Previously

Stages of code injection

1. Inject code

2. Hijack control flow

But step 1 is getting harder!

Code reuse attacks

~~0. Inject code~~

1. Hijack control flow

How do we stop the hijacking?

Stopping hijacking

Stack protection

CFI: control flow integrity

Full *memory safety*

... which we'll discuss next time ... which is now

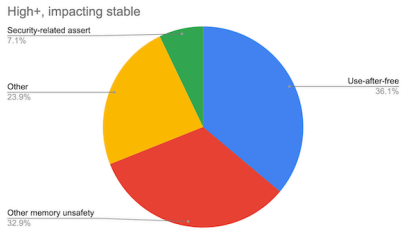
Memory safety

Two categories:

- spatial memory safety
- temporal memory safety

How to achieve?

- write perfect software!
- *memory-safe* languages



Source: Chromium project

_____ memory safety refers to an inability for code to write outside of defined boundaries. For example, modifications to an array should not be able to cause changes outside of that array. Modifications to an object should not be able to cause changes outside of that object.

A related concept is _____ memory safety: an inability of code to access memory _____. For example, some code could be given a pointer to a heap-allocated object; we would like to know that this code will only be able to modify that memory as long as _____. This is also why friends don't let friends return _____ from functions: that pointer *used* to point at a local variable, but now it points at some arbitrary chunk of stack memory that could be used for anything.

Writing perfect software is... not a realistic plan. People make mistakes, so we had better build systems that can accommodate the occasional human error!

> *Counterpoint: if one person's brief lapse in judgement can bring down the whole org, we're building our systems all wrong.*
>
> > *We need to make online security a mandatory subject in our schools. It's not just about protection of personal devices and data, but one person's brief lapse in judgement can bring down a school, a payroll system, or a hospital. 2/2*
> > *— Kimler for SC (@kimlerforsc.bsky.social) May 12, 2024 at 5:56 PM*
>
> *— @trombonehero.bsky.social*

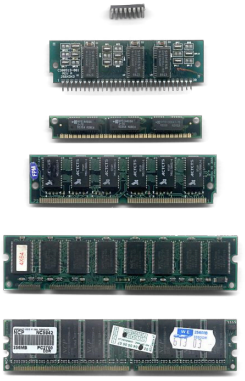
Memory-safe languages, although excellent, are only a _____ to the general problem. We'll talk about why at the end of this lecture.

Program execution

Q: how do we load a value from memory?

A: it depends on the language!

- compiled
- interpreted
- bytecode-interpreted



Different languages provide for different modes of memory access.

How do we categorize languages?

- programing **paradigm** (OO, functional, etc.)
- memory management (manual vs **garbage-collected**)
- **compiled** vs **interpreted**

Compiled languages

Examples?

Where are memory access decisions made?

Examples of languages that compile to machine instructions: _____, _____, _____, _____, _____, _____...

The _____ may prevent certain kinds of accesses at compile time. For example, some code is supposed to be able to access _____ but other code isn't (see example: private.cpp). However, at runtime, all we have are _____ that _____ and _____ values.

Interpreted languages

Examples?

Where are memory access decisions made?

Examples of languages that are *at least primarily* intepreted (they may use _____ _____ or even _____ (AOT) compilation as an implementation detail) include _____, _____, _____, _____ and, of course, _____.

In such languages, other people's code doesn't get compiled directly to native machine instructions, it is _____. An interpreted language has an _____ that can make additional decisions about how (or whether!) to honour a request made by an interpreted statement or expression.

For example, in private.js, the code outside of the `f` function has no way to inspect the low-level memory details of the object returned from `f`. The question of whether or not to allow an access doesn't depend on _____, it depends on the _____.

Bytecode-interpreted languages

What's different?

Why?

A bytecode-interpreted language (e.g., anything that runs on the _____) includes a _____ for its bytecode. Instead of interpreting Java or Scala, those languages can be compiled to the Java bytecode format, which is executed by a lower-level _____. This is also true for _____: you can compile languages like _____, _____, _____ and _____ (see: https://github.com/appcypher/awesome-wasm-langs) into _____ and then execute the result in any Web browser with much greater speed than interpreting from source.

In a bytecode-interpreted language, we get some of the benefits of compilation, e.g., we don't have to parse a bunch of program text every time we run the program. We *also* get some of the benefits of an interpreter, such as _____! That means we can't, for example, walk off the end of an array.

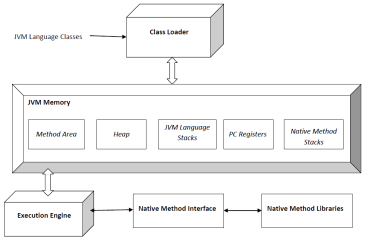
Example: Java

Memory management

Memory access

Bytecode and TCBs

`SecurityManager`



Li Gong *et al.*, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2", in *USITS '97: Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

A Java program, like any other program, runs in a _____ that has a _____ _____. One key difference from compiled programs, however, is that the user code is never exposed to those _____. It's kind of like a _____ _____ of real physical memory. Instead of pointers, Java programs see _____, and unlike pointers, _____.

In such a bytecode-interpreted language, all memory accesses have to go through _____ _____.

However, there is no such thing as a free lunch. One of the costs of using any sort of interpreter is that the interpreter becomes _____... and thus we tend to have a _____ _____!

Java, in particular, also has interesting facilities for disabling features like reflection, which by design circumvent the normal type rules of the language. A `SecurityManager` running on the JVM will also allow you to control access to external resources like files and network sockets. You can even attach privileges like "can access this external URL" to specific pieces of code based on the code's identity... but more about that later when we get to the lecture on Code Signing.

So... perfection?

Write all software in a memory-safe language?

TCB considerations

Memory safety in compiled languages

1. Compiler-added run-time safety checks
2. Limited unsafety
3. Continued dangers of native instructions

High-level language interpreters have to be written in something. You might be able to write a lot of a Java interpreter in Java, but at the lowest levels you will find _____, sometimes _____.

Languages like _____ and _____ claim to provide memory safety, but they are compiled languages. How is this possible?

The compiler can add extra code to check some accesses at run time. For example, if you are indexing within an array, the compiler can implicitly add code such as `if 0 <= i < n`.

Languages that aspire to "systems programming" (i.e., things that have to be aware of or manipulate the lowest-level primitives such as hardware registers) have to allow for unsafe operations. There is no memory-safe way to perform arbitrary register, memory or I/O operations, so these kinds of languages have to provide some way to break abstraction layers. C code can include assembly via the `asm` keyword. Rust code can explictly violate memory safety guarantees if it uses the `unsafe` keyword.

Even with those checks, however, if you load someone else's native instructions and execute them, _____!

Safe compiled code?

What is a language?

Software

AddressSanitizer, CCured, Cyclone, "fat pointers", Go, Rust, ...

Hardware:

Arm MTE, CHERI, Hardbound, MPX, segmentation, Watchdog, ...

When we think of a language, we typically think about _____ and the _____ for writing it. However, in addition to _____, we also have _____ that are defined by language specifications and — crucially — _____ _____. If we take this expanded view of what makes a language, we can see a number of approaches applied in various places that can be used to improve the security of compiled code, too.

Software

AddressSanitizer (and other "sanitizers" like Thread Sanitizer and the Undefined Behaviour Sanitizer) can help spot memory errors during testing that might otherwise have gone unnoticed. CCured is an example of an approach that uses static analysis to figure out how pointers in a C program are "meant" to be used and dynamic analysis to ensure that they are, in fact, used that way. Cyclone is a C dialect with better memory safety properties than vanilla C, which it is designed to be compatible with (or at least easy to adapt from). Newer languages like Go and Rust have more expressive type systems that make it possible to write memory-safe code even in high-performance compiled languages with limited run-time checking.

Hardware

Arm MTE has been adopted by Android to detect memory safety violations at run time. Hardbound, MPX and Watchdog attempt to provide various forms of hardware memory safety enforcement. CHERI is a designed-for-security instruction set extension for ARM and MIPS that is just about to ship its first hardware prototypes; it has the potential to change _____ by allowing high-level object accesses to be precisely enforced by hardware.

Summary

Memory safety

Memory-safe language concepts

Safe unsafe languages?