Fail-Safe C: the solution for preventing security holes in C programs

Yutaka OIWA (Research Team for Software Security, RCIS, AIST)

C language and security

In 1970s:

- Designed for early Unix systems
  - Simple, fast language
  - Flexible raw memory access using pointers (to replace assembly languages)

In year 2006:

- Causes many security holes
  - Lack of language-level memory safety
  - Lack of high-level support for complex data structures

>50% of CERT-reported security holes are caused by pointer misuses

- **Raw memory flexibility is not important for many programs.**
- **Safety is very important for current Internet-related programs.**

Fail-Safe C: a powerful solution for security problems in C

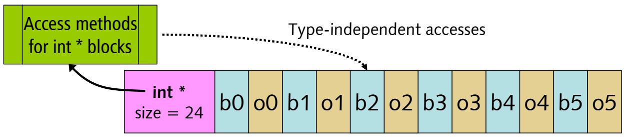
- **100% ANSI-C upper-compatible**
- **100% memory safe**
  - Safety comparable to Java, C#, Lisp or ML
- Supporting various C idioms
  - Not all programs are strictly ANSI-C compatible, sigh.
- Incurs as small overhead as possible

Implementation Techniques

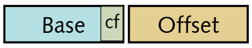
(1) Typed memory blocks & Access Methods

- Every memory blocks are “objects”: it knows
  - How many elements it contains
  - what is the type of its contents
  - how to read/write its contents

Even if a pointer is cast, memory accesses are safe using access methods associated to the referred block.



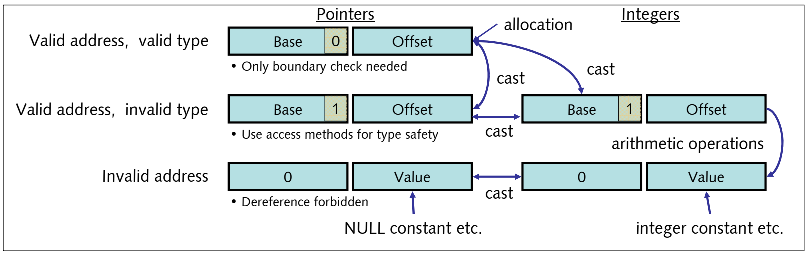
(2) Smart pointers & Cast flags



- Each pointer (represented in 2 words) remembers
  - which blocks it points to
  - whether it is cast or not (*i.e. type-valid pointer or not*)

If a pointer is not cast, memory accesses are fast.

- Access block contents directly, without access methods.
- Implementation trick used to reduce the access cost.
  - » No additional cost for cast flag checking.
  - » The same as that for Java and ML (in theory).



Other features

- Support for various uses of malloc()
  - Deferred type decision for dynamically-allocated blocks
  - Supports accesses for “remainder” area
    - For “buffer at struct tail” idioms.
- Type-safe linker supports ***safe*** separate compilation
  - It detects all type mismatch between modules.
  - Archive files are also supported.

Current status:

- Compiler and linker are available
  - Now support 100% of ANSI-C features
  - Easy-to-use: just type “**fscc**” instead of gcc.
    - As usual, users just see *.c, *.o, *.a, a.out files.
- “Safe” standard library implementations partly available
  - Custom “wrappers” implemented for 226/1108 functions
- Supports various existing programs
  - **OpenSSL, BIND9 (named), thttpd** with almost no source file modification
- Performance overhead: vary from x1.06 to x10 times.
  - Measured using BYTEmark2 and OpenSSL
  - Static optimizations/analysis will reduce them in future.

Project page: http://www.rcis.aist.go.jp/project/FailSafeC-en.html *or* http://failsafec.jp/

- Developers/researchers preview release will be available within this fiscal year (Dec. 2006 or Jan. 2007, hopefully).

* This project is partially supported by “New-generation Information Security R&D Program” from the Ministry of Economy, Trade and Industry (METI), Japan. A part of library implementation is done as a joint research with Lepidum, Co. Ltd.

Research collaborations

- Fail-Safe C to Java (Kamijima @ Tohoku Univ.)
  - See also his poster presentation in this session!
- VitC (FSC with information flow analysis, Furuse et al. @ U. Tokyo)

Related work

- CCured [Necula et al. 2002]
  - Compile-time analysis for “wild” (cast) pointer
  - Assumes all objects pointed by wild pointer as wild ⇒ many objects may be “polluted” by one wild pointer