
Language-Based Security
SAFE PROGRAMMING LANGUAGES

Reading

- Chapter 3 of Erik Pool Notes available on TEAMS

Producing more secure code

- **One can try to produce more secure C(++) code:**
  - by reading and making other people read CERT secure coding guidelines for C and C++ at **http://www.securecoding.cert.org**
  - **Secure programming for Linux and UNIX HOWTO, D. Wheeler**
  - **Awareness**
- More structural way to improve security:
  - **Improve the programming language toolchain**
    - Compiler (**CFI**), run-time support (**Stack Canaries**). …
  - **improve the programming language primitives**
    - not just to prevent memory corruptions flaws, but maybe other common problems too…

Programming Language Design
Language Based Security

- **Security features & guarantees provided by the programming language**
  - **safety guarantees**,
    - incl. memory-safety, type-safety, thread-safety
  - **access control**
    - visibility/access restrictions with eg. **public**, **private**
    - sandboxing mechanism inside programming language (**security monitor**)
  - **information flow control**
    - Flow of data inside code.
- Some features dependent on each other:
  - type safety relies on memory safety
  - sandboxing relying on memory & type safety

Programming languages can help security

- Providing the programmer with good **APIs/libraries**
  - APIs with parametrised queries/prepared statements for SQL – more secure string libraries for C
- Including convenient **language abstractions**,
  - Exceptions, to simplify handling error conditions
  - Execution monitor to encapsulate code
- **Making assurance of security easier by being able to understand code in a modular way**

Safety vs security

- **safety**: **protecting a system from accidental failures**
- **security**: **protecting a system from active attackers**

Precise border hard to pin down, but **what is good for safety is also good for security**, so often the distinction is not so relevant.

Un aspetto di safety è un aspetto di security, molto spesso queste due cose sono intra allacciante. Voglio prevenire che il sistema non si blocchi a causa di fallimenti, magari non previsti. Security invece vuol dire che io voglio proteggere un sistema dall' attaccante, infatti normalmente quando parliamo di sicurezza abbiamo il modello del comportamento del sistema, il modello dell'attaccante e l'analisi della sicurezza, proprio perché pensiamo un' entità esterna e sicuramente però le cose sono un pò intrallacciate quindi quello che è buono per la safety è sicuramente anche buona per la sicurezza.

Safe programming languages

- One can write insecure programs in **ANY** programming language:
  - one can forget input validation in any language
- Flaws in the **program logic** can never be ruled out
- Still...some safety features can be nice

Il punto è che uno può scrivere programmi scorretti in un qualunque linguaggio di programmazione, cioè avere un linguaggio di programmazione safe non garantisce che la logica sia bacata, quindi può scrivere programmi che fanno schifo anche all'interno di un linguaggio di programmazione safe. Il punto è che però alcune caratteristiche del linguaggio ti guidano nel produrre dei programmi che sono valuable, come direbbero gli inglesi per la sicurezza.

A (small) problem

The assignment

a[i] = (byte) b

Under which conditions do make sense?

Allora per capire quello che voglio dire, facciamo veramente un esempio semplice. Questo assegnamento a[i] prende un b che viene convertito in un byte e lo inserisce nell'array. Allora questo assegnamento, per essere valido intendo dire che abbia un senso compiuto dal punto di vista del linguaggio di programmazione, non dal punto di vista della logica del programma, deve soddisfare alcune condizioni.

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

a innanzitutto deve essere non null e deve essere un array di byte. Il fatto che sia non null vuol dire che io non posso cercare di accedere un qualcosa che non ha un valore associato come riferimento. La i deve essere un intero, non deve essere un intero negativo e deve essere minore della dimensione dell' array. Allora tutte queste sono le condizioni che permettono di rendere quel comando che li abbiam scritto, sensato.

Cioè da una parte abbiamo il programmatore e dall'altra parte abbiamo il linguaggio di programmazione che viene utilizzato per codificare in termini di una logica computazionale l'applicazione. Allora il dilemma è caratterizzato da quelle due cose che ho scritto, una in rosso e un'altra in blu. e Da un lato dice che è chi programma che è responsabile delle condizioni di safety e di sicurezza dell'applicazione. Vuol dire che nel caso precedente, prima di arrivare a questo punto, a questo assegnamento deve essere sicuro che ha a[] è non null ed è un array di byte. i deve essere non negativo e deve essere minore della dimensione della array e deve essere un intero. Devo poter fare l'operazione di casting. Chi programma che si deve assicurare che questo valga. L'altro approccio è che il linguaggio di programmazione va a verificare il codice e garantisce che questi tre punti che qui sono scritti vengono gestiti dal programmatore, in modo particolare se io sto accedendo un vettore nullo mi solleva un'eccezione, se io vado a dare un accesso al di fuori delle dimensioni dell' array mi fa un controllo e mi solleva un'eccezione.

Programmer vs programming language

1. **The programmer is responsible for ensuring safety conditions**
2. **The language is responsible for checking "safe" approach**

- Lots of debates about the pros & cons highlight tension between
  - flexibility, speed and control vs safety & security
- But … **execution speed != speed of development of secure code**
- **….. and maybe programmers are more expensive than CPU**

Se lasciamo tutto al programmatore e quindi a chi programma diamo a chi programma una maggiore flessibilità. Perché se lo fa il sistema vuol dire che ci abbiamo un po di overhead, ma il punto che normalmente viene detto è: guardate che la velocità di esecuzione e la flessibilità nello sviluppo di programmi è diversa dalla velocità di sviluppo di un codice sicuro e con velocità di sviluppo intendo dire dal tempo che viene impiegato per fare in modo che io ho un codice di un applicazione che rispetto al modello dell attaccante mi garantisce che l' attaccante non è in grado di prendere il controllo. Seconda cosa, costa molto di più il tempo di un uomo, quello che i manager chiamerebbero il tempo uomo nel senso nel costo dello stipendiale di una persona che cpu. Ovviamente il fattore umano è decisamente importante.Se il valore non può essere fatto il casting mi solleva un eccezione. Il linguaggio di programmazione solleva un'eccezione a tempo di esecuzione presentando una situazione anomala. Ma lo potrebbe fare anche staticamente se riesce a determinare alcuni aspetti. Allora ovviamente queste due dimensioni di analisi hanno dei pro e dei contro

Safe programming languages

- Safe programming languages impose **some discipline** or restrictions on the programmer
  - **provide suitable abstractions to the programmer, with associated guarantees**
- But … this takes away some freedom & flexibility from the programmer,
- … hopefully extra safety and clearer understanding makes it worth this.

#1 Safety (for us)

- A programming language can be considered safe if one can **trust** the **abstractions** provided by the programming language
- The programming language **enforces** these abstractions and **guarantees** that they **cannot be broken**
  - boolean is either true or false, and never 23 or null
  - Programmer doesn't have to care if true is represented as 0x00 and false as 0xFF or vice versa

La prima cosa che intendo è che io considero un linguaggio di programmazione safe, se il programmatore o chi lo usa può essere sicuro, confidente, della affidabilità delle astrazioni definite dal linguaggio di programmazione, ovvero le astrazioni definite dal linguaggio di programmazione non possono essere rotte da un attaccante. Se ho un linguaggio di programmazione che dice come sono i booleani, i booleani sono true o false indipendentemente da come saranno rappresentate dentro alla macchina. Non gliene frega niente al programmatore se ho rappresentato true con il valore 0x00 e il valore false con 0xFF. Vuol dire l' astrazione presente nel linguaggio hanno delle garanzie di affidabilità

Il linguaggio di programmazione mette un comportamento che è chiaro e preciso per ogni astrazione presente nel linguaggio. Vuol dire che la descrizione del linguaggio di programmazione non può lasciare il comportamento indefinito. Gli esempi che noi abbiamo visto, nel caso di errori di memory safety, erano degli esempi in cui lo standard del C o del C++ diceva che quella è la situazione in cui non c'era qualcosa da fare, per cui una qualunque scelta del compilatore andava bene, perché il comportamento nel caso limite dell'errore non era definito.

#2 Safety (for us)

Programs have a **precise & well defined semantics** (ie. meaning)

One can understand the behaviour of programs in a **modular** way

Program behaviors are not left **undefined**

Quindi quello che intendo dire per safety relativamente a i comportamenti è che sia chiaro il modello del comportamento. Questo è il motivo per cui quando abbiamo parlato di science security abbiamo detto: deve essere sempre chiaro il modello del comportamento e non ambiguo. I

I secondo aspetto importante è che uno deve essere in grado di ragionare su programmi e di mettere assieme programmi in modo composizionale, ovvero deve avere un meccanismo che ci permette di affrontare la complessità, e un meccanismo ancora una volta di natura linguistica, che ci permette di affrontare la complessità della decomposizione del programma in sottoprogrammi.

Safe programming languages

Warning: this is overly simplistic, as there are many dimensions of safety

Sulla parte estrema della sinistra, c'è il codice macchina che è piuttosto unsafe molto molto efficiente, molto flessibile ma complicato ad esempio di ragionare in modo composizionale. Un po più safe di Java abbiamo Scala, un linguaggio di programmazione che integra la programmazione imperativa, la programmazione funzionale, la programmazione a oggetti e la programmazione ad eventi come il modello ad attori in un solo contesto.

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 some dimensions, there can be many levels of safety

Il safety ha diverse dimensioni, perchè si parlare di memory safety di type safety di arithmetic safety se vi ricordate un esempio di un'esercitazione, abbiamo visto esattamente questo aspetto. Vorremmo avere delle possibilità di usare dei valori che sono null negli oggetti, dovremmo avere la possibilità di comprendere che cosa vuol dire avere delle strutture dati immutable. Vogliamo capire che cosa vuol dire avere l' assenza di aliasing. Sono tutti aspetti di safety che hanno un ruolo.

Levels of safety: buffer overflow

1. let an attacker inject arbitrary code
2. *possibly* crash the program (or else corrupt some data)

'unsafe'; some undefined semactics

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

'safe'

Per andare a vedere questo ruolo delle dimensioni di safety prendiamo questo esempio Andiamo a vedere le diverse dimensioni. La dimensione con indice intero più basso sono quelle più unsafe, le dimensioni con indice più alto sono quelle più safe. Sicuramente un buffer overflow è terribilmente unsafe se permette di fare un injection di un codice di un attaccate. Dall'altra parte nella dimensione più safe, il tentativo di scrivere al di fuori di una dimensione non buffer, se ne accorge il compilatore, quindi non viene mandato in esecuzione. Però ci sono diverse dimensioni. Ad esempio sempre unsafe ma un po' meno unsafe di dare contro all'attaccante è il fatto che sotto certe condizioni, quando ad esempio si corrompono dei dati, il programma potrebbe fare un'abort a run time. Oppure potrebbe definitivamente, quindi un segmentation fault, potrebbe quindi terminare a tempo di esecuzione. Ovviamente è una forma un po più safe perché non da controllo all' attaccante, però fa abortire il programma, quindi ovviamente non è gradevole, infatti un altro livello maggiore di safety è garantire che il programma possa e il programmatore possa gestire, tramite l'eccezione, le politiche e scrivere da programma le politiche di gestione delle situazioni anomale che si possono presentare. Poi la politica potrebbe essere semplicemente "termino", ma ci potrebbero essere delle politiche che portano alla terminazione in un modo più gradevole (try-catch).

Se io vado a vedere il la toolchain dei linguaggi di programmazione e voglio garantire delle proprietà di safety, ci sono delle cose che vengono viste a tempo di compilazione, ovvero staticamente prima che il programma venga mandato in esecuzione. In modo particolare l'analisi semantica, il sistema dei tipi e quando dico analisi statiche, intendo dire ad esempio dei meccanismi tipo CFI che nella fase del back end dei compilatori vanno a definire qual'è il comportamento aspettato in termini di CFG. Oppure possono fare altra analisi che vanno a comprendere come è il flusso di provenienza dei dati, per comprendere, ad esempio, aspetti di controllo e di sanificazione dei dati che provengono dall' input.

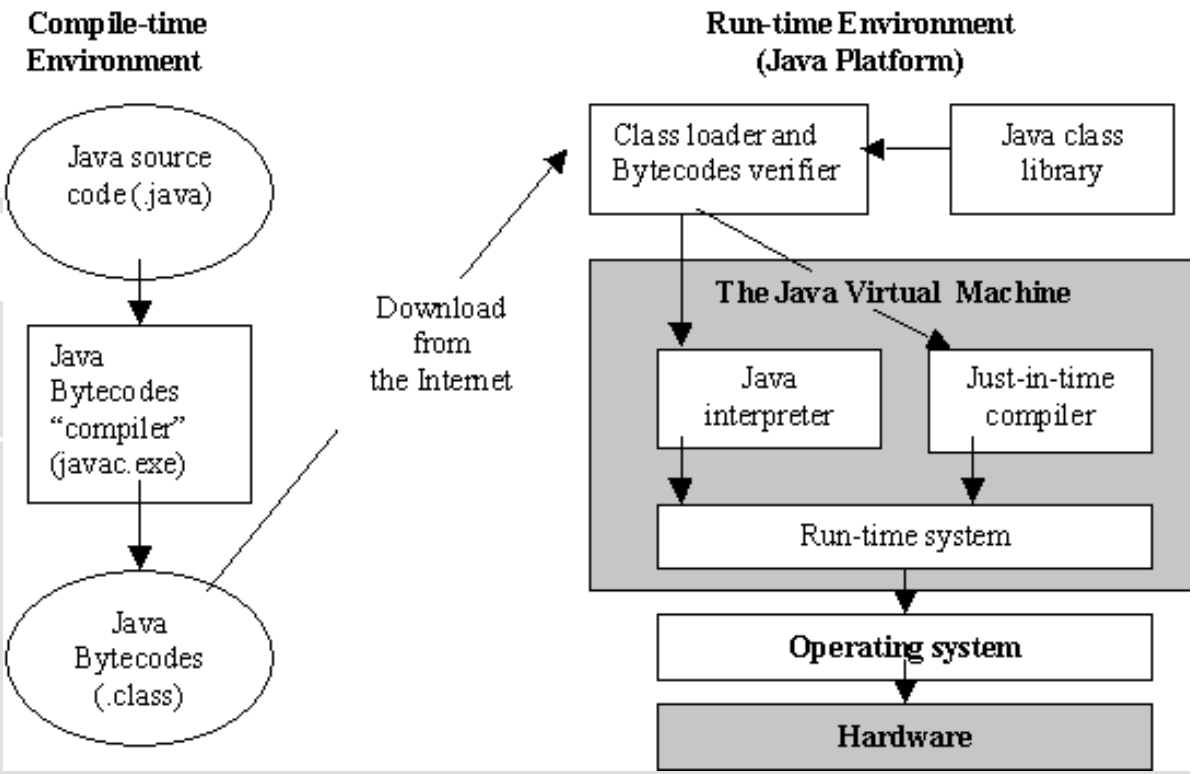
Ensuring safety

- Mechanisms to provide safety include
  - **compile time semantic analysis: type checking, static analysis** …
  - **runtime checks**, **array bounds checks, checks for nullness, runtime type checks**, …
  - **automated memory management using a garbage collector programmer does not have to free() heap-allocated data**
  - using an **execution engine** (*the virtual machine of the language*) **to do the things above**

Tutti questi aspetti vengono gestiti staticamente dal programmatore. Poi ci sono degli aspetti che deve gestire il run time dell'esecuzione del linguaggio, ad esempio in alcuni casi non è possibile sapere se un accesso a un array ricade all'interno della dimensione dell' array. Allora i compilatori della classe del c semplicemente se ne strafottono di questo problema in Java questo problema è gestito dal run time. Non solo, in alcuni linguaggi vengono fatti dei check a run time per andare a vedere i valori null oppure vengono fatti dei controlli a run time per andare a controllare tutte quelle informazioni di tipo che erano solo in parte note a livello di compilazione. Poi ci sono altri aspetti della gestione della memoria dinamica, ad esempio il garbage collector. Uno può avere un esecution engine che è diverso dalla macchina hardware nuda e cruda che uno ha di fronte. Potrebbe avere una macchina virtuale dell esecuzione del linguaggio. La macchina virtuale dell esecuzione del linguaggio si può occupare di tutte queste cose che abbiamo detto pocanzi, ma se ne può occupare, perché il comportamento della macchina virtuale del linguaggio è stato completamente specificato nello stesso stile in cui il comportamento del linguaggio è stato specificato.

Example: the Java Virtual Machine

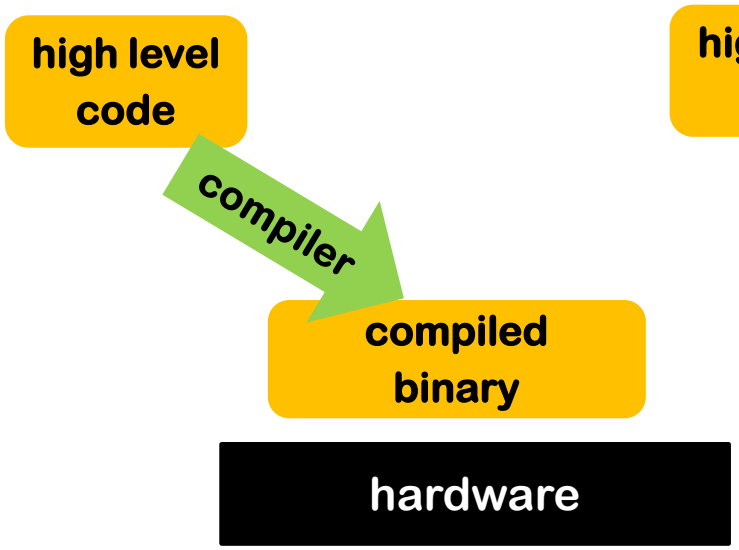
- The javaVirtualMachine(JVM)
  - runs the **bytecode verifier** (bcv) to type-check code,
  - performs some **run time checks**
  - periodically invokes the **garbage collector**



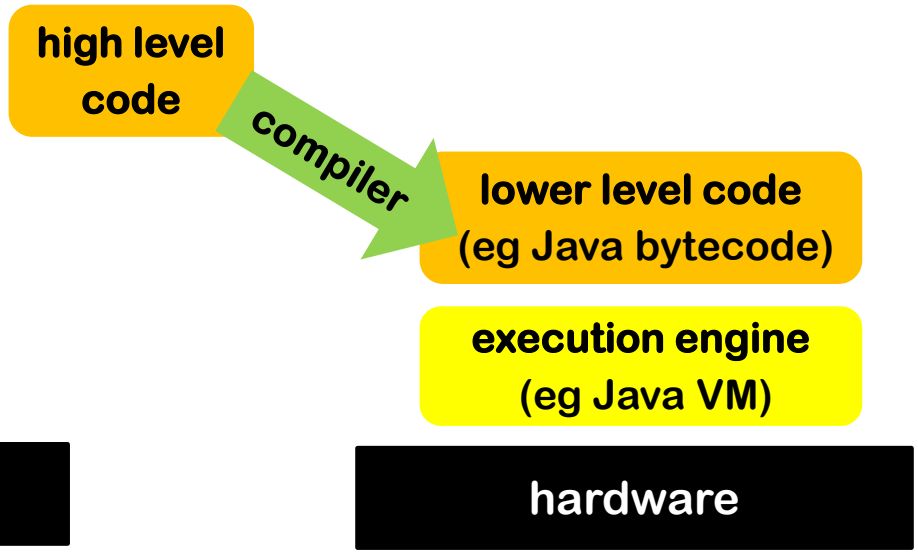

ON PROGRAMMING LANGUAGE IMPLEMENTATION

COMPILATION VS VIRTUAL MACHINES

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 the execution engine can provide checks at runtime

Language based Security: memory safety

- A programming language is **memory-safe** if
  - programs can never access unallocated or de-allocated memory
    - no segmentation faults at runtime
- Program can never access uninitialised memory
  - Default values

Memory safety

- We could switch off OS access control to memory.
  - Assuming there are no bugs in our execution engine...
- We don't have to zero out memory before de-allocating it to avoid information leaks (within the same program).
  - Again, assuming there are no bugs in our execution engine...

Unsafety

- Unsafe language features that break memory safety
  - no array bounds checks
  - pointer arithmetic
  - null pointers,
    - but only if these cause undefined behaviour

Folk tales (C null pointers)

Common (and incorrect!) C-folklore:

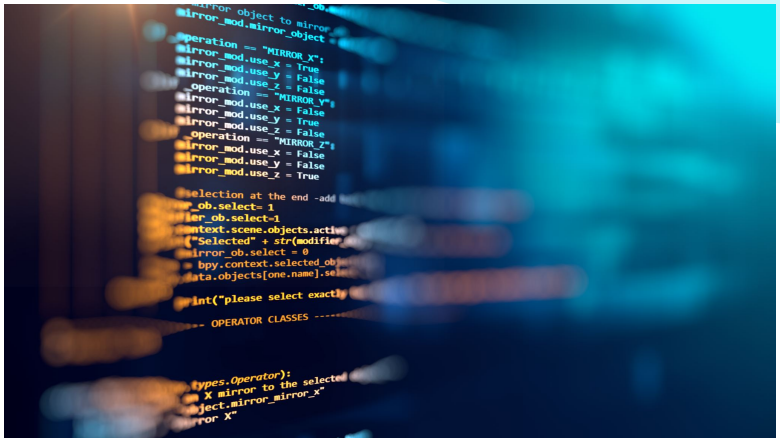
Dereferencing a NULL pointer will crash the program.

The C standard only guarantees:

the result of dereferencing a null pointer is undefined.

Ad esempio questo qui è un folk tales relativo a C, quindi una favola. Sostanzialmente è un folklore che dice che se io vado a dereferenziare un puntatore null il programma termina, ma questo lo standard del C mica lo dice. Lo standard del C dice soltanto che il risultato di dereferenziare un puntatore a null non è definito fine. Quindi a quel punto questo è un tipico esempio di quelle cose che dicevo prima, cioè un qualunque comportamento a questo punto è ammesso perché non è definito quindi qualunque cosa se non è definita va bene.

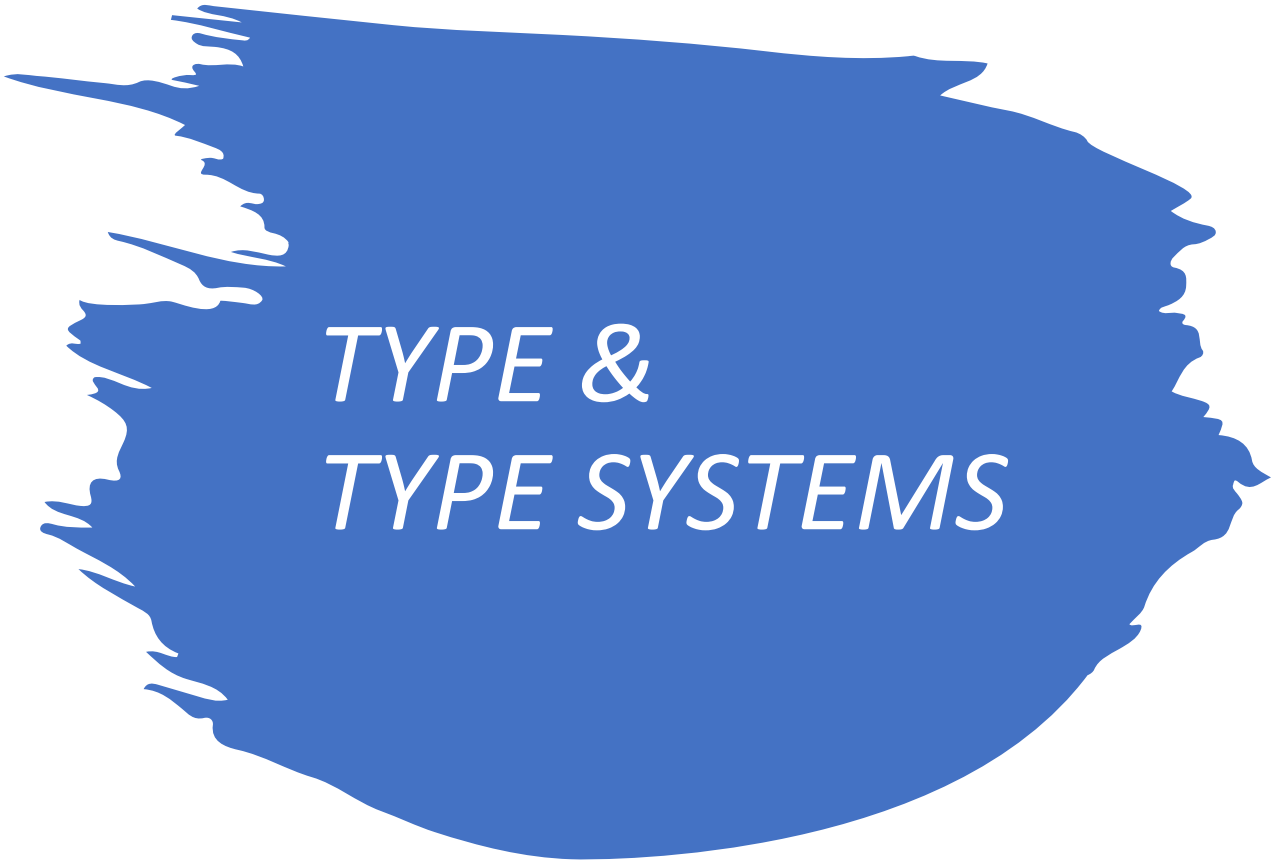
C Null Pointers
(making tales more concrete)



See Secure Coding guidelines for C

https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers

for discussion of a security vulnerability in a PNG library caused by a null dereference that didn't crash (on ARM processors).


TYPE &
TYPE SYSTEMS

Types

Allora un'altra cosa che è abbastanza importante nei linguaggi di programmazione sono la struttura dei tipi, I tipi asseriscono delle proprietà invarianti relativamente ai costrutti e all' astrazione presente nel linguaggio di programmazione. Se io dico che una variabile ha tipo int vuol dire che quella variabile conterrà un valore intero. Il valore intero dipenderà da come sono rappresentati gli interi in quella macchina, ma se il mio sistema di tipi mi dice che quello è un intero, non mi devo preoccupare di come saranno rappresentati in quella macchina.

- Types assert **invariant properties** of program element
  - Variable **x** will always hold an **integer**
  - Method **m** will always return an object of class **A** (or one of its subclasses)
  - The array **arg** will never store more than 10 items
- **Type is a property of program constructs**
- There is a wide range of expressivity in type systems!
  - Good reference **Types and Programming Languages** Benjamin C. Pierce, MIT Press

Son sicuro che non lo posso rompere perchè quello è un intero. Il sistema di tipi vi dirà anche che il metodo m restituirà un oggetto della classe A o uno delle sue sottoclassi, se uso quel meccanismo del principio di sostituzione. Bene la segnatura del metodo, mi definisce esattamente le proprietà dei parametri in ingresso e in uscita del main. Quando io ho un metodo che mi prende un vettore arg e dice che è di dimensione 10 bene questo vuol dire che non posso metterci più di 10 elementi. Allora questo cosa vuol dire? Vuol dire che le strutture dei tipi è una struttura che dice come sono fatte le proprietà del linguaggio in termini dei costrutti elementari del linguaggio. A questo punto normalmente uno si fida e perché si fida dei tipi, perché i linguaggi di programmazione operano con un meccanismo che permette di avere l'affidabilità dei tipi.

Sostanzialmente un sistema di tipi viene eseguito in fase di analisi statica subito dopo aver definito la rappresentazione intermedia e quella che normalmente viene chiamata analisi semantica che usa la tabella dei simboli e che assegna ai tipi i costrutti del programma e va a vedere se usando le informazioni relativamente alla struttura del sistema dei programmi, il programmatore ha scritto un programma che viola la proprietà dei tipi.

Type systems

A type system is a collection of rules that assign types to program constructs (more constraints added to checking the validity of the programs, violation of such constraints indicate errors)

A languages type system specifies which operations are valid for which types

Type systems provide a concise formalization of the semantic checking rules

Type rules are defined on the structure of expressions

Type rules are language specific

Vuol dire che il linguaggio di programmazione deve definire esattamente quali operazioni sono valide per tutti i tipi e per tutti i costrutti del linguaggio, quindi questo vuol dire ancora una volta che per essere safe e per ammettere controllo dei tipi, devo definire tutti i possibili casi di uso. Allora a questo punto il sistema dei tipi permette di assegnare un tipo ai programmi. Tipicamente un sistema dei tipi opera tramite regole, ovvero opera per composizionalità

Se per definire il tipo di un'espressione complessa si determina il tipo degli argomenti e in base alle caratteristiche che il linguaggio di programmazione, nella definizione del linguaggio, associa al costruttore principale dell'espressione del comando, viene definito il tipo del risultato.

Why do we need type systems

- Consider the statement written in some assembly language fragment
  - **addi $r1, $r2, $r3**
- What are the types of $r1, $r2, $r3?
- Assembly language is untyped
- This instruction allows you to add the contents of a register to an immediate value (a constant) and store the result in a (possibly) another register

Stiamo facendo la somma dei rigidi, dei valori contenuti nei registri r2 ed r3 e lo mettiamo nel registro di r1. Qual è il contenuto del registro? Una costante, un intero qualunque cosa qual'è il risultato? Nell'altro registro che cos'è? Un intero? Non lo so dal punto di vista dell'assembly sono sequenze di bit né più e né meno per cui, ad esempio, posso fare la somma di quello che esternamente rappresentava come un intero, con un qualcosa che rappresentavo con stringa o con un pointer. Dal punto di vista dell'assembly questa cosa qui è sostanzialmente indifferente, visto che non a livello di implementazione uso sequenze di bit.

Why do we need type systems

- It doesnt make sense to add a function pointer and an integer in C
- It does make sense to add two integers
- But both espressions have the same assembly language implementation!

L'uso del sistema dei tipi è per evitare che ci siano errori, cioè garantirsi che se ho un sistema con un controllo statico dei tipi garantisco che non ci saranno mai a tempo di esecuzione degli errori dovuti a un uso non corretto dei tipi e quindi vuol dire che staticamente se c'è un uso non corretto dei tipi, il compilatore lo trova e non manda il programma in esecuzione.

Use of types

- **Detect errors:**
  - Memory errors, such as attempting to use an integer as a pointer.
  - Violations of abstraction boundaries, such as using a private field from outside a class.
- **Help compilation:**
  - When Python sees x+y, its type systems tells it almost nothing about types of x and y, so code must be general.
  - In C, C++, Java, code sequences for x+y are smaller and faster, because representations are known.

Però questa cosa qui è solo la prima parte della discussione, perché il sistema dei tipi garantisce anche un altra cosa, garantisce un aiuto in fase di compilazione, ovvero se io so che quell operazione è un'operazione tra interi, allora a questo punto posso utilizzare queste informazioni per determinare in modo efficiente la operazione da interi sulla macchina intermedia d'esecuzione del linguaggio, in modo tale che il codice eseguito sia eseguito in modo molto molto più efficiente, ovvero se io so che due variabili x e y sono interi e devo fare la somma di x più y, a questo punto riprendo subito la scelta di fare la somma tra interi e quindi a quel punto, conoscendo la rappresentazione, perché so il tipo, e ovviamente so la rappresentazione, perché quando vado a compilare sulla macchina intermedia, so come su questa macchina intermedia vengono rappresentati i valori, a questo punto ho del codice più efficiente e più snello.

Type checking and type inference

- **Type Checking** is the process of verifying fully typed programs
- **Type Inference** is the process of filling in missing type information
- The two are different, but are often used interchangeably

Io ho parlato indifferentemente di controllo dei tipi, però sono stato impreciso. Nei linguaggi di programmazione abbiamo la type checking e la Type inference che sono due cose diverse. Il type checking va a verificare che un linguaggio tipato rispetta il tipo. Vuol dire che io annoto dal livello del linguaggio di programmazione il type checking mi va a verificare se effettivamente la definizione dei tipi che ho dato è conforme con le regole del linguaggio. Invece la type inference è un meccanismo che chi programma non definisce, è il sistema che inferisce per lui e questo è, ad esempio, l'uso che abbiamo in ML o in Ocaml. Il sistema di inferenza dei tipi è usato nei generici sia in Java che C#. Con la Diamond notation uno può avere dei meccanismi di inferenza dei tipi sui tipi generici, esattamente nello stesso stile dell inferenza dei tipi di OCaml.

Static vs dynamic

- Type checking can be done
  - at **compile time** (**static typing**) or
  - at **runtime** (**dynamic typing**)
  - or a combination.
- **Type soundness** (aka **type safety** or **strong typing**)
  A language is type safe if the assertions are guaranteed to hold at run-time

Il type checking può essere fatto a compilazione, a tempo di esecuzione o con una combinazione dei due, ad esempio nei linguaggi tutti completamente interpretati, vien fatto dinamico. In Java è fatto un po statico, un po dinamico. A questo punto quello che uno deve garantire è la correttezza, quella che si chiama la Type soundness, cioè deve garantire che se un linguaggio è Type safe, allora vuol dire che le informazioni relative alle garanzie di tipo devono valere anche a runtime.

INTERMEZZO

TYPE ANALYSIS

VEDERE CODICE types.ml

La volta scorsa abbiamo visto i tipi statici di un linguaggio di natura funzionale perchè erano i tipi che noi avevamo creati come annotazioni delle espressioni che catturavano i comportamenti astratti del programma. Esistono altre nozioni che emergono a tempo di esecuzione con i dati concreti e non quelli astratti. Nei linguaggi ad oggetti uno ha la nozione di tipo della classe a tempo di compilazione.

Static types and dynamic types

- The dynamic type of an object is the class *C* that is used in the "new *C*" expression that creates the object
  - A run-time notion
  - Even languages that are not statically typed have the notion of dynamic type
- The static type of an expression is a notation that captures all possible dynamic types the expression could take
  - A compile-time notion

Il polimorfismo di sottotipo, nei linguaggi come per esempio il Java, il tipo statico è un tipo più generale che contiene tutti i sottotipi dinamici contenuti in quella classe

Qui vedete un qualcosa scritto in un linguaggio tipo Java e vedete, ci sono tutte le informazioni di natura statica che io posso associare all'oggetto ad esempio, vedete, abbiamo quella, quella string che si chiama greeting che è una costante che è privata, quindi è è un informazione di tipo statico che ci dice che è accessibile soltanto all'interno della classe demo e quindi dall'esterno di questa classe non è visibile e il fatto che venga messo che abbiamo definito una costante in un valore intero che chiamiamo con CONST con Final static Int ci dice che quella è una costante che è una variabile d'istanza perché c'è il costrutto statico, il costrutto statico che vuol dire che appartiene a tutte le istanze degli oggetti della classe e non alle variabili d'istanza.

Example

```
public class Demo{
  static private string greeting = "Hello";
  final static int CONST = 43;

  static void  Main (string[]  args){
     foreach (string name in args){
         Console.Writeline(sayHello(name));
     }
  }

  public static string sayHello(string name){
     return greeting + name;
  }
}
```

`greeting` only accessible in class Demo

`CONST` will *always* be 43

sayHello will always retur a string

sayHello will always be called with 1 parameter of type string

Un'altra informazione statica è la segnatura dei metodi, ad esempio il metodo sayHello ha una segnatura che ci dice che prende una stringa come parametro e restituisce una stringa. Queste son tutte nozioni di tipo statico che sono gestite dal compilatore.

Type safety

- Type-safety programming language guarantees that programs that pass the type-checker can only manipulate data in ways allowed by their types
  - One cannot multiply booleans, dereference an integer, take the square root of reference, ...
- Type safety avois lots of room for undefined behaviour
  - In OO languages: no "Method not found" errors at runtime

Discussion

- **Programming languages can be**
  - memory-safe, typed, and type sound:
    - Java, C#, Rust, Go
      - though some of these have loopholes to allow unsafety
    - Functional languages such as Haskell, ML, OCaml, F#
  - memory-safe and untyped
    - LISP, Prolog, many interpreted languages
  - memory-unsafe, typed, and type-unsafe
    - C,C++
      - Not type sound: using pointer arithmetic in C, you can break any guarantees the type system could possibly make

Breaking type soundness (in C++)

For a C(++) program we can make no guarantees whatsoever in the presence of untrusted code.

a buffer overflow in some library can be fatal
in a code review we have to look at all code to make guarantees

```
class DiskQuota {
  private:
     int MinBytes;
     int MaxBytes;
};

void EvilCode(DiskQuota* quota) {
   // use pointer arithmetic to access
   // the quota object in any way we like!
   ((int*)quota)[1] = MAX_INT;

}
```

Avoiding buffer overflow (Java, C#)

- Language-based solution: Buffer overflow is ruled out at **language-level,** by combination
  - compile-time typechecking (static checks)
  - at load-time, by bytecode verifier (bcv)
  - runtime checks (dynamic checks)

```
public class A extends Super{
  protected int[] d;
  private A next;

  public A() { d = new int[3]; }
  public void  m(int j) { d[0] = j; }
  public setNext(Object s)
        next = (A)s;
  }
}
```

runtime checks for 1) non-nullness of d, and 2) array bound

runtime check for type (down)cast

Java: run-time checks

Buffer overflow still exists in Java and C#

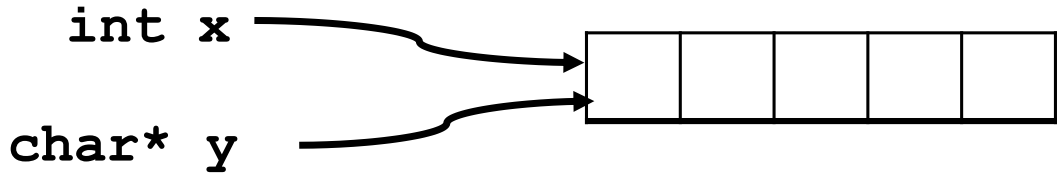
- Buffer overflows can still exist:
  - in native code
  - C#: code blocks declared as unsafe
  - through bugs in the Virtual Machine (VM) implementation, which is typically written in C++....
  - through bugs in the implementation of the type checker, or worse, bugs in the type system (unsoundness)
- The VM (incl. the type checker, byte code verifier) is part of the **Trusted Computing Base** (TCB) for memory and type-safety,

Discussion

Type safety is an extremely **fragile** property:
one tiny flaw brings the whole type system crashing down

Example: type confusion

Data values and objects are just blobs of memory. If we can create type confusion, by having two references with different types pointing the same blob of memory, then *all* type guarantees are gone.



Type confusion attacks

```
public class A{

    public Object x;

     ...

}
```

```
public class A{

  public int x;

  ...

}

public class B{

 void setX(A a) {

  a.x = 12;

 }

}
```

What if we could compile B against A but we run it against A?

We can do pointer arithmetic again!

If Java Virtual Machine would allow such so-called *binary incompatible* classes to be loaded, the whole type system would break.

Type system (soundness)

- **Representation independence**
  - Booleans: it does not matter if we represent true as 0 and false as 1 (or viceversa),
  - if we execute a given program with either representation, the result is guaranteed to be the same
  - we could test this, or try to prove it.
- **Theory helps: Given a formal mathematical definition of the programming language, we could prove that it does not matter how true and false are represented for all programs**

Type systems (soundness)

- Two formal definitions of the programming language
  - a **typed operational semantics**, which records and checks type information at runtime
  - an **untyped operational semantics**, which does not
- Prove their equivalence for all well-typed programs:
  - **prove the equivalence of a defensive execution engine (which records and checks all type information at runtime) and a normal execution engine which does not for any program that passes the type checker.**
- Research: Several works have formalised the semantics and type system of Java, using theorem provers (Coq, Isabelle/HOL), to prove such results.

Richer forms of types

- Distinguish **non-null** & **possibly-null** (nullable) types

  public @NonNull String hello = "hello";
  - to improve efficiency
  - to prevent null pointer bugs or detect (some) of them earlier, at compile time
- Programming language mainstream:
- C# supports nullable types written as A? or Nullable<A>
  - In Java you can use type annotations @Nullable and @NonNull
  - Scala, Rust, Swift have non-null vs nullable option types

Richer forms of types

- **Alias control:** restrict possible interferences between modules due to aliasing.
- **Information flow**: controlling on the way tainted information flows through an implementation.
- **Resource usage and acccess control**: controlling accesses to resources

We will discuss Information flow, Resource usage and access control in furter lectures

Richer forms of types

- **Alias control:** restrict possible interferences between modules due to aliasing.
- **Information flow**: controlling on the way tainted information flows through an implementation.
  - We will discuss in lab
- **Resource usage and acccess control**: controlling accesses to resources
  - We will outline the main design ideas

Language-based guarantees

- **visibility**: public, private, …
  - private fields not accessible from outside a class
- **immutability**
  - In Java: final int i = 5;
  - in C(++) : const int BUF_SIZE = 128;
  - In Java String objects are immutable
- Scala, Rust provides a more systematic distinction between mutable and immutable data to promote the use of immutable data structures


Thread Safety &
Aliasing

Races

- Two threads both execute the statement

  x = x+1;

  where **x** initially has the value **0**.
- What is the value of x in the end?
  - **x can have value 2 or 1**
- **Data race**: x = x+1 is not an **atomic operation**, but happens in two steps - reading x and assigning it the new value - which may be interleaved in unexpected ways

```
class A {
        private int i ;
        A() { i = 5 ;}
        int geti() { return i; }
 }
```

Can geti() ever return something else than 5?

Yes!

Thread 1, initialising x

```
static A x = new A();
```

Thread 2, accessing x

```
j = x.geti();
```

You'd think that here x.geti() returns 5 or throws an exception, depending on whether thread 1 has initialised x

Execution of thread 1 takes in 3 steps

1. allocate new object m
2. m.i = 5;
3. x = m;

Hence: x.geti() in thread 2 can return 0 instead of 5

the compiler or VM is allowed to swap the order of these statements, because they don't affect each other

Solution

```
class A {
        private final int i ;
        A() { i = 5 ;}
        int geti() { return i;}
}
```

Now geti() always return 5.

Declaring a private field as final fixes this particular problem

- due to ad-hoc restrictions on the initialisation of final fields

Data races & thread safety

- A program contains a **data race** if **two execution threads simultaneously access the same variable and at least one of these accesses is a write**
  - data races are highly non-deterministic, and a pain to debug!
- **thread-safety** = **the behaviour of a program consisting of several threads can be understood as an interleaving of those threads**
- In Java, the semantics of a program with data races is effectively undefined:
- **Moral of the story: Even "safe" programming languages can have very weird behaviour in presence of concurrency**

Java Signature Bug

the method getSigners returns an alias to the private array signers. Because arrays in Java are always mutable, this allows untrusted code to obtain a reference to this array and then, change the content.

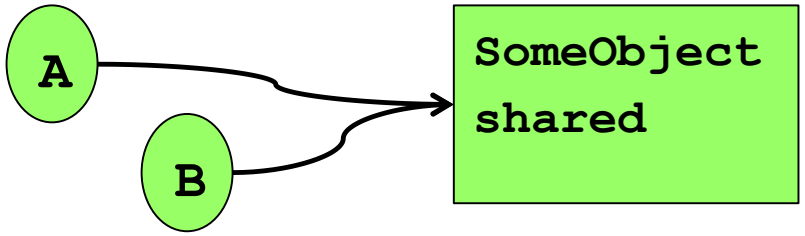
```
package java.lang;
public class Class {
   private String[] signers;

   /** Obtain list of signers of given class */
   public String[] getSigners() {
      return signers;
   }
 ...
```

The solution is for getSigners to return a copy of the array signers rather than an alias.

Threads and aliasing

Aliasing: two threads or objects A and B both have a reference to the same object `shared`



Aliasing (Issues)

- Concurrent (multi-threaded) context: data races
  - Locking objects (synchronized methods in Java) can help… expensive & risk of deadlock & lock-free data structure
- Single-threaded context: dangling pointers
- Single-threaded context: broken assumptions
  - If A changes the shared object, this may break B's code, because B's assumptions about shared are broken

```
1  public void f(char[] x){
2    if (x[0] != 'a') { throw new Exception(); }
3    // Can we assume that x[0] is the letter 'a'  here?
4    // No!! Another concurrent execution thread could
5    //       change the content of x at any moment
```

If there is aliasing, another thread can modify the content of the array at any moment.

In a multi-threaded program, aliasing of immutable data structures are safer.

```
1   public void f(String x){
2     if (x.charAt(0) != 'a') { throw new Exception(); }
3     // We CAN assume that x[0] is the letter 'a‘ here?
4     // Yes, as Java Strings are immutable
5       ...
```