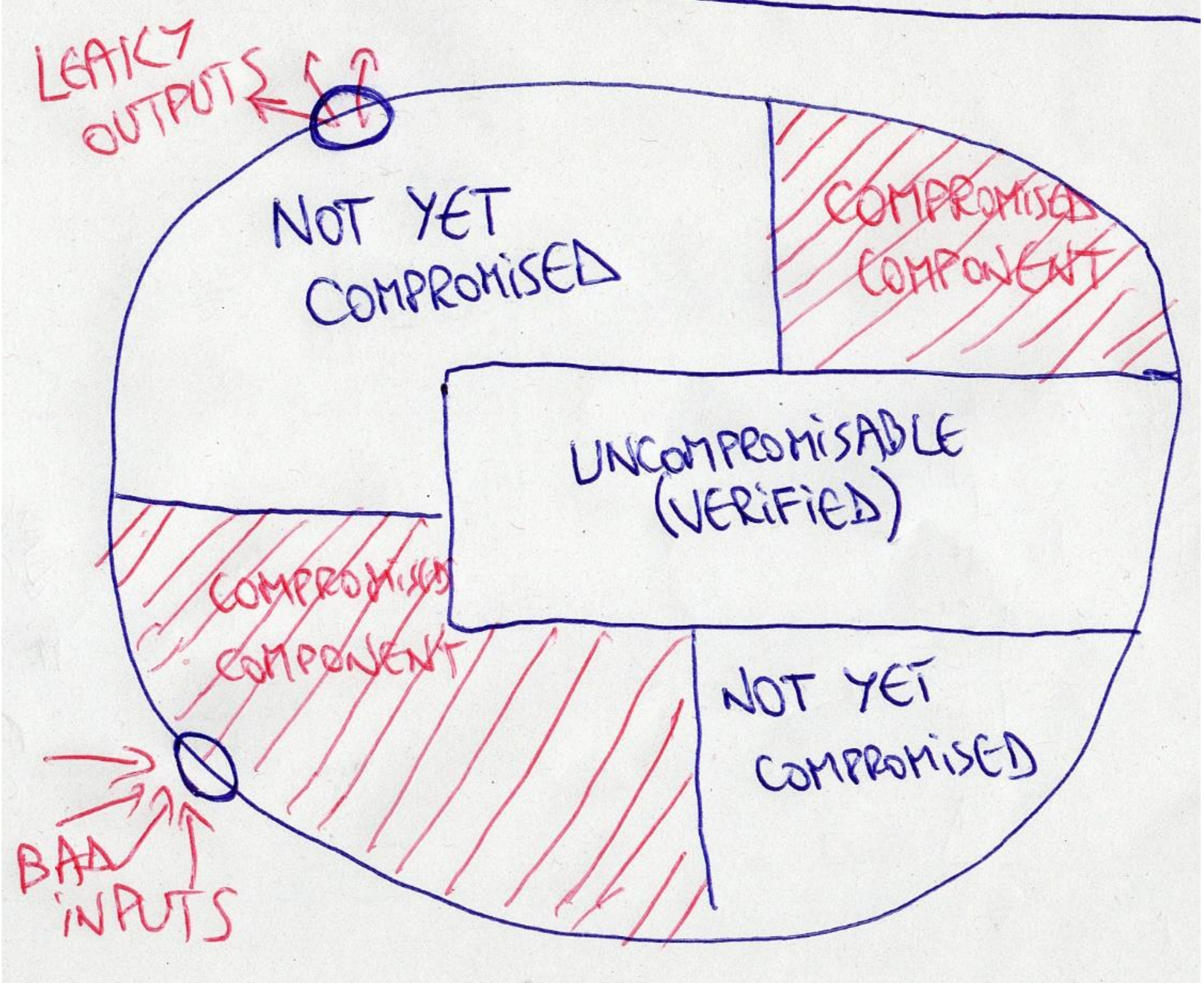
- **Malicious/compromised code**
  - component, library, plugin, host
- **Passive/active observer**
  - outputs, time, side-channels, ...

- **Malicious inputs**




UNTRUSTED PLUGIN
BAD INPUTS
UNTRUSTED PLUGIN
UNTRUSTED ~~PLUGIN~~ LIBRARY
HOST APPLICATION


LEAKY
OUTPUTS
BAD
INPUTS
SHARED
RESOURCE
SECURE
ENCLAVE
KEY
MALICIOUS
HOST


LEAKY OUTPUTS
NOT YET COMPROMISED
COMPROMISED COMPONENT
UNCOMPROMISABLE (VERIFIED)
COMPROMISED COMPONENT
NOT YET COMPROMISED
BAD INPUTS

Source-level security reasoning

- Frequent goal in **formally secure compilation**:

  Reason about security in the source language

  (or "the safe part" of the source language)

  – without needing to worry about compilation chain
- **No "low-level" attacks**
- **Watertight source language abstractions**

Source-level security reasoning

Preserving security of source programs

- **trace properties** (safety, liveness)
- **hyperproperties** (noninterference)
- **relational (hyper)properties** (obs. equivalence)

... against low-level attacks from

- **malicious "context"** (host, library, plugin)
- **compromised components**
- **powerful observer** (e.g. measuring time)

What is secure compilation?

1. **Making the source language safer** and **making it easier to express security intent**

2. **Making exploits more difficult**

3. **Enabling source-level security reasoning**

Backup questions

Enabling source-level security reasoning

1. How to relate source-target traces?
2. Does the attacker/context need to be represented as a program?

Deepak has more ...