Protean Code ltd
Process, Technology and Training for Critical Software

Sanitizing Sensitive Data: How to get it Right (or at least Less Wrong…)

Roderick Chapman, 22nd September 2017

Contents

Contents

- **The problem…**
- Technical issues
- Design goals
- Language support
- A policy for sanitization
- Related and further work

The problem…

Protean Code ltd
Process, Technology and Training for Critical Software

- "Secure coding" standards call for "sanitization" of "sensitive" data after it has been used.
  - What does this actually mean?
  - How do you do it?
  - How do you know you've got it right?
  - Oh.. and we had to do this for a client project, to meet GCHQ evaluation standards…

The problem…

Protean Code ltd
Process, Technology and Training for Critical Software

- Standards survey:
    - GCHQ IA Developers’ Note 6: Coding Requirements and Guidance
    - CERT Coding Standards
    - ISO SC22/WG23 Technical Report 24772
    - Common Weakness Enumeration (CWE)
    - Cryptography Coding Standard

The problem…

Protean Code ltd
Process, Technology and Training for Critical Software

- GCHQ IA Developers' Note 6: Coding Requirements and Guidance

  "Sanitise all variables that contain sensitive data (such as cryptovariables and unencrypted data) … sanitisation may require multiple overwrites or verification, or both."

  "if a variable can be shown to be overwritten shortly afterwards, it may be acceptable not to sanitise it, provided it is sanitised when it is no longer needed. 'Shortly' is not defined more precisely, since it will depend on the situation"

Protean Code ltd
Process, Technology and Training for Critical Software

The problem…

- GCHQ IA Developers' Note 6: Coding Requirements and Guidance

  "Sanitise all variables that contain sensitive data (such as cryptovariables and unencrypted data) … sanitisation may require multiple overwrites or verification, or both."

  "if a variable can be shown to be overwritten shortly afterwards, it may be acceptable not to sanitise it, provided it is sanitised when
  'Shortly' is not defined more
  on the situation"

What does this actually mean?

Protean Code ltd
Process, Technology and Training for Critical Software

Contents

- The problem…
- **Technical issues**
- Design goals
- Language support
- A policy for sanitization
- Related and further work

Technical issues

Protean Code ltd
Process, Technology and Training for Critical Software

- So why not just "write zeroes" to the variable?

```
declare
   T : Word32;
begin
   -- Do stuff with T;

   -- Now sanitize T
   T := 0;
end;
```

Technical issues

Protean Code ltd
Process, Technology and Training for Critical Software

- So why not just "write zeroes" to the variable?
  - T is local, therefore final assignment is *dead* in information-flow terms.
  - Optimizing compilers can remove the final assignment. Oops!
    - Modern compilers try *very hard* to remove redundant loads and stores.
  - "All zeroes" might not be a valid value, so can a legal assignment statement be written at all?

Technical issues

Protean Code ltd
Process, Technology and Training for Critical Software

- Derived values and copies…
- If A and B are “sensitive”, then

```
C := A op B;
```

- Is C also “sensitive”? Does C require sanitization?
- What about copies of sensitive date – in compiler-generated local variables, CPU registers, data cache? How do you sanitize those (without assembly language programming…)?

Technical issues

Protean Code ltd
Process, Technology and Training for Critical Software

- By-Copy parameter passing…
- See above – copies are bad!
- How do you sanitize a By-Copy “in” (aka “const”) parameter anyway???

Technical issues

Protean Code ltd
Process, Technology and Training for Critical Software

- CPU data caching and memory hierarchy.
- Memory subsystem in a modern CPU is really really complicated!
  - Multiple levels of cache.
  - Instruction re-ordering and write coalescing etc. etc.
  - Register renaming (more copies!)
  - Operating system paging and virtual memory? Has a copy of my sensitive data been written to disk?!?!

Contents

Protean Code ltd
Process, Technology and Training for Critical Software

- The problem…
- Technical issues
- **Design goals**
- Language support
- A policy for sanitization
- Related and further work

Design goals…

Protean Code ltd
Process, Technology and Training for Critical Software

- For a recent project…design constraints and goals:
  - Sanitization code in source Ada (using SPARK subset) – no assembly language required.
  - Portable – no use of non-portable or implementation-defined language features.
  - *No chance* of changing the compiler in the scope of this project.
  - “Bare metal” embedded target, so compatibility with GNAT Pro Zero-FootPrint (ZFP) runtime library.
  - Compatible with both SPARK 2005 and SPARK 2014 languages and verification tools.
  - “Just works” (with confidence) at *any* compiler optimization level.

Protean Code ltd
Process, Technology and Training for Critical Software

Contents

Ada support

Protean Code ltd
Process, Technology and Training for Critical Software

- Various language mechanisms were investigated, including…
  - Volatile aspect
  - Limited and By-Reference types
  - Inspection_Point pragma

Volatile

Protean Code ltd
Process, Technology and Training for Critical Software

- Easy! Just mark a sensitive object as Volatile, and the compiler will respect the reads and writes exactly as indicated in the source code…
- Better still … use Volatile *Types* for all sensitive data.
- Looks good, but…

Volatile

Protean Code ltd
Process, Technology and Training for Critical Software

- Problems with Volatile

  1. It is a blunt instrument – it preserves *all* reads and writes of an object, not just the “last one”, so some performance penalty…

  2. Compilers don’t always get it right anyway…

Protean Code ltd
Process, Technology and Training for Critical Software

Volatiles Are Miscompiled, and What to Do about It

Eric Eide
University of Utah, School of Computing
Salt Lake City, UT USA
eeide@cs.utah.edu

John Regehr
University of Utah, School of Computing
Salt Lake City, UT USA
regehr@cs.utah.edu

ABSTRACT

C's volatile qualifier is intended to provide a reliable link between operations at the source-code level and operations at the memory-system level. We tested thirteen production-quality C compilers and, for each, found situations in which the compiler generated incorrect code for accessing volatile variables. This result is disturbing because it implies that embedded software and operating systems—both typically coded in C, both being bases for many

as volatile, the C compiler must ensure that every use (read or write) of that location in the source program is realized by an appropriate memory operation (load or store) in the compiled program. Accesses to volatiles are considered to be side-effecting operations, and they are therefore part of the observable behavior of a program that must not be changed by an optimizing compiler. Embedded software commonly relies on volatile variables in order to access memory-mapped I/O ports, to communicate between concurrent

- Paper from EMSOFT 2008.
- Have compilers improved? Not sure…
- Are "commercial" compilers better than "open source" (e.g. GNAT Pro vs FSF GCC vs LLVM)? Don't know…

Limited types

Protean Code ltd
Process, Technology and Training for Critical Software

- A very useful mechanism in Ada…
  - No assignment by default. Good!
  - Passed By-Reference, so no copies. Good!

Protean Code ltd
Process, Technology and Training for Critical Software

By-Reference types

- Another useful mechanism in Ada, and useful where limited types are not appropriate.

- Some types are defined to be “By Reference”, so avoids copying of sensitive parameters, for example

    - Tagged (“OO”) types (RM 6.2(5))

    - Record with Volatile component (RM C.6(18)

Inspection_Point

Protean Code ltd
Process, Technology and Training for Critical Software

- Pragma added in Ada 95, but little used (or understood?)

- *Very* useful requirement in RM H 3.2(9):

  'The implementation is not allowed to perform "dead store elimination" on the last assignment to a variable prior to a point where the variable is inspectable. Thus an inspection point has the effect of an implicit read of each of its inspectable objects.'

- Exactly what we want! But…

Inspection_Point

Protean Code ltd
Process, Technology and Training for Critical Software

- How does it work?
- GCC sources gcc/ada/gcc-interface/trans.c, in function Pragma_to_gnu ()

```
tree gnu_expr = gnat_to_gnu (gnat_expr);
...
ASM_VOLATILE_P (gnu_expr) = 1;
```

- So … see concerns above over correct compilation of Volatile.

Pattern 1

Protean Code ltd
Process, Technology and Training for Critical Software

- Combining these ideas yields a pattern for a sanitized abstract data type:

```
package Sensitive is
   type T is limited private; -- so no assignment

   procedure Sanitize (X : out T);
   pragma No_Inline (Sanitize);
private
   type T is limited record -- so by-reference
      F : … -- and so on…
   end record;
end Sensitive;
```

Pattern 1

Protean Code ltd
Process, Technology and Training for Critical Software

- To allow for alternative implementations (i.e. for different targets/operating systems), the body of Sanitize is supplied as a separate subunit.
- For a ZFP/Bare-Metal target, we might write:

```
separate (Sensitive)
procedure Sanitize (X : out T) is
begin
      X.F := 0; -- or other valid value
      pragma Inspection_Point (X);
end Sanitize;
```

SPARK

Protean Code ltd
Process, Technology and Training for Critical Software

- In both SPARK 2005 and SPARK 2014, a sanitizing assignment is reported as “Ineffective” by information-flow analysis.
  - So ... expect this, and justify:

```
pragma Warnings (Off, "unused assignment",
                 Reason => "Sanitization");
T := 0;
pragma Inspection_Point (T);
```

SPARK

Protean Code ltd
Process, Technology and Training for Critical Software

- What about proof?
- For non-limited types, we could declare a constant for the “sanitized value”, and use that in post-conditions and/or assertions.
    - Note, though, that the value of the constant must be valid, so a value with representation 2#0000_0000_…# might not be OK.
- For limited types, we could declare a Boolean-valued “Is_Sanitized” function, thus:

SPARK

Protean Code ltd
Process, Technology and Training for Critical Software

```
package Sensitive is
   type T is limited private; -- so no assignment

   function Is_Sanitized (X : in T) return Boolean
      with Ghost; -- no implementation

   procedure Sanitize (X : out T)
      with Post => Is_Sanitized (X);
   pragma No_Inline (Sanitize);

private
   -- As before…
end Sensitive;
```

Contents

- The problem…
- Technical issues
- Design goals
- Language support
- **A policy for sanitization**
- Related and further work

Policy

- Identification and Naming
  - Project must clearly define what is “sensitive”.
    - Consider global and local variables carefully…
    - May also depend on physical characteristics of memory (e.g. Stack might be in “secure RAM”, but global data isn’t…)

Policy

- Identification and Naming
  - Sensitive *constants* are not permitted.
  - Define a naming convention for sensitive types, variables and formal parameters.
  - Choose convention to facilitate automated search of compiler and tool output.

Policy

Protean Code ltd
Process, Technology and Training for Critical Software

- Types and Patterns
  - Use by-reference types for sensitive data.
  - Use limited types as per Pattern 1 above where possible.
  - Use pragma Warnings to suppress SPARK flow error.

Contents

Protean Code ltd
Process, Technology and Training for Critical Software

- The problem…
- Technical issues
- Design goals
- Language support
- A policy for sanitization
- **Related and further work**

Protean Code ltd
Process, Technology and Training for Critical Software

Related and Further Work (1)

- Special compiler switch to automatically sanitize local data?
  - "-ferase-stack" perhaps?
- Actually, this has already been done at least twice:
  - Laurent Simon at Cambridge – see his paper "Erasing Secrets from RAM" from RWC 2017
  - The team at www.embecosm.com in LLVM.
- Will it work with Ada?

Related and Further Work (2)

- What about a new language-defined Aspect?

```
Key : Word32 with Sensitive; -- ???
```

- Then compiler takes care of it?

Protean Code ltd
Process, Technology and Training for Critical Software

Related and Further Work (3)

- What is the impact of Link-Time Optimization (LTO)?
  - No idea…
- Can we use information-flow analysis to track values derived from sensitive data?
  - Like “taint analysis” in other languages…

Protean Code ltd
Process, Technology and Training for Critical Software

Questions…