Software Security

‘Safe’ programming languages

Erik Poll

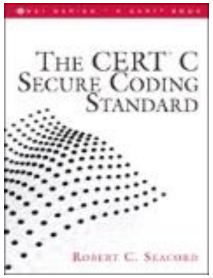
Radboud Universiteit Nijmegen

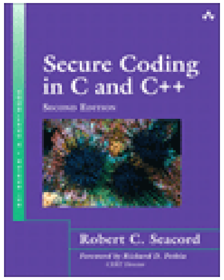
Producing more secure code

1. You can try to produce more secure C(++) code.

   Not just using SAST & DAST tools, eg PREfast & fuzzing,
   but also by reading – or making other people read – books like

   CERT secure coding guidelines for C and C++
   at http://www.securecoding.cert.org





2. More structural way to improve security:
   improve the programming language

   – not just to prevent memory corruptions flaws,
     but other common problems too…

Safe(r) programming languages

You can write insecure programs in ANY programming language

Still…some languages are safeR than others by

to ruling out certain classes of bugs

making them less likely

make them easier to spot

or mitigating their impact

Language-based security

Security features & guarantees provided by programming language

- **Safety guarantees, incl.**

  memory **safety**
  initialisation **safety**
  type **safety**
  thread **safety**

  Many *kinds* & *levels*

- **Access control & modularisation**

  Eg. visibility/access restrictions with eg. `public, private`

Some features are interdependent, eg

- **type safety & just about anything else relies on memory safety**
- **memory safety relies on type safety**

Other ways the programming language can help

A programming language can also help security by

- offering good APIs/libraries, eg.
  - APIs with parametrised queries/prepared statements for SQL
  - more secure string libraries for C
- offering convenient language features
  - esp. exceptions, to simplify handling error conditions
- making assurance of the security easier, by
  - being able to understand code in a modular way
  - only having to review the public interface, in a code review

General idea behind safety

Under which conditions does

```
a[i] = (byte)b
```

make sense?

`a` must be a non-null byte array;
`i` should be a non-negative integer less then array length;
`b` should be (castable to?) a byte

Two approaches

1. the programmer is responsible for ensuring these conditions
   "unsafe" approach
2. the language is responsible for checking this
   "safe" approach

Heated debates about pros & cons highlight tension between

flexibility, speed and control vs safety & security

But execution speed ≠ speed of development of secure code

and programmer's time may be more expensive than CPU cycles

Safe programming languages

Safe programming languages

- impose some discipline or restrictions on the programmer
- offer some abstractions to the programmer,
  with associated guarantees

This takes away some freedom & flexibility from the programmer, but hopefully extra safety and easier understanding makes it worth this.

Attempts at a general definition of safety

A programming language can be considered *safe* if

1. **You can trust the abstractions provided by the programming language**

   The programming language enforces these abstractions and guarantees that they cannot be broken

   - **Eg programmer does not need to know how `Strings` are represented**

2. **Programs have a precise & well defined *semantics* (ie. *meaning*)**
   - **More generally, leaving things UNDEFINED in any specification is asking for security trouble**

3. **You can understand the behaviour of programs in a modular way**

'safer' & 'unsafer' languages

Scala

machine code Java Kotlin Haskell

C MISRA-C C#

ML

C++ OCaml

Rust

Python

Prolog

more 'unsafe' 'safe' even more 'safe’

This is overly simplistic; there are many dimensions of safety

Functional languages such as Haskell are safe because data is immutable (no side-effects) which also makes them thread-safe

Dimensions & levels of safety

There are many dimensions of safety

memory-safety, type-safety, thread-safety, arithmetic safety; guarantees about (non)nullness, about immutability, about the absence of aliasing,...

For each dimension, there can be many levels of safety

Eg, in increasing level of safety, going outside array bounds may:

1. let an attacker inject arbitrary code
2. *possibly* crash the program (or else corrupt some data)
3. *definitely* crash the program
4. throw an exception, which the program can catch to handle the issue gracefully
5. be ruled out at compile-time

'unsafe'; some undefined semactics

'safe'

Safety: how?

Mechanisms to provide safety include

- compile time checks, eg type checking
- runtime checks, eg array bounds checks, checks for nullness, runtime type checks, ...
  - for things that cannot be guaranteed by compile time checks
- automated memory management

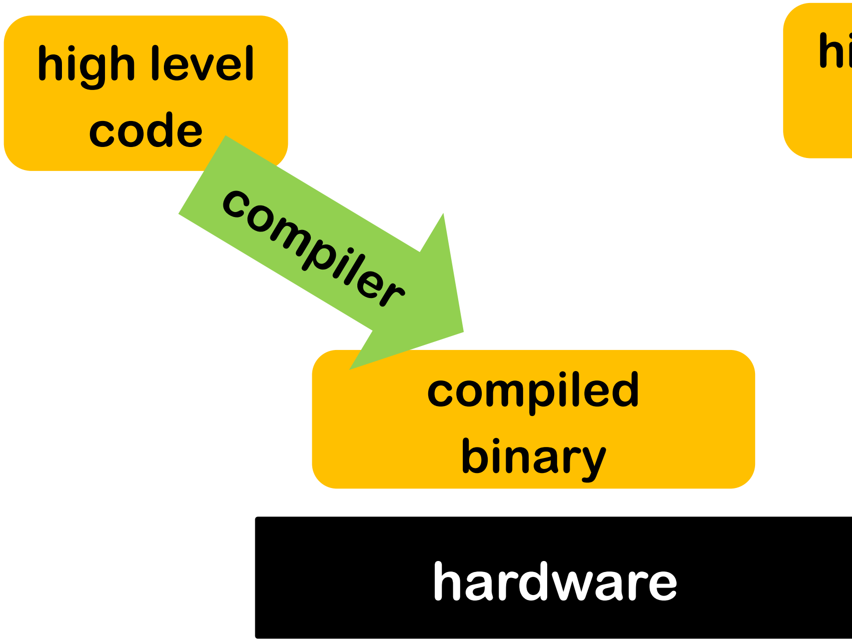
  so programmer does not have to `free()` heap-allocated data

Safe programming languages often use an execution engine (aka interpeter aka runtime aka platform) do the things above

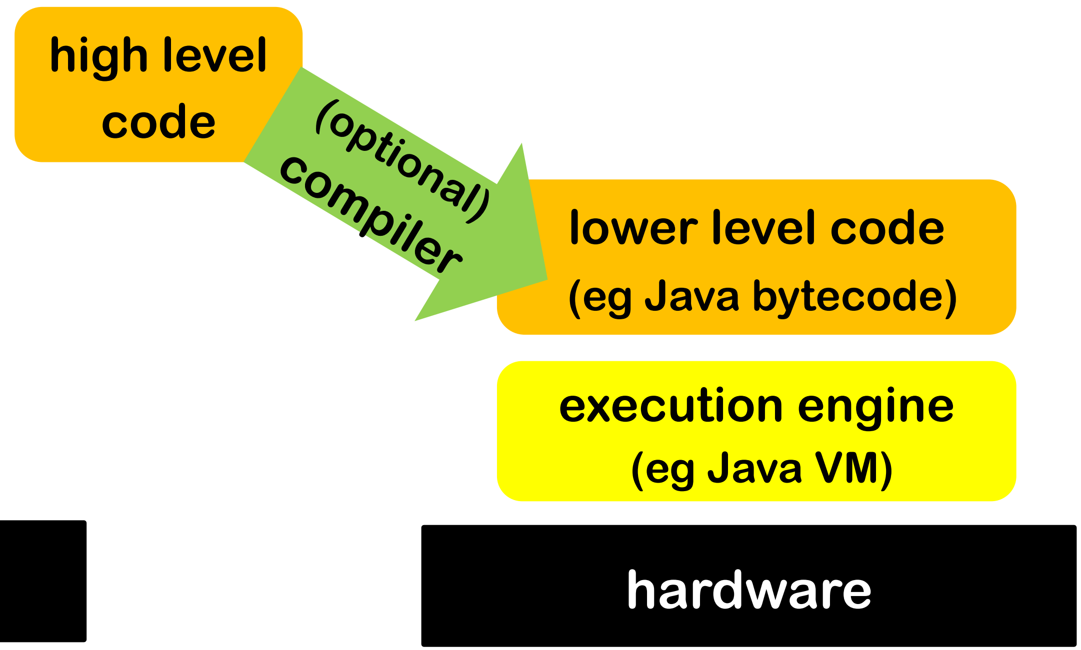
Eg Java Virtual Machine (VM) or Java Runtime Environement (JCRE)

Compiled vs interpreted code

Compiled binary runs on bare hardware



Execution engine (aka 'runtime') isolates code from hardware



Any defensive measures have to be compiled into the code.

The programming language (platform) still 'exists' at runtime and execution engine can do checks at runtime

Memory-safety


America's Cyber Defense Agency
NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE
Search
Topics
Spotlight
Resources & Tools
News & Events
Careers
About
Home / Resources & Tools / Resources
SHARE:
The Case for Memory Safe Roadmaps
Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously
Publish Date: December 06, 2023
RELATED TOPICS: CYBERSECURITY BEST PR


America's Cyber Defense Agency
NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE
Search
Topics
Spotlight
Resources & Tools
News & Events
Careers
About
Home / Resources & Tools / Resources
SHARE:
Exploring Memory Safety in Critical Open Source Projects
Publish Date: June 26, 2024

Rust

Real momentum building behind Rust

as the memory-safe language for low-level 'systems' software

https://rustnl.org


RustWeek 2026
May 18-23 — Utrecht, NL
The world's biggest Rust conference includes one day of workshops, two days of talks, a hackathon, social activities, and more!
Tickets
Week schedule
RustNL

Memory-safety – with/without initalisation safety

There are two flavours of memory safety:

A programming language is memory-safe if it guarantees that

1. programs never access unallocated or de-allocated memory
   - hence also: no segmentation faults at runtime
2. possibly also initialization safety
   program never access *uninitialised* memory

Memory safety

Unsafe language features that break memory safety

1. **not having array bounds checks**
2. **allowing pointer arithmetic**
3. **null pointers, *but only if these cause undefined behaviour***

Null pointers in C

What happens if you dereference a NULL pointer in C?

Common (and incorrect!) folklore:

dereferencing a NULL pointer will crash the program.

But, the C standard only guarantees

the result of dereferencing a null pointer is undefined.

So it *may* crash the program, but ANYTHING ELSE *might happen*

See the CERT Secure Coding guidelines for C
https://www.securecoding.cert.org/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointer
for discussion of a security vulnerability in a PNG library caused by a null dereference that didn't crash (on ARM processors)

Memory safety

Unsafe language features that break memory safety

1. **no array bounds checks**
2. **pointer arithmetic**
3. **null pointers**, ***but only if these cause undefined behaviour***
4. **manual memory management with eg. malloc, new & free**

Manual memory management can be avoided by

a) **not using the heap at all (eg in MISRA C)**

b) **automating memory management**