

Cyclone: A safe dialect of C

CS6V81-002: Language-based Security

February 18, 2008

The Problem: C

- Many security violations in C programs are due to the design of the language itself
- The same characteristics that make C powerful can also lead to safety violations
- Current solutions offer only partial protection
- So why stick with C? Why not just use a safe language instead?
 - Legacy code
 - Efficiency
 - Memory management
 - Low-level control

The Solution: Cyclone

- A safe dialect of C:
 - Guarantee safety (no valid program can violate safety policies)
 - Maintain as much compatibility with C as possible
 - Make porting C programs feasible
 - Keep overhead to a minimum
 - Make it easy for C programmers to learn
- Add needed limitations to C
- Compensate by adding safe ways to perform unsafe tasks

The Cyclone Project

- Began as part of the Typed Assembly Language (TAL) project
- Popcorn, a language built to test TAL annotations, was the predecessor of Cyclone
- 2 years in development
- 110,000 lines of Cyclone code
 - 35,000 for the compiler
 - 15,000 for support libraries and programs
 - 50,000 to port benchmarking programs

NULL

- Attempting to read from, write to, or free a NULL pointer in a C program will probably cause the program to crash
- Cyclone compensates by inserting runtime checks
- To reduce the number of checks, "never-NULL" pointers can be used by simply replacing '*' with '@'
- What happens when a check fails?

Buffer Overflows

- Pointer arithmetic is a commonly used technique in C
- Bad pointer arithmetic can cause buffer overflows
- Cyclone compensates by not allowing arithmetic on *-pointers or @-pointers
- Instead, Cyclone introduces "fat pointers", represented by '?'
- Fat pointers include extra information needed for bounds checking
- This extra info adds overhead, but also guarantees safety and adds functionality

Uninitialized Pointers

- Similar to NULL pointer problems, C can allow operations on pointers that haven't been initialized, leading to horrible segmentation faults
- Cyclone solves this with static analysis, returning a compiler error
- However, the system is not perfect, and may force declaring variables earlier to avoid false detections

Dangling Pointers

- Returning a pointer to a local variable in C will yield another invalid pointer, this time to deallocated memory
- C compilers can check for this, but the checks are often ineffective
- Cyclone uses "region analysis" to verify that pointers only reference valid variables

Free

- The free function extremely hard to verify
- To ensure safety even over functionality, Cyclone's makes free a no-op
- This is admittedly "drastic"
- To compensate, Cyclone provides a garbage collector
- If finer grained control is needed, "growable regions" can be used
- These blocks resemble structs, but automatically deallocate when they go out of scope

Type-Varying Arguments

- C functions can be written so that variable types vary depending on how the function is called
- C compilers check these calls for safety, but can still let errors slip through
- Cyclone solves this issue with tagged unions, represented by "tunion"
- Similar to normal C unions, these allow one variable to hold multiple datatypes
- Tags are used to help the compiler determine which type to use

Goto

- Even most C programmers hate the very mention of goto
- The goto statement can allow scopes, which can allow invalid pointer references and many other problems
- Cyclone uses static analysis to ensure that a goto statement does not change scope
- Safe goto statements are allowed, while compiler errors are generated for unsafe ones
- Side Note: It is possible to write complex C programs using goto statements instead of functions (NOT recommended)

Performance

- Porting benchmarks from C to Cyclone required only ~10% of code lines to be changed
- Of these, 20-50% of the changes were just changing * pointers to ? pointers (fat pointers)
- Actual overhead varies greatly depending on how a program uses memory
- Some benchmarks require almost no overhead, while the worst was slower by a factor of 3
- Porting to Cyclone actually revealed errors in benchmarking programs

Conclusion

- Cyclone increases safety while staying as true as possible to the roots of C
- Increased safety is achieved without sacrificing the low-level functionality that makes C attractive in the first place
- Serious obstacles to portability still exist
- Even with proper standard libraries, etc., could Cyclone replace C?
- Would code-producers be willing to port legacy code to ANY C alternative?

References

- Trevor Jim, Greg Morrisett, Dan Grossman, Michael Hicks, James Cheney, Yanling Wang. "Cyclone: A safe dialect of C". In *Proceedings of the USENIX Annual Technical Conference*, Monterey, CA, June 2002
- "Cyclone User's Manual". Cyclone.
<http://cyclone.thelanguage.org/wiki/User%20Manual>
(accessed February 17, 2008).