Libabigail & ABI compatibility

Taming the runtime linking problem

Ben Woodard Consulting Engineer
Dodji Seketeli Tools Engineer
Aug 2017

Review: Problems due to ABI Compatibility

Reminder: We are talking about ABI not API

API directly affects this (with C++ templates the API is the ABI).

There are several problems caused by ABI compatibility problems either real or feared.

- Linking objects compiled by different compilers not recommended
  - Unforeseen problem at runtime
  - Need to compile libs for all compilers
- Library Version problems
  - Match library version with application
  - Match compiler

Review: Libabigail

Libabigail reads ELF, DWARF, & XML to build an IR of the ABI

- ELF, DWARF -> IR of ABI -> can be saved as XML
  - Other formats possible e.g. MacO or Windows
  - Not just DWARF to DWARF comparison
    - Comparison between compilers
    - Saved ABI XML represents all ABI elements
  - Compare saved ABI to ELF, DWARF for new object
    - Used to maintain ABI stability
    - No need to keep old objects around.

Libabigail what’s new:

- Kernels - we call this KABI
- abipkgdiff/fedabipkgdiff
    - Used in production
    - fedabipkgdiff - should be easily adaptable to other build systems
    - Fedora dist.abicheck - CI for the distro
- Hundreds of bug fixes
    - Kernels pushed C ABI hard
    - Fedora usage provided extensive testing

Runtime Linking Problems

Linking objects compiled by different compilers can sometimes cause unexplained problems. Why?

- ABI is supposed to make this work properly.
  - C works
    - C is very old & simple
    - Otherwise system libraries wouldn't work.
    - Mostly API problems and no symbol versioning
  - C++
    - Standard based on ia64 work with updates
    - Willing to stand behind and support Clang & GCC compatibility.
    - RHEL7 has dual ABI pre/post C++11
  - FORTRAN
    - Unknown

Dual ABI

The evilness that lingers....

Mostly a problem for RHEL7 where the system compiler is pre-gcc 5.1

Different than language standard i.e. -std=c++14

Libstdc++ supports dual-ABI on RHEL7 with DTS compilers.

General rule of thumb: recompile all C++ libraries with C++11

ABI tags

- Uses inline namespaces to encode ABI tags within the mangled names.
- Not just useful for libstdc++
- To help detect problems -Wabi-tag

DWARF

- Not prescriptive
- Still evolving - mostly to deal with C++ challenges
- Inconsistency leads to false positives
    - Biggest problem with C++
    - Inconsistent names
- Weaknesses leads to false negatives
- Not normalized
    - e.g. spaces introduced in type names
- Quality is still a problem with some compilers

Today

1. The corpus is much more complete this year. Using libabigail to detect changes in API and ABI changes between versions of the library should work as long as they have DWARF and are compiled with some version of the same compiler.
2. GCC and Clang both have conforming ABIs (to the best of our knowledge they are compatible). Libabigail is likely to report lots of false positives for ABI changes particularly for generic programming.
3. Other compilers DWARF is not as good and this leads to false negatives and a few false positives.

Templates are hard

Example:

```
#include <string>
#include <vector>

std::vector<std::string> var;
```

Templates are hard

```
dodji@adjoa:gcc-llvm$ ../../build/tools/abidiff test3-gcc.o test3-clang.o
Functions changes summary: 0 Removed, 0 Changed, 0 Added function
Variables changes summary: 0 Removed, 1 Changed, 0 Added variable
Function symbols changes summary: 11 Removed, 1 Added function symbols not referenced by debug info
Variable symbols changes summary: 0 Removed, 0 Added variable symbol not referenced by debug info

1 Changed variable:

  [C]'std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
std::allocator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > var' was changed to
'std::vector<std::__cxx11::basic_string<char>, std::allocator<std::__cxx11::basic_string<char> > > var' at test3.C:4:1:
    type of variable changed:
     type name changed from 'std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
std::allocator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > >' to
'std::vector<std::__cxx11::basic_string<char>, std::allocator<std::__cxx11::basic_string<char> > >'
     type size hasn't changed

     1 base class deletion:
       struct std::_Vector_base<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
std::allocator<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > at stl_vector.h:74:1
     1 base class insertion:
       struct std::_Vector_base<std::__cxx11::basic_string<char>, std::allocator<std::__cxx11::basic_string<char> > > at
stl_vector.h:74:1
```

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos