Exploits as Insecure Compilation

Jennifer Paykin, Eric Mertens, Mark Tullsen, Luke Maurer, Benoît Razet, and Scott Moore

PriSC, January 25 2020

A compiler is secure if it doesn't introduce exploits.

A compiler is secure if it doesn't introduce exploits.

A compiler is insecure if it introduces exploits.

A compiler is secure if it doesn't introduce exploits.

A compiler is insecure if it introduces exploits.

- **how insecure is it?**

A compiler is secure if it doesn't introduce exploits.

A compiler is insecure if it introduces exploits.

- **how insecure is it?**
- **with respect to a particular program?**

Definition (Weird Machines)

The computational model made accessible by hacking a particular program.

Definition (Weird Machines)

The computational model made accessible by hacking a particular program.

1. Idealized, “correct” state machine specification that preserves security properties

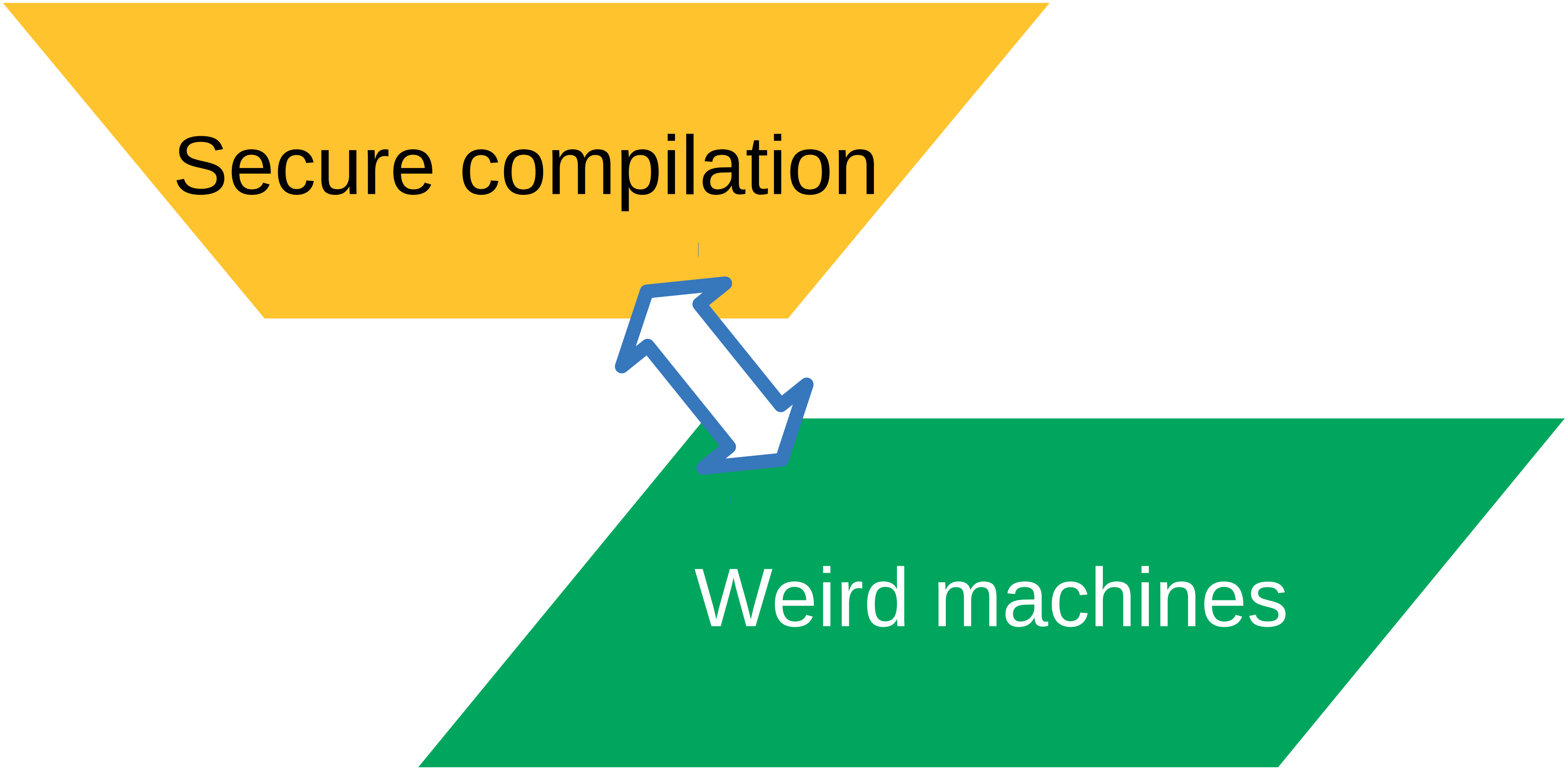
2. Concrete “implementation” model that admits additional behaviors

(Vanegue 2014, Dullien 2017, Bratus & Shubina 2017)

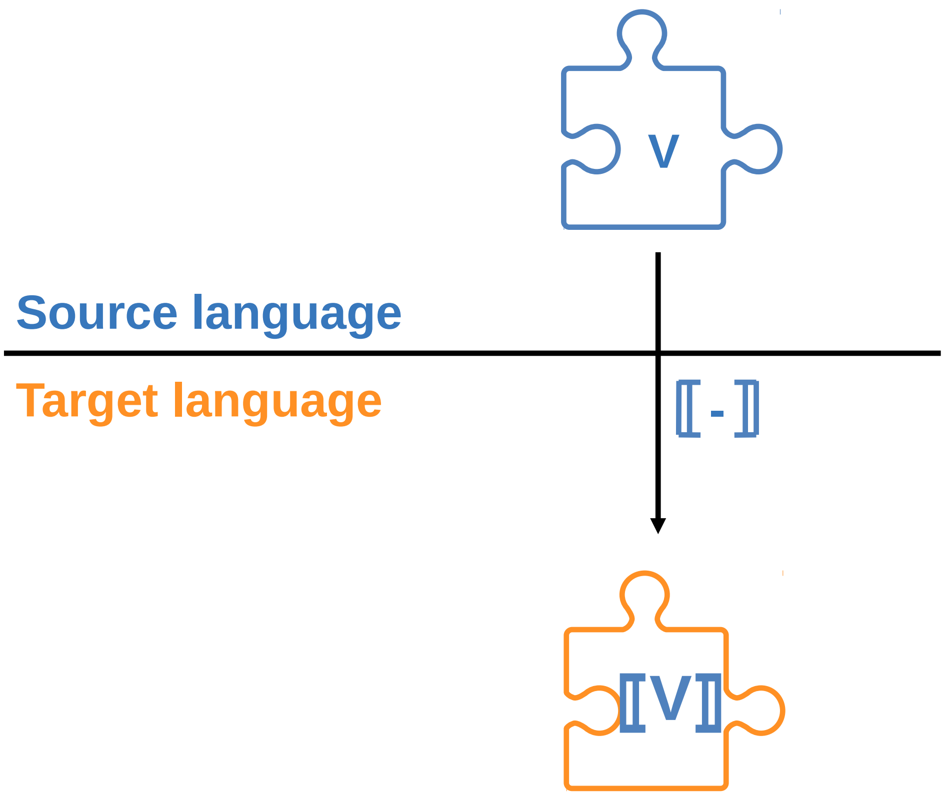
Insecure Compiler

1 program in high-level **source** language
for which security properties are enforced

2 implementation in low-level **target** language
that admits additional behaviors


Secure compilation
Weird machines

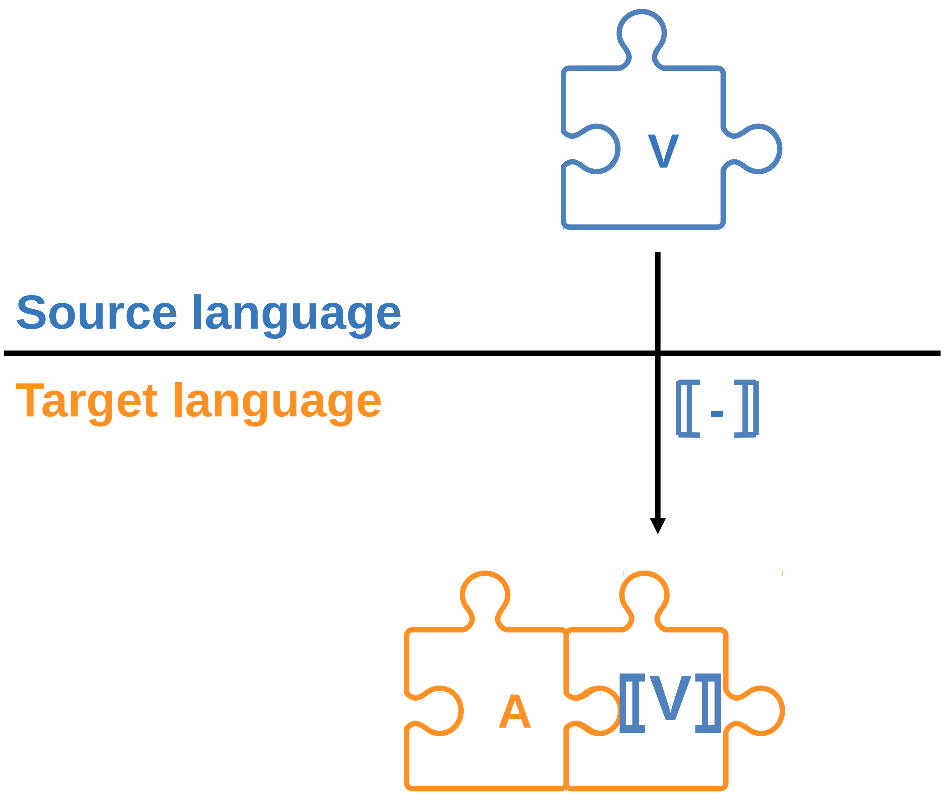
Exploits as violations of secure compilation



Definition

An *exploit*
of a source component **V**

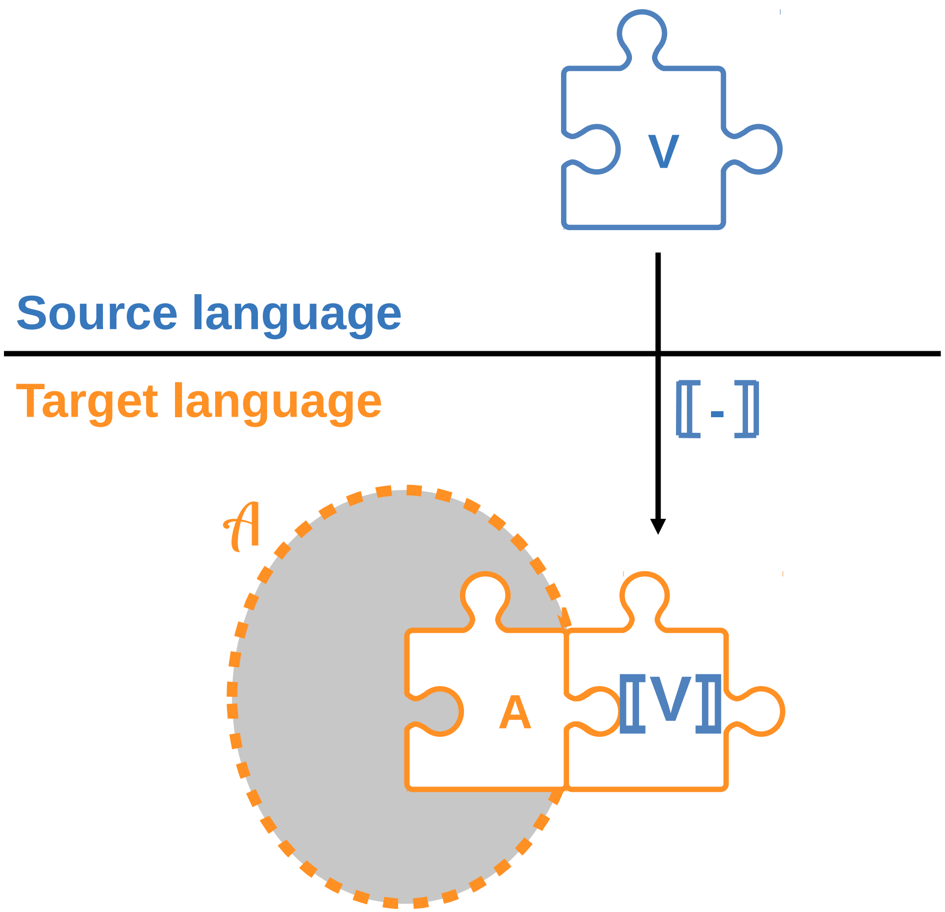
Exploits as violations of secure compilation



Definition

An *exploit*
of a source component **V**
is a context **A**

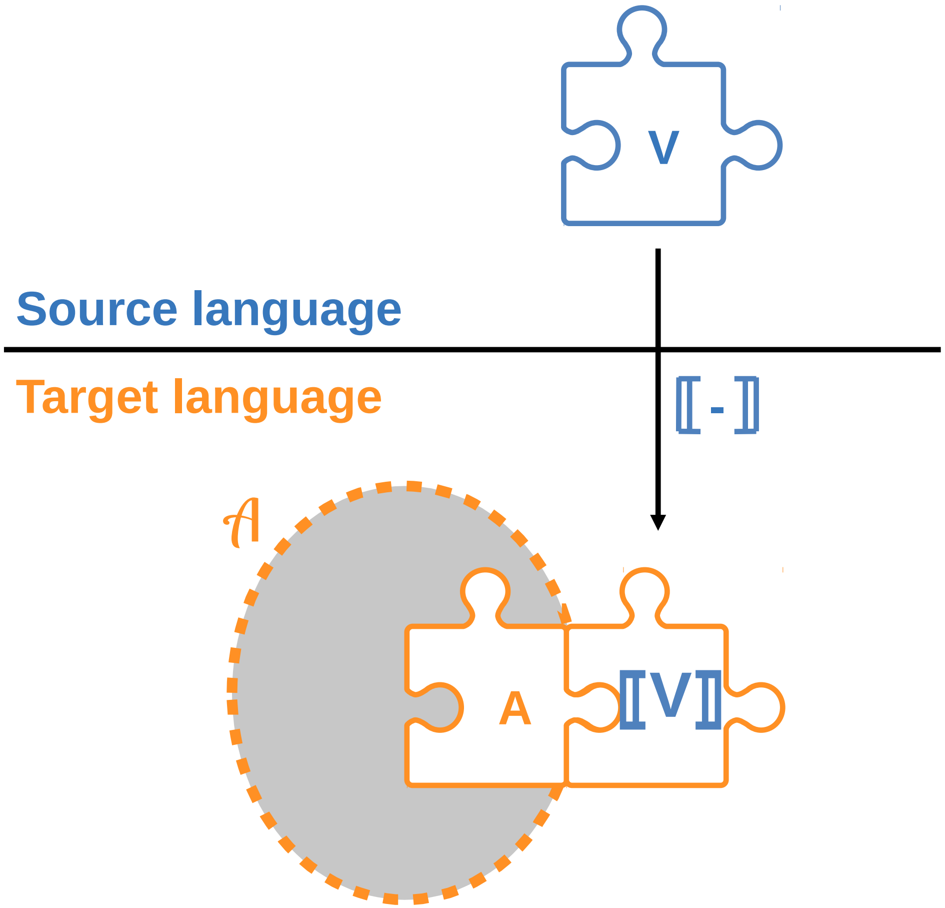
Exploits as violations of secure compilation



Definition

An *exploit* *of a source component* **V** is a context **A** from attack class $\mathcal{A}$

Exploits as violations of secure compilation

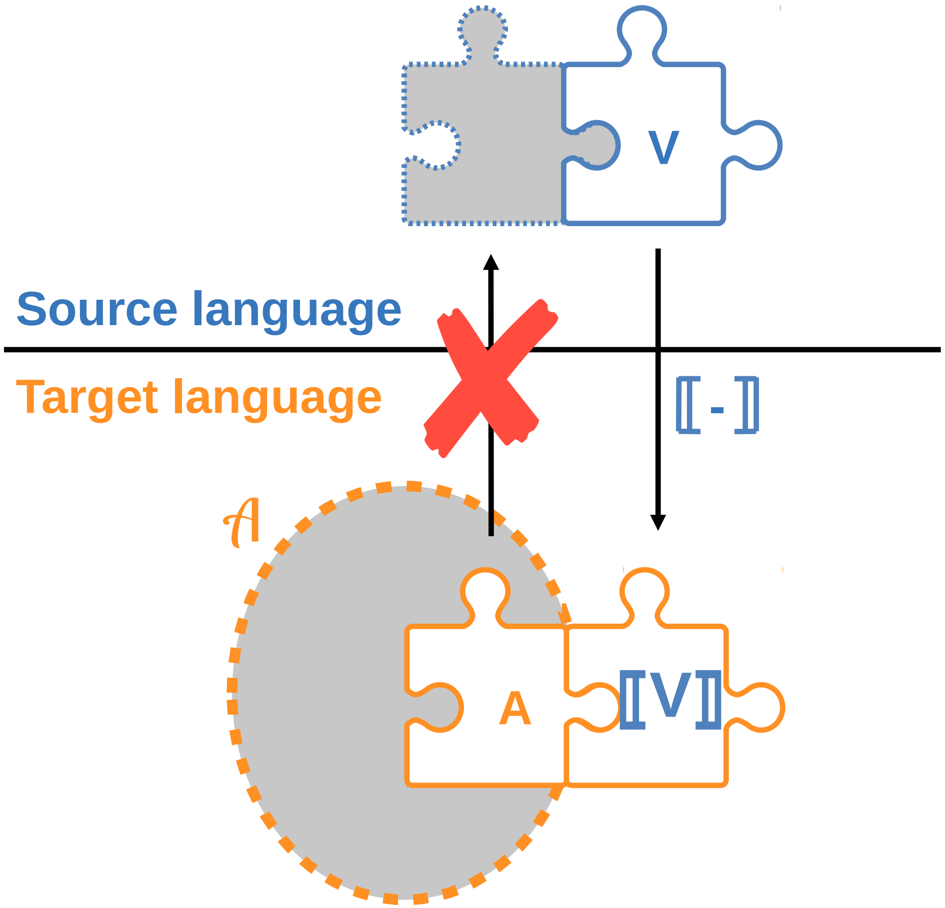


Definition

An *exploit*
of a source component **V**

is a context **A**
from attack class 𝒜
such that
the behavior of **A**[⟦V⟧]

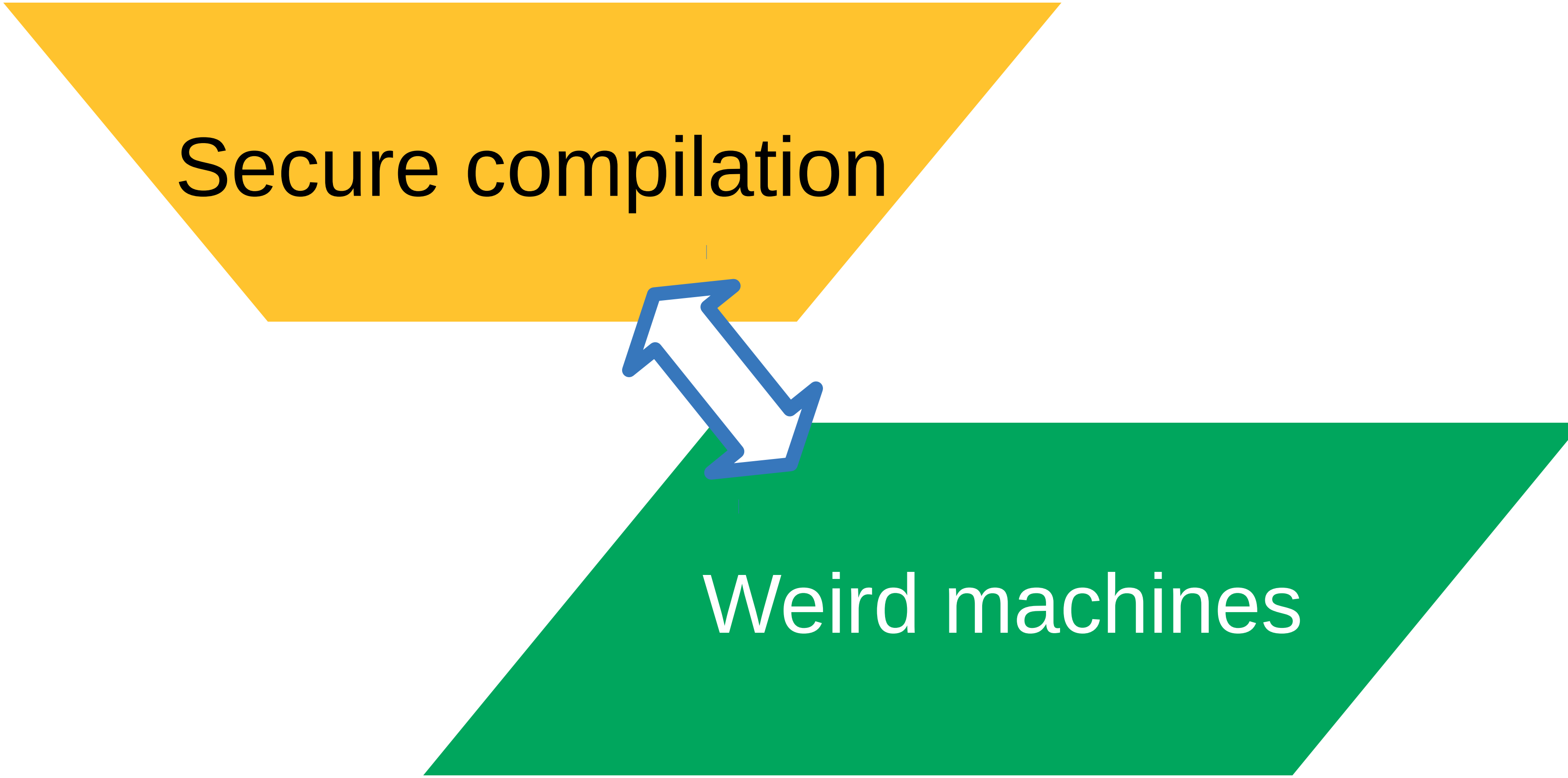
Exploits as violations of secure compilation



Definition

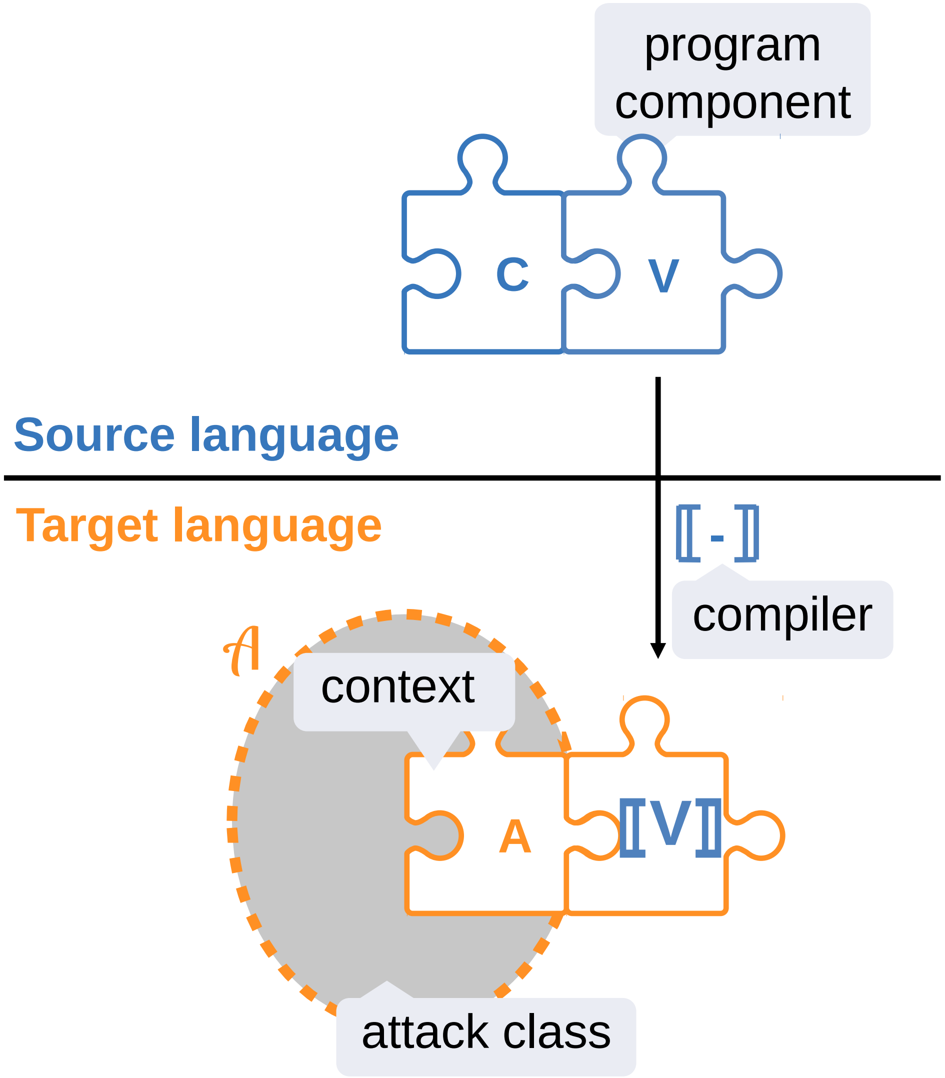
An *exploit*
of a source component **V**

is a context **A**
from attack class 𝒜
such that
the behavior of **A**[**⟦V⟧**]
cannot be simulated by **V**
in the source language.



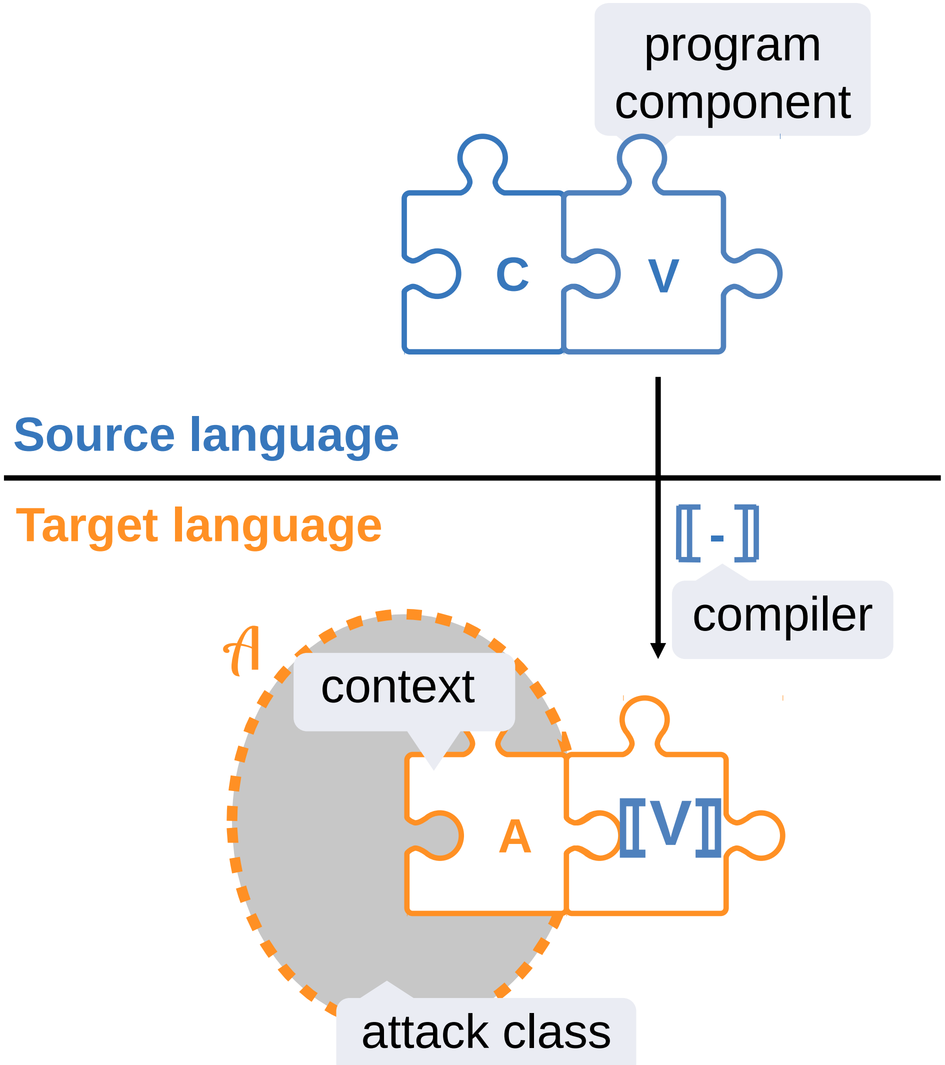
Hypothesis:
Definitions match intuitions

Framework



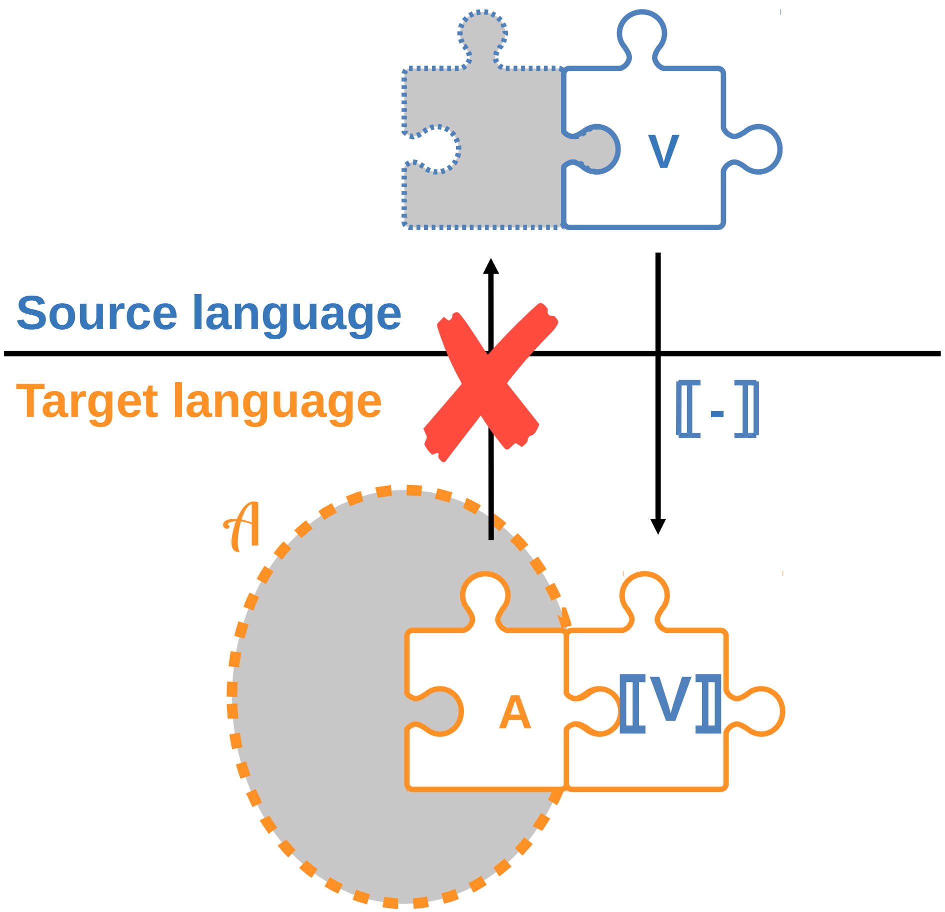
Exploit type	return-oriented programming (ROP)
Source	C
Compiler	clang
Target	assembly
Component	complete C program
Context	command-line input
Attack class	command-line input
Behavior	output traces

Framework



Exploit type	Spectre (Patrignani and Guarnieri 2020)
Source	non-speculative semantics
Compiler	no-op
Target	speculative semantics
Component	program in memory
Context	memory, cache, PC, etc...
Attack class	prepare cache, invoke function, query cache...
Behavior	timing information

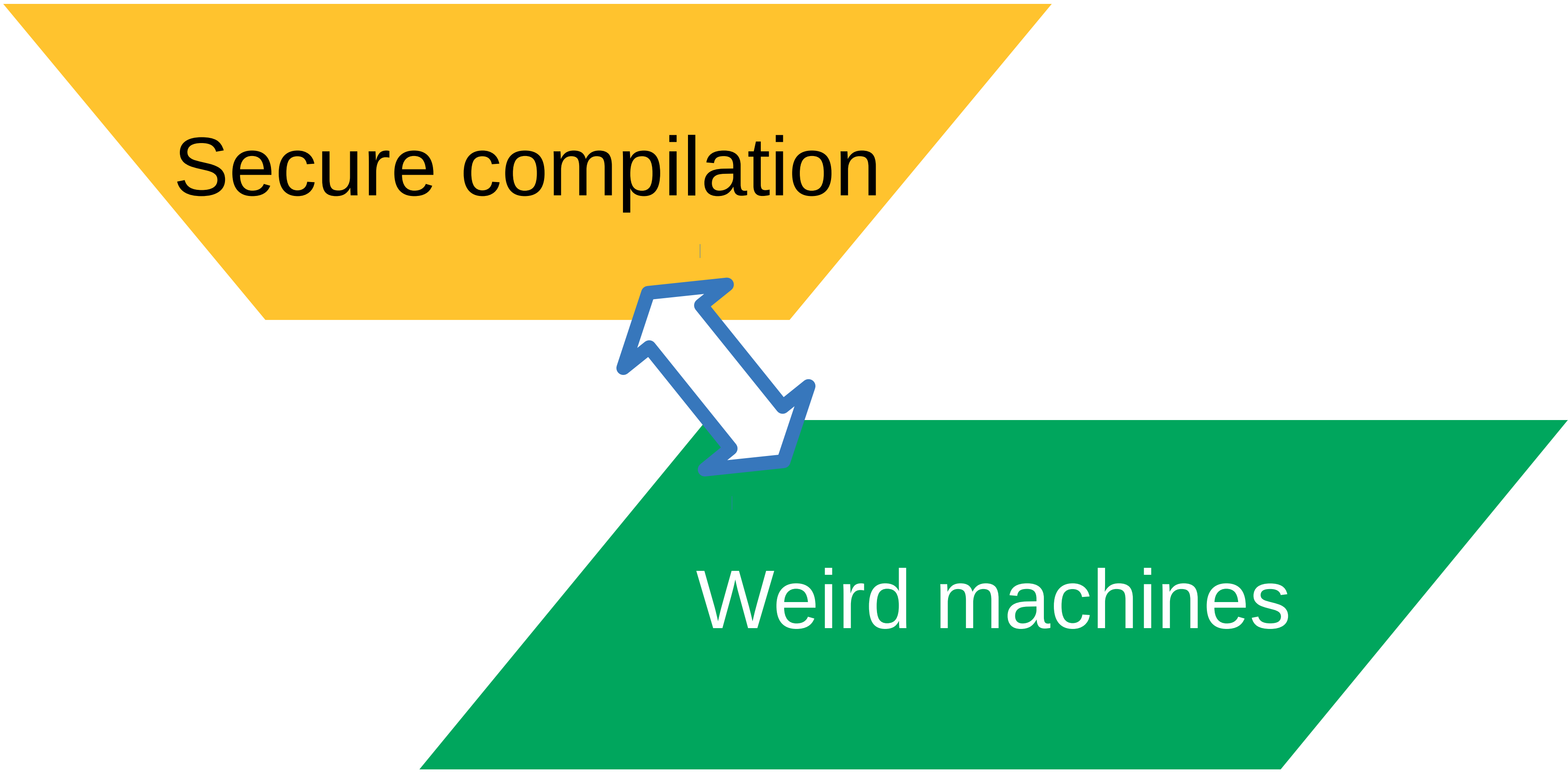
Exploits as violations of secure compilation



Definition

An *exploit*
of a source component **V**

is a context **A**
from attack class 𝒜
such that
the behavior of **A**[⟦V⟧]
cannot be simulated by **V**
in the source language.



Constructive procedure to answer:

Is **A** an exploit of **V**?

Robust Property Preservation

Definition (Abate et al 2019)

A compiler satisfies *robust hyper-property preservation* (RHP) if, ∀ source programs **V** and ∀ hyper-properties H ⊆ B:

$$(\forall \mathbf{C}^{\mathbf{S}}.\ \text{Behavior}(\mathbf{C}^{\mathbf{S}}[\mathbf{V}]) \in H) \Rightarrow$$
$$(\forall \mathbf{C}^{\mathbf{T}}.\ \text{Behavior}(\mathbf{C}^{\mathbf{T}}[[\![\mathbf{V}]\!]]) \in H)$$

* approx: behaviors = sets of traces, so H is a set of (set of traces)

Robust Property Preservation

Definition (Abate et al 2019)

A compiler satisfies *robust hyper-property preservation* (RHP) if, ∀ source programs **V** and ∀ hyper-properties $H \subseteq B$:

$$(\forall C^S.\ \mathrm{Behavior}(C^S[V]) \in H) \Rightarrow$$
$$(\forall C^T.\ \mathrm{Behavior}(C^T[[\![V]\!]]) \in H)$$

Theorem (Abate et al 2019)

A compiler satisfies RHP iff ∀ source programs **V:**

$$\forall C^T, \exists C^S.\ \mathrm{Behavior}(C^S[V]) = \mathrm{Behavior}(C^T[[\![V]\!]]).$$

* approx: behaviors = sets of traces, so H is a set of (set of traces)

Robust Property Preservation

$\forall C^T, \exists C^S . \text{Behavior}(C^S [V]) = \text{Behavior}(C^T[[\![V]\!]])$.

Robust Property Preservation

$\forall C^T, \exists C^S . \text{Behavior}(C^S [V]) = \text{Behavior}(C^T[[\![V]\!]])$.

Definition

An exploit of a source programs V is a context $A \in \mathcal{A}$ such that

$\neg\exists C^S . \text{Behavior}(C^S [V]) = \text{Behavior}(C^T[[\![V]\!]])$.

Robust Property Preservation

Definition

An exploit of a source program **V** is a context **A** ∈ $\mathcal{A}$ such that

$$\forall \mathbf{C^S} \,.\, \text{Behavior}(\mathbf{C^S}[\mathbf{V}]) \neq \text{Behavior}(\mathbf{C^T}[[\![\mathbf{V}]\!]]).$$

Robust Property Preservation

Definition

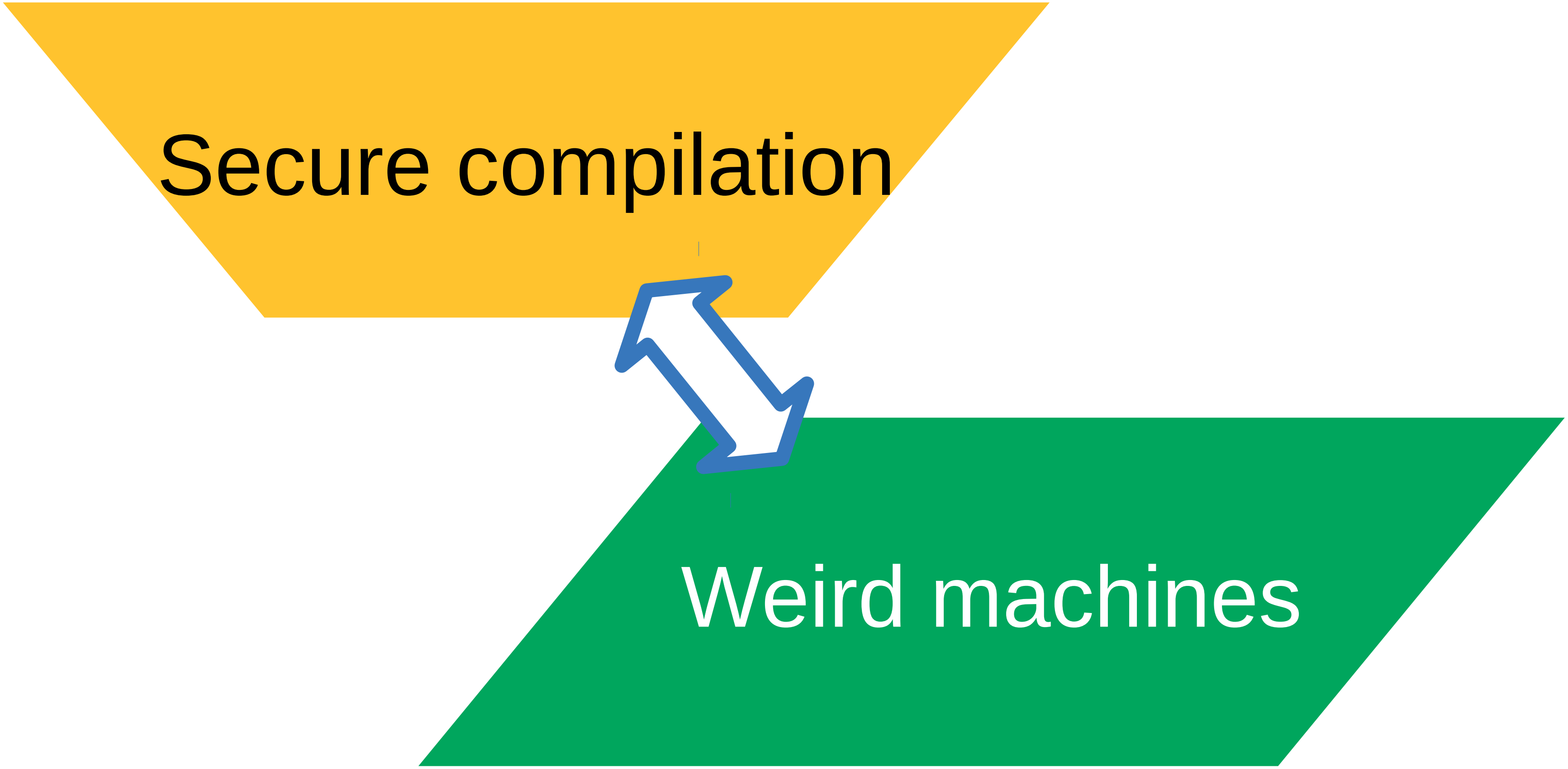
An exploit of a source program $\mathbf{V}$ is a context $\mathbf{A} \in \mathcal{A}$ such that

$$\forall \mathbf{C}^{\mathbf{S}} .\ \text{Behavior}(\mathbf{C}^{\mathbf{S}}\ [\mathbf{V}]) \neq \text{Behavior}(\mathbf{C}^{\mathbf{T}}[[\![\mathbf{V}]\!]]).$$

Theorem

$\mathbf{A}$ is an exploit of $\mathbf{V}$ iff RHP is violated:
$\exists$ hyper-property $H \subseteq B$ such that

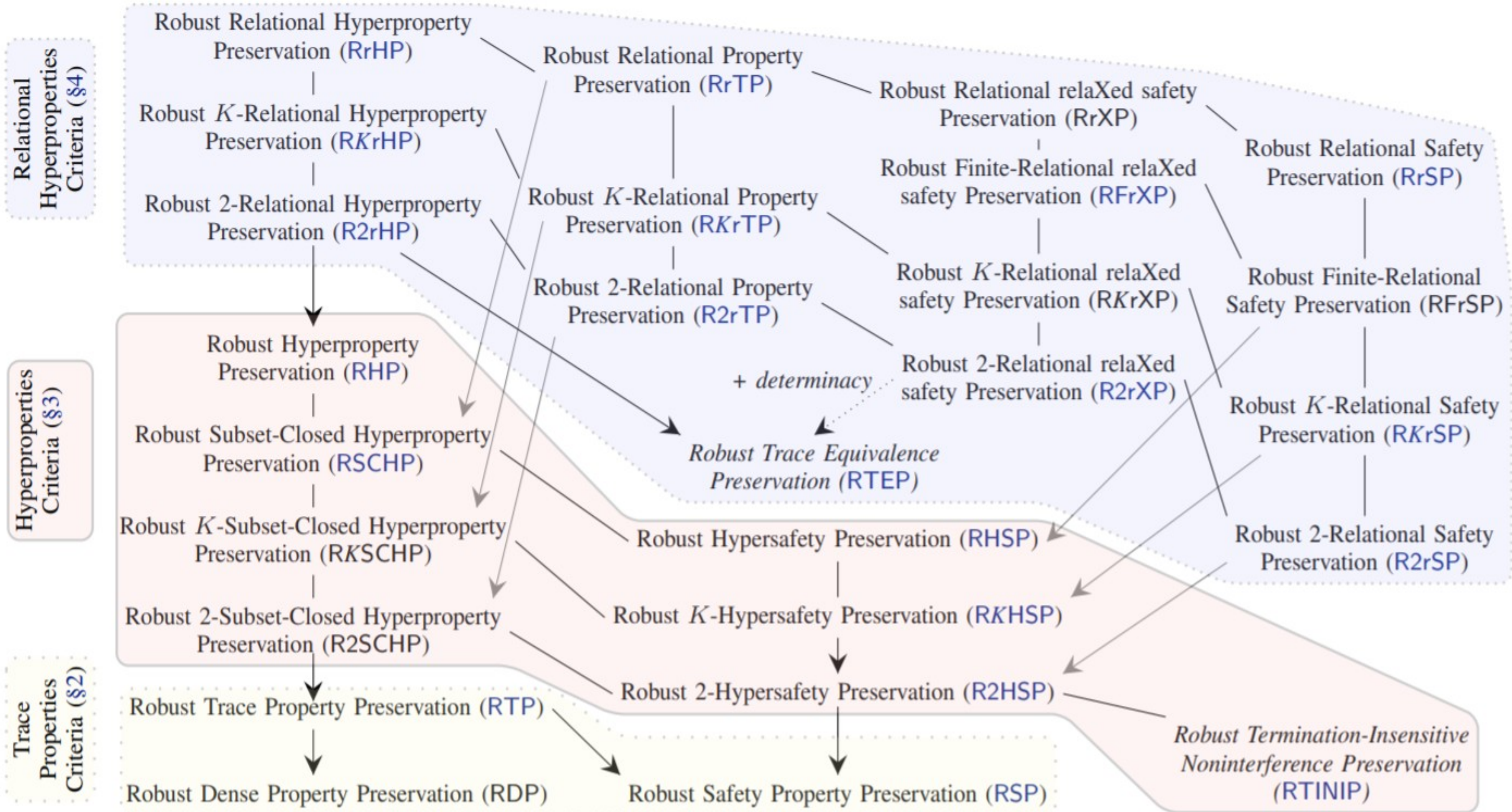
$$(\forall \mathbf{C}^{\mathbf{S}}.\ \text{Behavior}(\mathbf{C}^{\mathbf{S}}[\mathbf{V}]) \in H)$$
$$\text{but} \quad \text{Behavior}(\mathbf{A}[[\![\mathbf{V}]\!]]) \notin H)$$



different security properties

= different attack classes

Hierarchy of robust property preservation classes



Abate et al. 2019

Hierarchy of exploit classes

1 identify a class of security properties of interest

2 identify property-free characterization

3 exploit class is negation of property-free characterization

CFI?

Full abstraction?

Trace Property Preservation

Definition

A *trace exploit* of a source program **V** is a context **A** $\in \mathcal{A}$ such that

$$\exists\, t \in \text{Behavior}(\mathbf{A}[[\![\mathbf{V}]\!]]).$$

$$\forall\, \mathbf{C^S},\ t \notin \text{Behavior}(\mathbf{C^S}[\mathbf{V}])$$

Trace Property Preservation

Definition

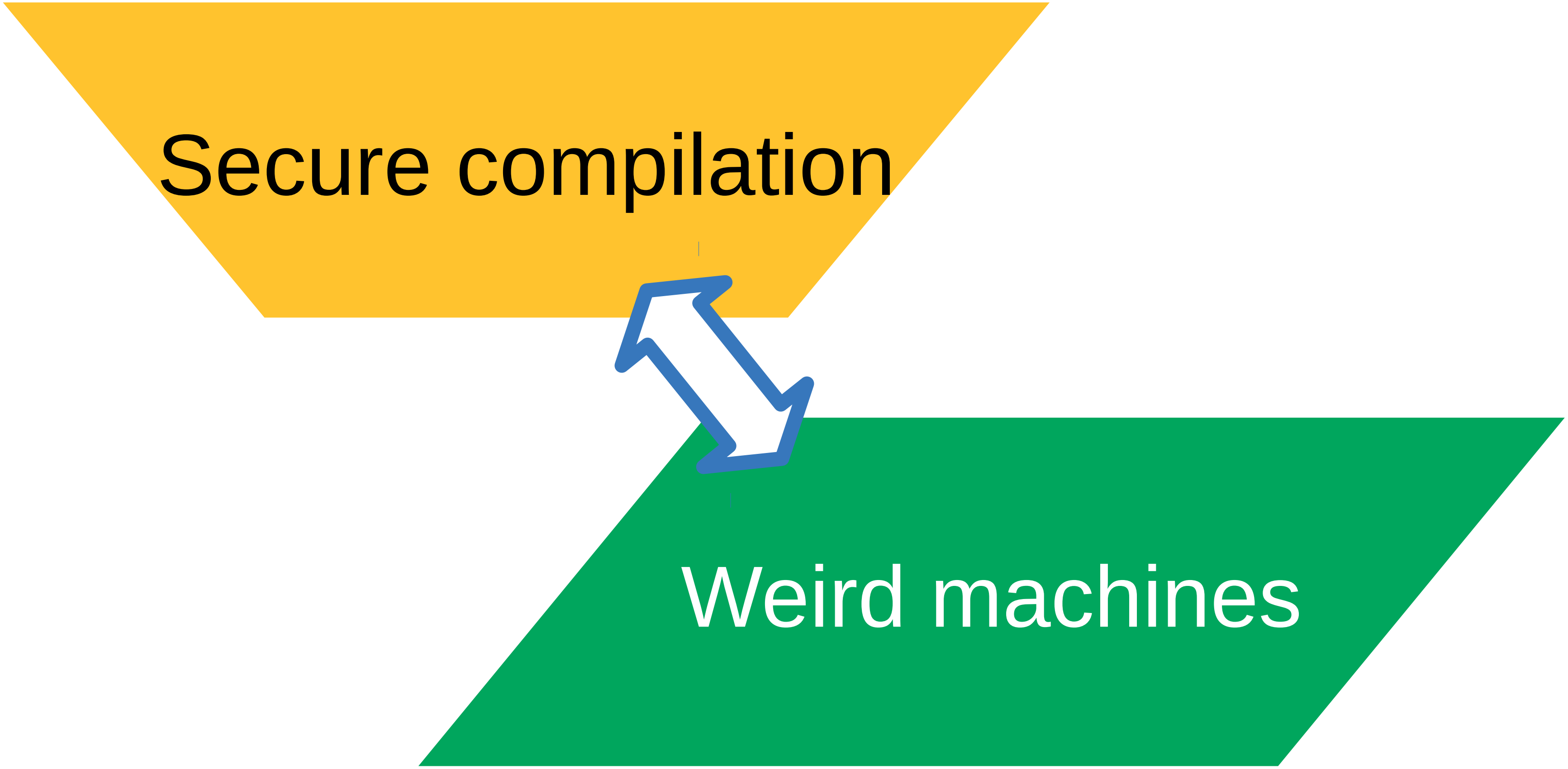
A *trace exploit* of a source program $\mathbf{V}$ is a context $\mathbf{A} \in \mathcal{A}$ such that

$$\exists\, t \in \text{Behavior}(\mathbf{A}[\llbracket \mathbf{V} \rrbracket]).$$

$$\forall\, \mathbf{C}^{\mathbf{s}},\ t \notin \text{Behavior}(\mathbf{C}^{\mathbf{s}}[\mathbf{V}])$$

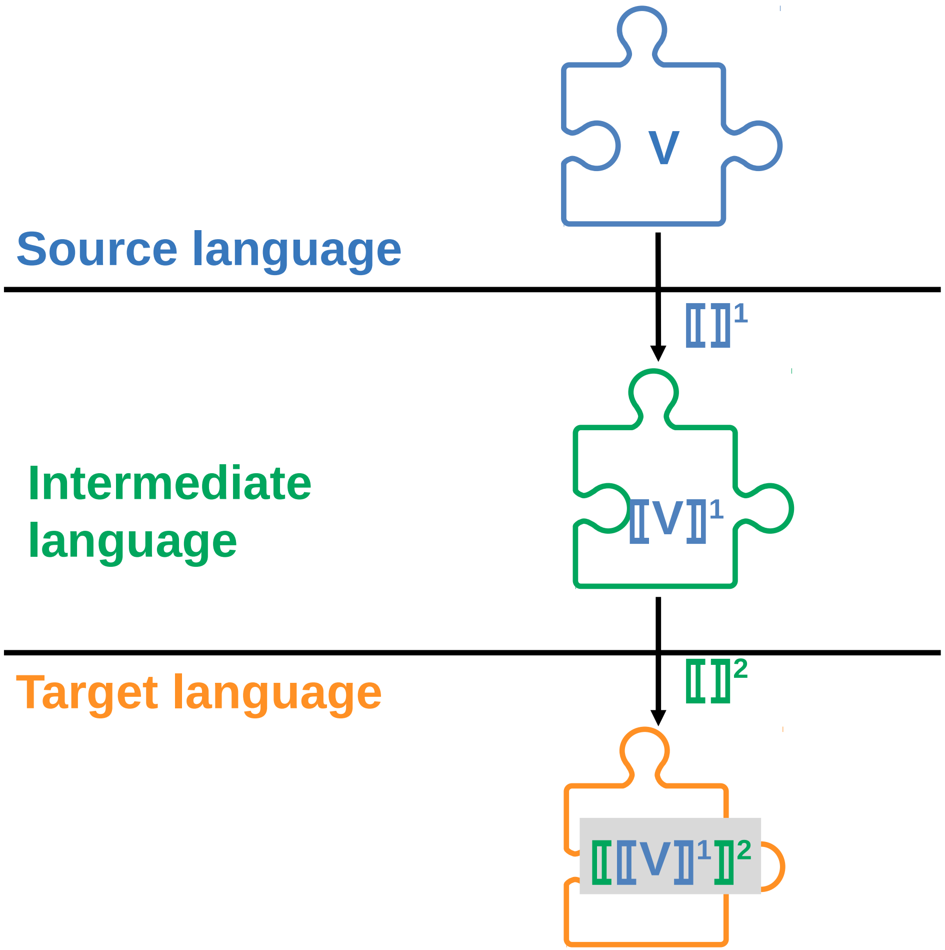
Theorem

- trace exploits ⊆ hyperproperty exploits.
- hyperproperty exploits ⊈ trace exploits
- e.g. side-channel attacks
- Trace exploits “more programmable” than hyperproperty exploits.

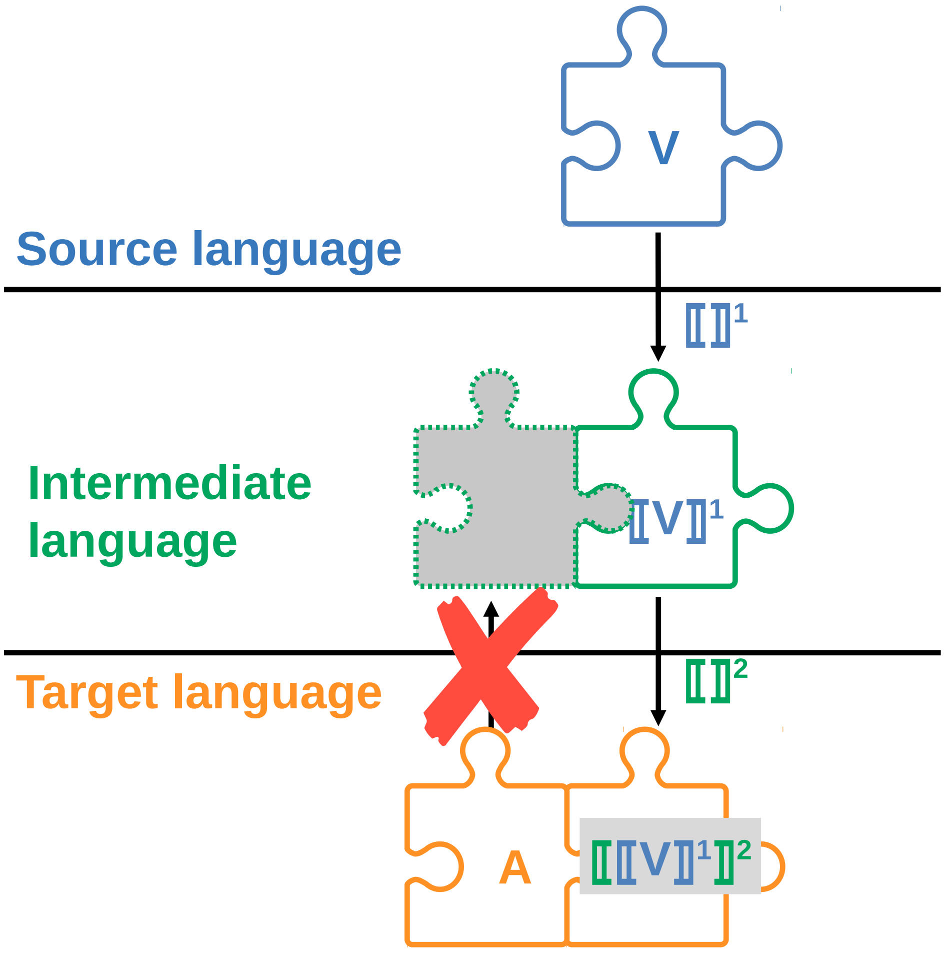


exploits compose
through compiler stages

Compositionality through compiler stages



Compositionality through compiler stages



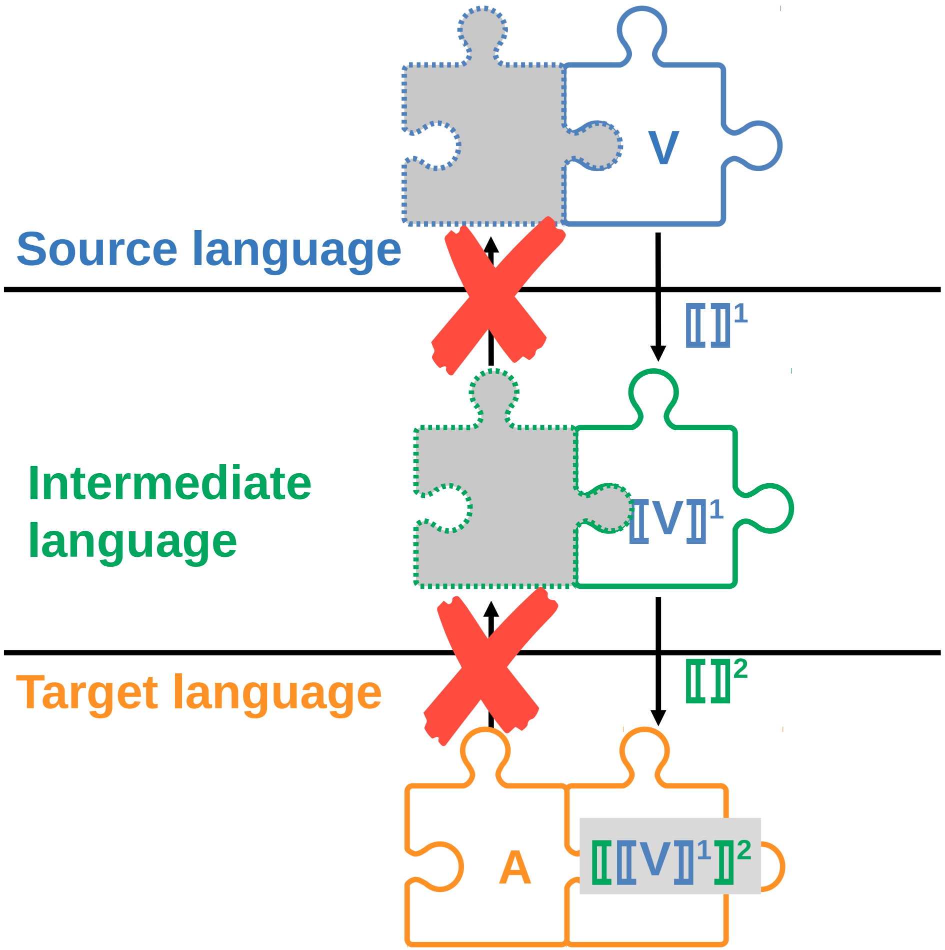
Theorem

If **A** is an exploit of $[\![V]\!]^1$ such that

- $[\![\;]\!]^1$ is correct for **V**; and
- behaviors are invertible,

then **A** is an exploit of **V**.

Compositionality through compiler stages



Theorem

If **A** is an exploit of $[\![V]\!]^1$ such that

- $[\![\;]\!]^1$ is correct for **V**; and
- behaviors are invertible,

then **A** is an exploit of **V**.

Takeaways in the paper...

Takeaways in the paper...

1 "Obvious" applications of secure compilation

- value in formalizing application strategy?

Takeaways in the paper...

1

“Obvious” applications of secure compilation

- value in formalizing application strategy?

2

Non-traditional “programming languages” and “compilers”

- no-op compilers with different operational semantics
- source language as state machines

Takeaways in the paper...

1 "Obvious" applications of secure compilation

- value in formalizing application strategy?

2 Non-traditional "programming languages" and "compilers"

- no-op compilers with different operational semantics
- source language as state machines

3 Trace-relating compilers

- source behaviors different from target behaviors
- behaviors need not be sets of traces

Next steps...

Study counterexamples to secure compilation

- while trying to design a secure compiler
- determine programmability of exploits in design
- given an insecure compiler, help designing mitigations

|galois|

Weird Machines as Insecure Compilation

Jennifer Paykin, Eric Mertens, Mark Tullsen,
Luke Maurer, Benoît Razet, and Scott Moore

PriSC 2020

This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-15-C-0124. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force and DARPA.