A Theory of Platform-Dependent Low-Level Software

Marius Nita
Dan Grossman
Craig Chambers

University of Washington WASP Group
wasp.cs.washington.edu

The memory-safety of a C program often depends on assumptions that hold for some but not all compilers and machines.

Example

```
struct S { void *buf; int    len; };
struct D { void *buf; size_t len; };
...
struct S ss[100];
...
struct D* ds = (struct D*)ss;
...
// treat ds[N].len as the length of ds[N].buf
```

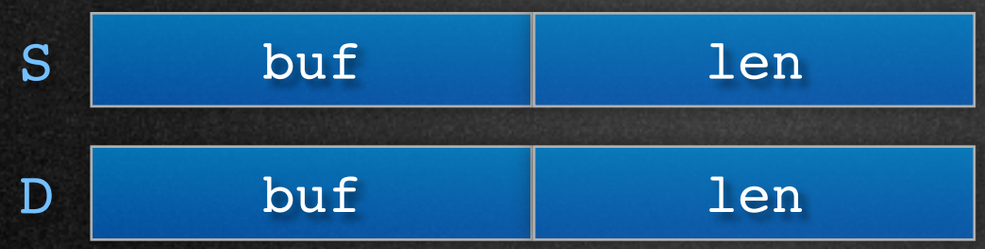
Example

```
struct S { void *buf; int    len; };
struct D { void *buf; size_t len; };
...
struct S ss[100];
...
struct D* ds = (struct D*)ss;
...
// treat ds[N].len as the length of ds[N].buf
```

Example

```
struct S { void *buf; int    len; };
struct D { void *buf; size_t len; };
...
struct S ss[100];
...
struct D* ds = (struct D*)ss;
...
// treat ds[N].len as the length of ds[N].buf
```

32-bit:

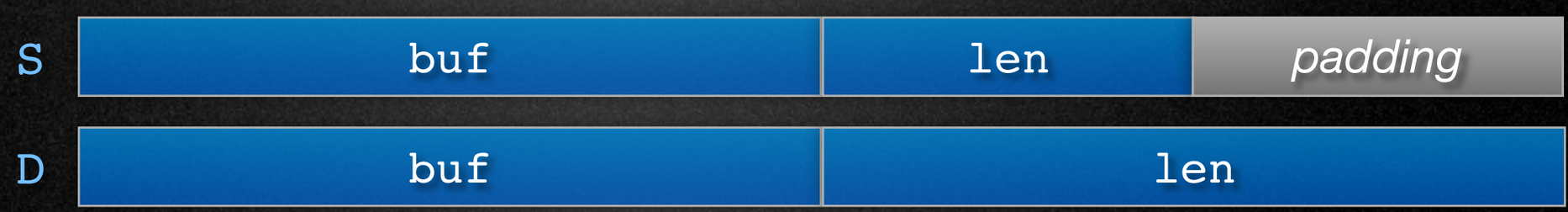


Example

```
struct S { void *buf; int    len; };
struct D { void *buf; size_t len; };
...
struct S ss[100];
...
struct D* ds = (struct D*)ss;
...
// treat ds[N].len as the length of ds[N].buf
```

32-bit:

LP-64:



Porting is Hard

- Low-level programs make platform-dependent assumptions:
  - How structs are padded.
  - Sizes of types.
  - Alignment restrictions of the underlying hardware.

Porting is Hard

- Low-level programs make platform-dependent assumptions:
  - How structs are padded.
  - Sizes of types.
  - Alignment restrictions of the underlying hardware.
- Porting to new platforms is hard:
  - Must identify which code needs to change.

Tool Support is Weak

- Lint-like technology.
- Grep.
- Compiler flags:
  - -Wpadded
  - -Wcast-align

POPL + C = ?

wasp.cs.washington.edu

POPL + C = ?

- Improving C (e.g., *CCured*, *Cyclone*, *Deputy*, *SAFECode*):
  - Assume a particular platform or
  - Make the same assumptions as the underlying C compiler.

POPL + C = ?

- Improving C (e.g., *CCured*, *Cyclone*, *Deputy*, *SAFECode*):
  - Assume a particular platform or
  - Make the same assumptions as the underlying C compiler.
- Formal semantics for C-like languages (e.g., *Leroy*, *Norrish*):
  - Omit platform-dependent operators or
  - Model platform-dependent steps as nondeterminism.

This Work

- Semantics for a C-like language:
  - Explicit notion of platform and platform-dependent steps.
  - Memory-safety is platform-dependent.

This Work

- Semantics for a C-like language:
  - Explicit notion of platform and platform-dependent steps.
  - Memory-safety is platform-dependent.
- A bug-finding tool:
  - Discovers a class of portability bugs in C programs statically.
  - Does not need physical access to target platforms.

wasp.cs.washington.edu

Formal Semantics

wasp.cs.washington.edu

Overview

- Explicit notion of **platform**.
- A platform plays two roles:
    - Parameter to the operational semantics.
    - Something that can be described with a **layout portability constraint**.

Overview

- Explicit notion of **platform**.
- A platform plays two roles:
  - Parameter to the operational semantics.
  - Something that can be described with a **layout portability constraint**.

- Given a program ***e***:
  - We extract a constraint ***S*** from ***e***.
  - ***e*** is memory-safe on all platforms **Π** described by ***S***.

Ingredients

- Operational semantics parameterized by platforms.

$$\Pi \vdash e \rightarrow e'$$

Ingredients

- Operational semantics parameterized by platforms.

$$\Pi \vdash e \rightarrow e'$$

- Constraints and constraint checking.

$$\Pi \models S$$

Ingredients

- Operational semantics parameterized by platforms.

$$\Pi \vdash e \rightarrow e'$$

- Constraints and constraint checking.

$$\Pi \models S$$

- Constraint extraction: type-and-effect system.

$$\Gamma \vdash e : \tau \; ; \; S$$

wasp.cs.washington.edu

Key Theorem

If $\Gamma \vdash e:\tau \; ; \; S$ and $\Pi \models S$

and $\Pi \vdash e \rightarrow^* e'$, then e' is not stuck on Π.

Core Language

- C-like language with many relevant features, including:
  - struct types
  - pointer casts: `(T*)e`
  - address-of-field operator: `&e→f`

Operational Semantics

- Byte-level memory model.
  - `*e` steps to a sequence of ***n*** bytes.
    - ***n*** = size of `e`'s type.
- As in C, pointer casts are unchecked.
  - `(T*)e` steps to `e`.

Parameter to the Operational Semantics

`*e` → *???*

- How many bytes do we fetch?

Parameter to the Operational Semantics

`*e` → *???*

- How many bytes do we fetch?
  - Depends on the size of `e`'s type.
  - The size of `e`'s type is platform-dependent.

Parameter to the Operational Semantics

`*e` → *???*

- How many bytes do we fetch?
- Depends on the size of `e`'s type.
- The size of `e`'s type is platform-dependent.

$$\Pi \vdash e \rightarrow e'$$

Layout Constraints

$$\mathbf{S} \quad = \quad \text{access}(4,8) \wedge \text{sizeof}(\texttt{long}) = 8$$

Layout Constraints

$$\mathbf{S} \quad = \quad \text{access}(4,8) \wedge \text{sizeof}(\texttt{long}) = 8$$

- True on platforms on which
  - 8-byte accesses at 4-byte alignments are allowed
  - type long is 8 bytes

Layout Constraints

$$\mathbf{S} \quad = \quad \text{access}(4,8) \wedge \text{sizeof}(\texttt{long}) = 8$$

- True on platforms on which
  - 8-byte accesses at 4-byte alignments are allowed
  - type long is 8 bytes

$$\Pi \models S$$

Extracting Constraints

- Type-and-effect system:

$$\Gamma \vdash e : \tau \; ; \; S$$

Extracting Constraints

- Type-and-effect system:

$$\Gamma \vdash e : \tau \; ; \; S$$

- Most rules are standard:

$$\frac{\Gamma \vdash e_1 : \tau' \; ; \; S_1 \qquad \Gamma \vdash e_2 : \tau \; ; \; S_2}{\Gamma \vdash e_1 ; e_2 : \tau \; ; \; S_1 \wedge S_2}$$

Pointer Cast Constraint

$$\frac{\Gamma \vdash e : \tau_{src}^{*} \; ; \; S}{\Gamma \vdash (\tau_{dest}^{*})e : \tau_{dest}^{*} \; ; \; S \wedge \mathit{subtype}(\; \mathit{layout}(\tau_{src})^{*}, \mathit{layout}(\tau_{dest})^{*} \;)}$$

Pointer Cast Constraint

$$\frac{\Gamma \vdash e : \tau_{src}^{*} \; ; \; S}{\Gamma \vdash (\tau_{dest}^{*})e : \tau_{dest}^{*} \; ; \; S \wedge \mathit{subtype}(\; \mathit{layout}(\tau_{src})^{*}, \mathit{layout}(\tau_{dest})^{*} \;)}$$

- *layout*: a type's in-memory layout
- *subtype*: subtyping of memory blocks

Checking Constraints

$$\Pi \vDash \mathit{subtype}(\ \mathit{layout}(\tau_{src})^{*}, \mathit{layout}(\tau_{dest})^{*}\ )$$

Checking Constraints

$$\Pi \vDash subtype(\ layout(\tau_{src})^*, layout(\tau_{dest})^*\ )$$

if and only if

$$\Pi.layout(\tau_{src})^*$$

is a **physical subtype** of

$$\Pi.layout(\tau_{dest})^*$$

Checking Constraints

$$\Pi \vDash \mathit{subtype}(\ \mathit{layout}(\tau_{src})^{*}, \mathit{layout}(\tau_{dest})^{*}\ )$$

if and only if

$$\Pi.\mathit{layout}(\tau_{src})^{*}$$

is a **physical subtype** of

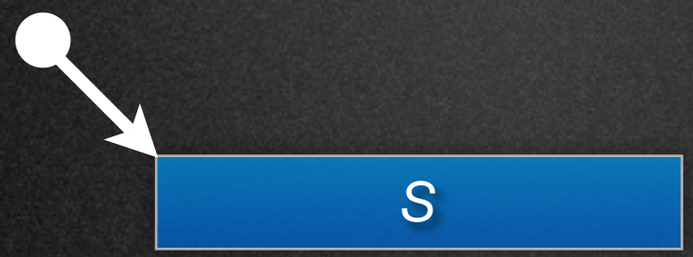
$$\Pi.\mathit{layout}(\tau_{dest})^{*}$$

Chandra/Reps
Condit *et al.*

Physical Subtyping

Drop the suffix under a pointer:

$\leq$



Physical Subtyping

Can always subsume to pad bytes:

$\leq$

Key Theorem

If $\Gamma \vdash e:\tau \; ; S$ and $\Pi \models S$

and $\Pi \vdash e \rightarrow^* e'$, then e' is not stuck on Π.

Key Theorem

If $\Gamma \vdash e:\tau ; S$ and $\Pi \models S$

and $\Pi \vdash e \rightarrow^* e'$, then e' is not stuck on Π.

- Proof employs a platform-dependent type system: $\Pi;\Gamma \vdash e:\tau$.

Key Theorem

If $\Gamma \vdash e:\tau ; S$ and $\Pi \models S$

and $\Pi \vdash e \rightarrow^* e'$, then e' is not stuck on Π.

- Proof employs a platform-dependent type system: $\Pi;\Gamma \vdash e:\tau$.
- **Theorem**: If $\Pi;\Gamma \vdash e:\tau$ and $\Pi \vdash e \rightarrow^* e'$, then e' is not stuck on Π.

Key Theorem

If $\Gamma \vdash e:\tau \; ; \; S$ and $\Pi \models S$

and $\Pi \vdash e \rightarrow^* e'$, then e' is not stuck on Π.

- Proof employs a platform-dependent type system: $\Pi;\Gamma \vdash e:\tau$.
- **Theorem**: If $\Pi;\Gamma \vdash e:\tau$ and $\Pi \vdash e \rightarrow^* e'$, then e' is not stuck on Π.
- **Theorem**: If $\Gamma \vdash e:\tau \; ; \; S$ and $\Pi \models S$, then $\Pi;\Gamma \vdash e:\tau$.

Bug-Finding Tool

wasp.cs.washington.edu

Bug-Finding Tool

- Tool that finds unportable pointer casts in C programs.

wasp.cs.washington.edu

Bug-Finding Tool

- Tool that finds unportable pointer casts in C programs.
- Uses a may-alias analysis to approximate run-time types.

wasp.cs.washington.edu

Bug-Finding Tool

- Tool that finds unportable pointer casts in C programs.
- Uses a may-alias analysis to approximate run-time types.
- Host/target setup:
  - Host: platform on which the program is known to run well.
  - Target: platform to which the program is to be ported.

Bug-Finding Tool

- Tool that finds unportable pointer casts in C programs.
- Uses a may-alias analysis to approximate run-time types.
- Host/target setup:
  - Host: platform on which the program is known to run well.
  - Target: platform to which the program is to be ported.
  - Warn only when a cast works on the host but not on the target.

Bug-Finding Tool

- Tool that finds unportable pointer casts in C programs.
- Uses a may-alias analysis to approximate run-time types.
- Host/target setup:
  - Host: platform on which the program is known to run well.
  - Target: platform to which the program is to be ported.
  - Warn only when a cast works on the host but not on the target.
- “Platforms” are defined by us procedurally.
  - 30-50 lines of OCaml per platform definition.

wasp.cs.washington.edu

Case Study

- Ran the tool on Spread:
  - Messaging bus for use by distributed applications.

Case Study

- Ran the tool on Spread:
  - Messaging bus for use by distributed applications.
- Found two bugs: one previously unreported.
  - Host: Standard 32-bit/gcc/X86 platform.
  - Target: gcc/LP-64 platform.
  - Zero false positives reported.

Case Study: Tool Output

```
struct scat_element { void *buf; int    len; };
struct iovec        { void *buf; size_t len; };
```

Case Study: Tool Output

```
struct scat_element { void *buf; int    len; };
struct iovec        { void *buf; size_t len; };
```

Case Study: Tool Output

```
struct scat_element { void *buf; int    len; };
struct iovec        { void *buf; size_t len; };
```

```
data_link.c:196: scat_element * ==> iovec *

    Host (Gcc/32-bit/X86):
        Src:  ptr(ptr(b) bbbb)
        Dest: ptr(ptr(b) bbbb)

    Target (Gcc/LP-64):
        Src:  ptr(ptr(b) bbbb----)
        Dest: ptr(ptr(b) bbbbbbbb)
```

Conclusion

- Semantics for a C-like language with unspecified type layout.
- Analysis that finds platform dependencies.
- Some uses:
  - Porting tools.
  - Documentation.
  - Specifications for ***safe*** languages with (partially) unspecified features.

wasp.cs.washington.edu

More in the Paper

- Alignments.
- Complete description of platforms and the constraint language.
- Extensions: arrays, recursive types, conditional execution.
- Detailed tool discussion.

wasp.cs.washington.edu

Thank You!

wasp.cs.washington.edu