# ABI Verification Overview

Dodji Seketeli <dodji@redhat.com>

GNU Cauldron - July 2013 - Moutain View, USA

# Outline

# Now let's talk about ...

Problem Statement

Existing stuff

What we are proposing

Current Status

TODO

# Background

- Suppose we have a program P that links dynamically to a library L(V) which version number is V.

# Background

- Suppose we have a program P that links dynamically to a library L(V) which version number is V.
- Question: Will P keep linking with L(V+1) *safely*?

# Background

- Suppose we have a program P that links dynamically to a library L(V) which version number is V.
- Question: Will P keep linking with L(V+1) *safely*?
- We want a system that answers that question by taking L(V) and L(V+1) as inputs.

# Background

- Suppose we have a program P that links dynamically to a library L(V) which version number is V.
- Question: Will P keep linking with L(V+1) *safely*?
- We want a system that answers that question by taking L(V) and L(V+1) as inputs.
- A different question of the same kind:

# Background

- Suppose we have a program P that links dynamically to a library L(V) which version number is V.
- Question: Will P keep linking with L(V+1) *safely*?
- We want a system that answers that question by taking L(V) and L(V+1) as inputs.
- A different question of the same kind:
  - A program P loads a DSO D. Are the symbols of P and D *compatible*?

# Background

- Suppose we have a program P that links dynamically to a library L(V) which version number is V.
- Question: Will P keep linking with L(V+1) *safely*?
- We want a system that answers that question by taking L(V) and L(V+1) as inputs.
- A different question of the same kind:
  - A program P loads a DSO D. Are the symbols of P and D *compatible*?
  - More of this type of questions anyone? Take your chance!

# Now let's talk about ...

Problem Statement

Existing stuff

What we are proposing

Current Status

TODO

# Existing approaches

- Get types and symbols information from the binary and its headers *somehow*.

# Existing approaches

- Get types and symbols information from the binary and its headers *somehow*.
  - Existing tools use DWARF to get the symbol and type information from binaries.

# Existing approaches

- Get types and symbols information from the binary and its headers *somehow*.
  - Existing tools use DWARF to get the symbol and type information from binaries.
  - Parse headers to get the information to be consumed *inline* by client code:

# Existing approaches

- Get types and symbols information from the binary and its headers *somehow*.
  - Existing tools use DWARF to get the symbol and type information from binaries.
  - Parse headers to get the information to be consumed *inline* by client code:
    - Small inline functions.

# Existing approaches

- Get types and symbols information from the binary and its headers *somehow*.
  - Existing tools use DWARF to get the symbol and type information from binaries.
  - Parse headers to get the information to be consumed *inline* by client code:
    - Small inline functions.
    - Templates present in headers but not instantiated in the binary.

# Existing approaches

- Get types and symbols information from the binary and its headers *somehow*.
  - Existing tools use DWARF to get the symbol and type information from binaries.
  - Parse headers to get the information to be consumed *inline* by client code:
    - Small inline functions.
    - Templates present in headers but not instantiated in the binary.
  - Parsing the inline information is *tricky* without compiler support. Template anyone?

# Well, okay, but what does the user actually get?

- Nice tool around: ABI Compliance Checker

# Well, okay, but what does the user actually get?

- Nice tool around: ABI Compliance Checker
- Suppose we want to check ABI compatibility for two versions of the mysql++ library (3.0.9 and 3.0.10)

# Well, okay, but what does the user actually get?

- Nice tool around: ABI Compliance Checker
- Suppose we want to check ABI compatibility for two versions of the mysql++ library (3.0.9 and 3.0.10)
- First, write a descriptor of the library.

`cat mysql++309.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<descriptor>
<version>
    3.0.9
</version>
<headers>
    /usr/include/mysql++/
</headers>
<libs>
    /usr/lib64/libmysqlpp.so
</libs>
<include_paths>
    /usr/include/mysql
</include_paths>
...
```

# Well, okay, but what does the user actually get?

- Write a similar mysql++3010.xml descriptor.

# Well, okay, but what does the user actually get?

- Write a similar mysql++3010.xml descriptor.
- Call the tool:

```
abi-compliance-checker -l libssh -old mysql++309.xml -new mysql++3010.xml
```

# Well, okay, but what does the user actually get?

- Write a similar mysql++3010.xml descriptor.
- Call the tool:

```
abi-compliance-checker -l libssh -old mysql++309.xml -new mysql++3010.xml
```

- Get A Nice Report!

http://blog.famillecollet.com/public/reports/mysql__-3.0.9-3.1.0.html

# Well, okay, but what does the user actually get?

- Write a similar mysql++3010.xml descriptor.
- Call the tool:

```
abi-compliance-checker -l libssh -old mysql++309.xml -new mysql++3010.xml
```

- Get A Nice Report!

http://blog.famillecollet.com/public/reports/mysql__-3.0.9-3.1.0.html

- But but, what if we have some breaking changes in some, say, inline un-instantiated member function template of a class?

# Well, okay, but what does the user actually get?

- Write a similar mysql++3010.xml descriptor.
- Call the tool:

```
abi-compliance-checker -l libssh -old mysql++309.xml -new mysql++3010.xml
```

- Get A Nice Report!

`http://blog.famillecollet.com/public/reports/mysql__-3.0.9-3.1.0.html`

- But but, what if we have some breaking changes in some, say, inline un-instantiated member function template of a class?
- Oops.

# Now let's talk about ...

Problem Statement

Existing stuff

What we are proposing

Current Status

TODO

# Our Proposal

- Have GCC emit the abi dump at compile time, package it along with the binary, and have a tool consume that abi dump.

# Our Proposal

- Have GCC emit the abi dump at compile time, package it along with the binary, and have a tool consume that abi dump.
- Devise a reusable library to emit and consume the dump format (libabigail).

# Our Proposal

- Have GCC emit the abi dump at compile time, package it along with the binary, and have a tool consume that abi dump.
- Devise a reusable library to emit and consume the dump format (libabigail).
- Use that library from GCC to emit the dump

# Our Proposal

- Have GCC emit the abi dump at compile time, package it along with the binary, and have a tool consume that abi dump.
- Devise a reusable library to emit and consume the dump format (libabigail).
- Use that library from GCC to emit the dump
- Use that library from the tools to consume the dump and present it to users.

# Our Proposal

- Have GCC emit the abi dump at compile time, package it along with the binary, and have a tool consume that abi dump.
- Devise a reusable library to emit and consume the dump format (libabigail).
- Use that library from GCC to emit the dump
- Use that library from the tools to consume the dump and present it to users.
- Secret wish, re-use the library/tools from within abi-compliance-checker.pl. Perl, Ugh.

# Our Proposal

- Have GCC emit the abi dump at compile time, package it along with the binary, and have a tool consume that abi dump.
- Devise a reusable library to emit and consume the dump format (libabigail).
- Use that library from GCC to emit the dump
- Use that library from the tools to consume the dump and present it to users.
- Secret wish, re-use the library/tools from within abi-compliance-checker.pl. Perl, Ugh.
  - But then I touched some TCL/DejaGNU stuff already. Perl can't be much worse, can it?

- Code example:

```
$ cat test0.cc

enum mode : short { in, out, top, bottom };

void
foo(mode& t)
{
    mode u = t;
}
```

# Generate an ABI dump for it

- ./cc1plus -std=c++11 -quiet -fdump-abi test0.cc

# Generate an ABI dump for it

- ./cc1plus -std=c++11 -quiet -fdump-abi test0.cc
- test0.cc.bi

```
<abi-instr version='1.0'>
  <type-decl name='short int' size-in-bits='16' alignment-in-bits='16' id='type-id-1'/>
  <enum-decl name='mode' filepath='test0.cc' line='1' column='6' id='type-id-2'>
    <underlying-type type-id='type-id-1'/>
    <enumerator name='in' value='0'/>
    <enumerator name='out' value='1'/>
    <enumerator name='top' value='2'/>
    <enumerator name='bottom' value='3'/>
  </enum-decl>
  <reference-type-def kind='lvalue' type-id='type-id-2' size-in-bits='64'
                      alignment-in-bits='64' id='type-id-3'/>
  <type-decl name='void' alignment-in-bits='8' id='type-id-4'/>
  <function-decl name='foo' mangled-name='_Z3fooR4mode'
                 filepath='test0.cc' line='4' column='1'
                 visibility='default' binding='global'>
    <parameter type-id='type-id-3'/>
    <return type-id='type-id-4'/>
  </function-decl>
</abi-instr>
```

# How do we manipulate that file ?

- Example: loading the dump:

```
#include "abg-reader.h"

void
poke_at_function_template(shared_ptr<function_template_decl> t);

void
foo()
{
  using namespace abigail;

  translation_unit tu;

  if (!reader::read_file("test.0.cc", tu))
    error_out(); // Oops, there was an issue.

  translation_unit::decls_type::const_iterator i;

  // Walk the global declarations and poke at function templates
  for (i = tu.get_global_scope()->get_member_decls().begin();
       i != tu.get_global_scope()->get_member_decls().end())
    {
      shared_ptr<function_template_decl> tmpl =
        dynamic_pointer_cast<function_template>(*i);

      poke_at_function_template(tmpl);
    }
}
```

# Now let's talk about ...

Problem Statement

Existing stuff

What we are proposing

Current Status

TODO

# Where we are now

- The library can represent up to templates basics.

# Where we are now

- The library can represent up to templates basics.
- G++ uses that to emit representation of decls and types up to classes, with virtual functions and bases.

# Where we are now

- The library can represent up to templates basics.
- G++ uses that to emit representation of decls and types up to classes, with virtual functions and bases.
- There a basic DejaGNU support.

# Where we are now

- The library can represent up to templates basics.
- G++ uses that to emit representation of decls and types up to classes, with virtual functions and bases.
- There a basic DejaGNU support.
  - Example: New test gcc/testsuite/g++.dg/abi-dump/bases-0.C

# Now let's talk about ...

Problem Statement

Existing stuff

What we are proposing

Current Status

TODO

# TODO

- Support for binary instrumentation archives

# TODO

- Support for binary instrumentation archives
- Write the diffing support into the library.

# TODO

- Support for binary instrumentation archives
- Write the diffing support into the library.
- Add support for missing constructs

# TODO

- Support for binary instrumentation archives
- Write the diffing support into the library.
- Add support for missing constructs
  - Template expressions

# TODO

- Support for binary instrumentation archives
- Write the diffing support into the library.
- Add support for missing constructs
  - Template expressions
  - More stuff generally

# TODO

- Support for binary instrumentation archives
- Write the diffing support into the library.
- Add support for missing constructs
  - Template expressions
  - More stuff generally
- Publish the Library!!

# So ...

- Questions ?

# So ...

- Questions ?

- Thank You!