# Type Safety

## Target Course

Programming Languages

## Learning Goals

A student shall be able to:

1. Apply principles of secure design and defensive programming techniques when developing software.

## IAS Outcomes

The CS2013 Information Assurance and Security outcomes addressed by this module are:

| IAS Knowledge Topic | Outcome |
|---|---|
| Defensive Programming | 2. Explain why you might choose to develop a program in a type-safe language like Java in contrast to an unsafe programming language like C/C++. |

## Dependencies

- Continues the discussion of the BMI example presented in the previous module.
- Cover at the same time that iteration is introduced. Material provides a nice way to motivate while loops.
- Material assumes selection has been covered.
- Optional material assumes knowledge of functions.

# Dependencies

- Cover material as type-safety is being discussed.

# Summary

This module defines type safe languages and gives examples of how using type un-safe languages incorrectly can leave an application open to security risks.

**Estimated Time:** Cover in a 30 minute lecture

# Materials

*What is a type-error?*

A type error is an error or undefined behavior which arises when a program attempts to perform an operation on a value on which the operation is undefined for. For example, a programming language might allow us to treat a boolean as an integer and allow a boolean to be added to an integer. Although the result will be undefined the language raises no compile or run time errors.

*What does type-safety mean?*

The type-safety of a programming language is the extent to which it prevents type-errors. The language can prevent a type-error at compile type or at runtime. For example consider a 32-bit quantity in the machine. It can represent an int, a floating point or 4 ASCII characters. Any of these interpretations may be correct given the context. In certain programming languages (e.g. assembly), the programmer is entirely responsible of keeping track of the data types. If the programmer grabs a 32-bit number which really represents an integer and performs a machine-level floating point addition on it, the result is undefined, meaning the results may be unpredictable from one computer to the next. Higher level programming languages usually


This document is licensed with a Creative Commons Attribution 4.0 International License ©2017

reduce the burden the programmer has on keeping track of data types, by identifying type errors at compile or run-time, there-by reducing undefined behavior. Some languages such as C/C++ define many constructs as undefined behavior, while other languages, for example Java, have less undefined behavior. The extent to which a language provides type-error checking at compile time or runtime determines its type-safety.

*What are some examples of type safe and un-safe languages?*

Versions of ML, Python and Java are believed to be type safe. Although C and C++ (even more so) are type-safe in many contexts, both languages also contain several common features which are not type-safe.

*What are some examples of type-safety issues in C or C++?*

There are many examples of type-unsafe behaviors in C and C++. Some involve understanding compiler and machine level details. (See all references for details). Here we restate an example from Keunwoo Lee's lecture notes [3].

Unlike in Java, C casts are *not* dynamically checked for "truthfulness" --- if the values are incompatible, the compiler will *do the best it can*, by reinterpreting the in-memory bit pattern of the expression being casted as if it belonged to the type being casted to. Below a pointer to a character is being cast to a pointer to a double:

```
void f(char* char_ptr) {
    double* d_ptr = (double*)char_ptr;
    (*d_ptr) = 3.5;
}
```

*Why is this risky?* Casts provide a trivial way to store arbitrary instructions into code addresses. A standard C-code "exploit" (malicious attack) involves writing some "bad" instructions into some location in memory that will later be executed. These instructions might, for example, store the entry location of virus-code.

```
/* pretend 0x010000 is a hexadecimal address of code */
int i = 0x010000;

/* treat this integer as a pointer, using a cast */
int * i_ptr = (int*)i;

/* store through this pointer --- placing bad data at address */
(*i_ptr) = 0xBADCAT;
```

Later, when the code jumps to `0x010000`, the code will execute the instruction `0xBADCAT` instead of whatever instruction was there before. (See [4] for an example of a casting type-error in C++.)

Examples of behavior which are not type-safe also result from the language allowing the programmer more control over memory allocations. For example it might be left to the programmer to check array bounds and failure to do so is the cause of buffer overflow attack. Here is an example to illustrate:

```
int buf[4];
buf[5] = 3; /* overwrites memory */
```

The code above overwrites whatever value was in memory location one away from the end of buf[4]. The location may have contained a string or a floating number but will now be overwritten with a 3. A language which is not type-safe would allow this to occur with no errors. Now, consider replacing the 3 above with an input from the user e.g.:

```
buf[5] = getUserInput(); /* overwrites memory */
```


This document is licensed with a Creative Commons Attribution 4.0 International License ©2017

Suppose the memory location being overwritten is actually a return address. Then one could craft a malicious input which overwrites the valid return address with the address of malicious code.

*Why should we choose to develop in a type-safe language like Java rather than in C/C++?*

Languages like C/C++ are optimized to produce efficient code in a manner which is not type-safe. This leads to situations where compiled code may not be what the programmer intended. C and C++ also allow the programmer to use powerful constructs which are not type-safe and using these incorrectly can result in unexpected program behavior and errors which are difficult to detect. Many security vulnerabilities start out as memory or integer operations that have undefined behavior or type-errors.

Type safe languages offer more type-error identification so software developers in type-safe languages are usually less likely to develop code vulnerable.

While C/C++ may be necessary to implement certain performance-critical or low level routines many (see all references) take the stance that these languages should be reserved for only these instances.

> "In the long run, unsafe programming languages will not be used by mainstream developers, but rather reserved for situations where high performance and a low resource footprint are critical." [2]

*Are there any advantages to languages which are not type-safe?*

Allowing undefined behavior in C/C++ may simplify the compiler's job, and allow it to generate very efficient code. For example, checking array-bounds or using garbage collection may be unacceptably expensive for some scientific computations.

Some operators (which may lead to undefined behaviors) were provided by a language to give programmers explicit control over low-level issues like the exact layout of objects in memory.

Connections in other courses

- Cover in a hardware course where assembly language is covered to emphasize that assembly language has very little type checking.

**Assessment**

1. Which of the following statements are true? (***multiple answer circle all that apply***)
   a. Type checking may be performed either statically or dynamically.
   b. A strongly-typed language reduces security risk.
   c. A strongly-typed language must have type checking done during compilation.
   d. Name equivalence and structure equivalence are two ways to determine type equivalence.
   e. None of the above.

   Answer: a, b, d

## References

[1] Ian Barland's Manifestos. Accessed Aug 2015. http://www.radford.edu/ibarland/Manifestoes/whyC++isBad-example.cc


This document is licensed with a Creative Commons Attribution 4.0 International License ©2017

[2] Ian Joyner. C++?? : A Critique of C++ (3rd Ed.) Available at: http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/cppcrit/index010.htm

[3] Keunwoo Lee's lecture notes. Accessed Aug 2015. http://courses.cs.washington.edu/courses/cse341/04wi/lectures/26-unsafe-languages.html

[4] John Regehr's Blog entry. A guide to A Guide to Undefined Behavior in C and C++. Accessed Aug. 2015. http://blog.regehr.org/archives/213

[5] Xi Wang, Haogang Chen, Alvin Cheung, Zhihao Jia, Nickolai Zeldovich, and M. Frans Kaashoek. 2012. Undefined behavior: what happened to my code?. In *Proceedings of the Asia-Pacific Workshop on Systems* (APSYS '12). ACM, New York, NY, USA.


This document is licensed with a Creative Commons Attribution 4.0 International License ©2017