Formalizing Stack Safety as a Security Property

Sean Noble Anderson  Portland State University
Roberto Blanco  Max Planck Institute for Security and Privacy
Leonidas Lampropoulos  University of Maryland, College Park
Benjamin C. Pierce  University of Pennsylvania
Andrew Tolmach  Portland State University
What Needs Protection on the Stack?

- Caller Frame
- Callee Frame
What Needs Protection on the Stack?

Caller Frame

Callee Frame

SP
What Needs Protection on the Stack?

Return Addr

SP
What Needs Protection on the Stack?

Return Addr

SP
What Needs Protection on the Stack?

- Array
- Private Local
- Return Addr
- SP
What Needs Protection on the Stack?

Array

Private Local

Return Addr

SP
What Needs Protection on the Stack?

Array

Private Local

Return Addr

SP

Old Activation Frame
What Needs Protection on the Stack?

- Array
- Return Addr
- Private Local
- Old Activation Frame
- SP
What Needs Protection on the Stack?

- Array
- Pass-by-reference
- Spilled Argument
- Return Addr
- Private Local
- SP
- Old Activation Frame
What Needs Protection on the Stack?

- Array
- Pass-by-reference
- Spilled Argument
- Return Addr
- Private Local
- Old Activation Frame
- SP
What Needs Protection on the Stack?

- Array
- Pass-by-reference
- Return Addr
- Address-taken Local
- Old Activation Frame
- Spilled Argument
- Private Local
- Old Activation Frame
- No sign for Address-taken Local
- No sign for Return Addr
- No sign for Pass-by-reference
- No sign for Array

CSF '23
What Needs Protection on the Stack?

... And registers!

- Array
- Spilled Argument
- Private Local
- Saved Register
- Pass-by-reference
- Return Addr
- Address-taken Local
- Old Activation Frame
Protection Mechanisms

- Shadow Stacks
- Stack Canaries
- Capabilities
- Bounds Checking + hardware
Protection Mechanisms

- Shadow Stacks
- Stack Canaries
- Capabilities
- Bounds Checking
- +hardware
- Tagged Hardware
Protection Mechanisms

- Shadow Stacks
- Stack Canaries
- Capabilities
- Bounds Checking
- +hardware

Tagged Hardware: PIPE
Protection Mechanisms

Tagged Hardware: PIPE
Complex

- Shadow Stacks
- Stack Canaries
- Capabilities
- Bounds Checking
- +hardware
Protection Mechanisms

- Shadow Stacks
- Stack Canaries
- Capabilities
- Bounds Checking
- +hardware

Tagged Hardware: PIPE

- Complex
- Formalizable

Tagged Hardware:
Protection Mechanisms

- Shadow Stacks
- Stack Canaries
- Capabilities
- Bounds Checking

Tagged Hardware: PIPE

- Complex
- Formalizable
- Lazy Enforcement

+hardware
Formal Properties Specifying Stack Safety,
Using Concepts from Theoretical Security
Seeking Specifications

Formal Properties Specifying Stack Safety, Using Concepts from Theoretical Security

\[ P(C,R,t_1,t_2) \] where

\[ C \rightarrow \text{Call} \]

\[ \downarrow t_1 \]

\[ \text{Return} \]

\[ \downarrow t_2 \]

\[ R \]
Formal Properties Specifying Stack Safety, Using Concepts from Theoretical Security

P \((C,R,t_1,t_2)\) where

Or:
\((C,R,t_1,t_2) \mathrel{R} (C',R',t_1',t_2')\)
Contributions

- Separate High-Level Security Notions
- Observation-based Definitions for Lazy Enforcement
- Flexible Model of Language Features
- Randomized Property-based Testing
Contributions

- Separate High-Level Security Notions
- Observation-based Definitions for Lazy Enforcement
- Flexible Model of Language Features
- Randomized Property-based Testing
Contributions

- Separate High-Level Security Notions
- Observation-based Definitions for Lazy Enforcement
- Flexible Model of Language Features
- Randomized Property-based Testing
Contributions

- Separate High-Level Security Notions
- Observation-based Definitions for Lazy Enforcement
- Flexible Model of Language Features
- Randomized Property-based Testing

Options:
- Pass-by-reference
- Callee-save Registers
- Tailcalls

... etc.
Contributions

- Separate High-Level Security Notions
- Observation-based Definitions for Lazy Enforcement
- Flexible Model of Language Features
- Randomized Property-based Testing
∀ C R k . k ∈ sealed(C) → C[k] = R[k]
Caller Integrity

∀ C R k : k ∈ sealed(C) → C[k] = R[k]
Caller Confidentiality = Non-interference

\[
\forall C \quad r_0
\]

\[
C \quad 0 \quad 5 \quad \text{(yellow box)}
\]
∀ \text{C C'} . (k \in \text{sealed}(C) \lor C[k] = C'[k]) \rightarrow
Caller Confidentiality = Non-interference

∀ C C' . C ~ C' →
Caller Confidentiality = Non-interference

\[ \forall C, C', R, R'. C \sim C' \rightarrow C \xrightarrow{t} R \rightarrow C' \xrightarrow{t'} R' \rightarrow \]
Caller Confidentiality = Non-interference

∀ C C' R R'. C ~ C' → C \xrightarrow{t} R \rightarrow C' \xrightarrow{t'} R' → t \approx t'
Caller Confidentiality = Non-interference

∀ C C' R R'. C \sim C' \rightarrow C \xrightarrow{t} * R \rightarrow C' \xrightarrow{t'} * R' \rightarrow t \approx t'
**Caller Confidentiality = Non-interference**

\[ \forall C, C', R, R'. \ C \sim C' \rightarrow C \xrightarrow{t} * R \rightarrow C' \xrightarrow{t'} * R' \rightarrow t \approx t' \]
Caller Confidentiality = Non-interference

∀ C C' R R'. C ~ C' → C \xrightarrow{t} * R → C' \xrightarrow{t'} * R' →

\( t \approx t' \land (\forall k. C[k] \neq R[k] \lor C'[k] \neq R'[k] \rightarrow R[k] = R'[k]) \)
Caller Confidentiality = Non-interference

∀ C C' R R'. C ~ C' → C \xrightarrow{t} * R → C' \xrightarrow{t'} * R' →$

\begin{align*}
t & \approx t' \land (\forall k. \text{consistent}(C, C', R, R', k))
\end{align*}
Programmable Interlocks for Policy Enforcement: PIPE

SP

PC

inst
Programmable Interlocks for Policy Enforcement: PIPE

inst → Micro-policy

fail
Lazy Depth Isolation

= unclaimed  = depth 0  = depth 1  = return address

Roessler and DeHon, 2018
Protecting the Stack
with Metadata Policies
and Tagged Hardware
Lazy Depth Isolation

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

= unclaimed  = depth 0  = depth 1  = return address

store

Micro-policy

PC  loc

SP  loc

...
Lazy Depth Isolation

= unclaimed  = depth 0  = depth 1  = return address

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

Micro-policy

store

loc

PC

Micro-policy

loc

PC

10 / 17
Lazy Depth Isolation

\[ \text{= unclaimed} \quad \text{= depth 0} \quad \text{= depth 1} \quad \text{= return address} \]

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

Micro-policy

store

location

location

PC

SP

PC
Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

Lazy Depth Isolation

= unclaimed  = depth 0  = depth 1  = return address

Micro-policy

store

loc

PC

loc

PC
Lazy Depth Isolation

= unclaimed  = depth 0  = depth 1  = return address

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

Micro-policy

store

PC

loc

loc
Lazy Depth Isolation

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

Micro-policy

store
loc
PC

Micro-policy

loc
PC

= unclaimed
= depth 0
= depth 1
= return address
Lazy Depth Isolation

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

= unclaimed  = depth 0  = depth 1  = return address

Micro-policy

store

loc

PC

loc

PC

...
Lazy Depth Isolation

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware

= unclaimed  = depth 0  = depth 1  = return address

Micro-policy

store  loc  PC  loc  PC
Lazy Depth Isolation

\[\text{store} \quad \text{loc} \quad \text{Micro-policy} \quad \text{load} \quad \text{loc} \quad \text{Micro-policy} \quad \text{fail}\]

\[\begin{array}{c}
\text{unclaimed} \quad \text{depth 0} \quad \text{depth 1} \quad \text{return address}
\end{array}\]

Roessler and DeHon, 2018
Protecting the Stack with Metadata Policies and Tagged Hardware
Properties, Observationally

\[ 5 \neq 42 \]

Integrity Violation*

Confidentiality Violation*
Properties, Observationally

* if the location is *relevant*
Properties, Observationally

* if the location is relevant

Confidentiality Violation*

Integrity Violation*

$5 \neq 42$

$C \overset{r_0}{\rightarrow} C'$

$R \overset{1}{\rightarrow} R'$

$t \approx t'$

$t \approx t'$

$t_2 \neq t_2'$

$t_2 \neq t_2'$
Properties, Observationally

\[ \text{irrelevant}(R, K) \triangleq \forall R'. (\forall k \notin K . C[k] = C'[k]) \to \]

\[ R \xrightarrow{t} * \perp \to R' \xrightarrow{t'} * \perp \to \]

\[ t \approx t' \]

\text{Caller Integrity} \triangleq \forall C R k . k : \text{sealed}(C) \to \text{irrelevant}(R, \{ k | C[k] \neq R[k] \})

\text{Caller Confidentiality} \triangleq \forall C C' R R'. C \sim C' \to C \xrightarrow{t} * R \xrightarrow{t'}* R' \to \]

\[ t \approx t' \land \text{irrelevant}(R, \{ k | \neg \text{consistent}(C, C', R, R', k) \}) \]}
Language Features via Security Semantics

- Security-relevant Operations
- Security Context
- Hyperproperties

Steps:
- Update
- Inform
Language Features via Security Semantics

Security-relevant Operations → Update → Security Context → Inform → Hyperproperties

- Call
- Tailcall
- Return
- Alloc
- Dealloc
Language Features via Security Semantics

Security-relevant Operations

- Update

Security Context

- Inform

Hyperproperties

Abstract stack mapping memory, registers to:
- Public
- Free
- Active
- Sealed

- Call
- Tailcall
- Return
- Alloc
- Dealloc
Language Features via Security Semantics

- **Security-relevant Operations**
  - Call
  - Tailcall
  - Return
  - Alloc
  - Dealloc

- **Update**

- **Security Context**
  - WBCF
    - Integrity
    - Confidentiality
  - Abstract stack mapping memory, registers to:
    - Public
    - Free
    - Active
    - Sealed

- **Inform**

- **Hyperproperties**

Given security context at call:
Language Features

- Callee-save Registers
- Pass-by-reference
- "Public" (address-taken) variables
- Arguments Spilled to Memory
- Tailcalls
- Memory Shared by Capability

Ongoing Work:
- Exceptions
- Concurrency
Property-based Testing w/ QuickChick

Random Program +

(Eager) Depth Isolation
Depth Isolation w/ Lazy Tagging
“Activation Isolation” w/ Lazy Tagging

Integrity Test
Conf. Test

X/?
X/?
Property-based Testing w/ QuickChick

Random Program + (Eager) Depth Isolation

Depth Isolation w/ Lazy Tagging

“Activation Isolation” w/ Lazy Tagging

Integrity Test

Conf. Test

✗ /

✗ ?

✗ ?
Property-based Testing w/ QuickChick

Random Program + {
  (Eager) Depth Isolation
  Depth Isolation w/ Lazy Tagging
  “Activation Isolation” w/ Lazy Tagging
  Injected Bugs
}

Integrity Test
Conf. Test

X/?  X/?
Future Work

- Testing Other Mechanisms
  - Cerise, CHERI-based calling convention
  - Software bounds checking
- Expanding model
  - Exceptions
  - Concurrency
- Mechanized Proofs
Summary

- Theory of Stack Safety: Integrity, Confidentiality, and/or WBCF
- Security Semantics: useful factorization over language features
- Observation-based properties to describe sophisticated enforcement mechanism
- Testing identifies flawed micro-policies