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?
<table>
<thead>
<tr>
<th>Definition (Weird Machines)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The computational model made accessible by hacking a particular program.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td><strong>1</strong></td>
</tr>
<tr>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>

(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 $\text{V}$ is a context $\text{A}$ from attack class $\overline{\text{A}}$. 

Source language

Target language

$\text{V}$

$\overline{\text{V}}$

$\text{A}$
Exploits as violations of secure compilation

Definition

An exploit of a source component $V$ is a context $A$ from attack class $\mathcal{A}$ 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 $\mathcal{A}$ such that the behavior of $A[V]$ cannot be simulated by $V$ in the source language.
Hypothesis: Definitions match intuitions
## Framework

### Source language

- **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
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 \( \mathcal{A} \) such that the behavior of \( A[\llbracket V \rrbracket] \) cannot be simulated by \( V \) in the source language.
Secure compilation

Weird machines

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 \( H \subseteq B \):

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

* 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 \textit{robust hyper-property preservation} (RHP) if, \( \forall \) source programs \( \mathcal{V} \) and \( \forall \) hyper-properties \( \mathcal{H} \subseteq \mathcal{B} \):

\[
(\forall \mathcal{C}^S. \text{Behavior}(\mathcal{C}^S[\mathcal{V}]) \in \mathcal{H}) \Rightarrow (\forall \mathcal{C}^T. \text{Behavior}(\mathcal{C}^T[\llbracket \mathcal{V} \rrbracket]) \in \mathcal{H})
\]

Theorem (Abate et al 2019)

A compiler satisfies RHP iff \( \forall \) source programs \( \mathcal{V} \):

\[
\forall \mathcal{C}^T, \exists \mathcal{C}^S. \text{Behavior}(\mathcal{C}^S[\mathcal{V}]) = \text{Behavior}(\mathcal{C}^T[\llbracket \mathcal{V} \rrbracket]).
\]

* approx: behaviors = sets of traces, so \( \mathcal{H} \) is a set of (set of traces)
Robust Property Preservation

∀\(C^T\), ∃\(C^S\). Behavior(\(C^S [V]\)) = Behavior(\(C^T[\llbracket V \rrbracket]\)).
Robust Property Preservation

∀C^T, ∃C^S . Behavior(C^S [V]) = Behavior(C^T[∥V∥]).

Definition

An exploit of a source programs V is a context A ∈ A such that

¬∃ C^S . Behavior(C^S [V]) = Behavior(C^T[∥V∥]).
Robust Property Preservation

**Definition**

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

$$\forall C^s. \text{Behavior}(C^s[V]) \neq \text{Behavior}(C^t[\llbracket V \rrbracket]).$$
Robust Property Preservation

Definition

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

$$\forall C^s. \text{Behavior}(C^s[V]) \neq \text{Behavior}(C^t[\|V\|]).$$

Theorem

$A$ is an exploit of $V$ iff RHP is violated:

$\exists$ hyper-property $H \subseteq B$ such that

$$(\forall C^s. \text{Behavior}(C^s[V]) \in H)$$

but

$$\text{Behavior}(A[\|V\|]) \notin H$$
Secure compilation

Weird machines

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?
A trace exploit of a source program $\mathcal{V}$ is a context $A \in \mathcal{A}$ such that 
\[ \exists t \in \text{Behavior}(A[\mathcal{V}]). \] 
\[ \forall C^s, t \not\in \text{Behavior}(C^s[\mathcal{V}]). \]
Trace Property Preservation

Definition

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

\[ \exists t \in \text{Behavior}(\langle A[V] \rangle). \forall C^S, t \notin \text{Behavior}(C^S[V]) \]

Theorem

- trace exploits $\subseteq$ hyperproperty exploits.
- hyperproperty exploits $\not\subseteq$ trace exploits
- e.g. side-channel attacks
- Trace exploits “more programmable” than hyperproperty exploits.
Secure compilation

Weird machines

exploits compose through compiler stages
Compositionality through compiler stages

Source language

Intermediate language

Target language
Compositionality through compiler stages

Theorem

If \( A \) is an exploit of \([V]^1\) such that \([V]^1\) is correct for \( V \); and behaviors are invertible, then \( A \) is an exploit of \( V \).
Compositionality through compiler stages

Source language

Intermediate language

Target language

Theorem

If \( A \) is an exploit of \([V]_1\) such that \([V]_1\) 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
Weird Machines as Insecure Compilation

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

PriSC 2020