Linux
Plumbers
Conference 2022

>> Dublin, Ireland / September 12-14, 2022

ABI Graphs

ABI information: BTF

```
[1] INT 'int' size=4 bits_offset=0 nr_bits=32 encoding=SIGNED
[2] FUNC_PROTO '(anon)' ret_type_id=1 vlen=0
[3] FUNC_PROTO '(anon)' ret_type_id=1 vlen=0
[4] FUNC 'launch_missiles' type_id=2 linkage=static
[5] FUNC 'defcon' type_id=3 linkage=static
```

```
int defcon() { return 2; }
int launch_missiles() { return 0; }

gcc -gbtf -c lm.c
bpftool btf dump file lm.o format raw
```

ABI information: BTF

```
[1] INT 'int' size=4 bits_offset=0 nr_bits=32 encoding=SIGNED
[2] FUNC_PROTO '(anon)' ret_type_id=1 vlen=0
[3] FUNC_PROTO '(anon)' ret_type_id=1 vlen=0
[4] FUNC 'launch_missiles' type_id=2 linkage=static
[5] FUNC 'defcon' type_id=3 linkage=static
```

```
int defcon() { return 2; }
int launch_missiles() { return 0; }

gcc -gbtf -c lm.c
bpftool btf dump file lm.o format raw
```

ABI information: BTF

```
[1] INT 'int' size=4 bits_offset=0 nr_bits=32 encoding=SIGNED
[2] FUNC_PROTO '(anon)' ret_type_id=1 vlen=0
[3] FUNC_PROTO '(anon)' ret_type_id=1 vlen=0
[4] FUNC 'launch_missiles' type_id=2 linkage=static
[5] FUNC 'defcon' type_id=3 linkage=static
```

```
int defcon() { return 2; }
int launch_missiles() { return 0; }

gcc -gbtf -c lm.c
bpftool btf dump file lm.o format raw
```

ABI information: ELF & DWARF

```
0000000000000000 T defcon
000000000000000b T launch_missiles
```

```
gcc -g -c lm.c
nm lm.o
readelf --debug-dump lm.o
```

```
<1><2e>: Abbrev Number: 1 (DW_TAG_subprogram)
    <2f>   DW_AT_external    : 1
    <2f>   DW_AT_name        : (indirect string,
offset: 0x7): launch_missiles
    <34>   DW_AT_type        : <0x4a>
    <38>   DW_AT_low_pc      : 0xb
    <40>   DW_AT_high_pc     : 0xb
 <1><4a>: Abbrev Number: 3 (DW_TAG_base_type)
    <4b>   DW_AT_byte_size   : 4
    <4c>   DW_AT_encoding    : 5  (signed)
    <4d>   DW_AT_name        : int
 <1><51>: Abbrev Number: 1 (DW_TAG_subprogram)
    <52>   DW_AT_external    : 1
    <52>   DW_AT_name        : (indirect string,
offset: 0x0): defcon
    <57>   DW_AT_type        : <0x4a>
    <5b>   DW_AT_low_pc      : 0x0
    <63>   DW_AT_high_pc     : 0xb
```

ABI information: libabigail XML

```
gcc -g -c lm.c && abidw lm.o
```

```
<abi-corpus version='2.1' path='lm.o' architecture='elf-amd-x86_64'>
  <elf-function-symbols>
    <elf-symbol name='defcon' type='func-type' binding='global-binding' visibility='default-visibility'
is-defined='yes'/>
    <elf-symbol name='launch_missiles' type='func-type' binding='global-binding'
visibility='default-visibility' is-defined='yes'/>
  </elf-function-symbols>
  <abi-instr address-size='64' path='lm.c' language='LANG_C11'>
    <type-decl name='int' size-in-bits='32' id='95e97e5e'/>
    <function-decl name='defcon' mangled-name='defcon' visibility='default' binding='global'
size-in-bits='64' elf-symbol-id='defcon'>
      <return type-id='95e97e5e'/>
    </function-decl>
    <function-decl name='launch_missiles' mangled-name='launch_missiles' visibility='default'
binding='global' size-in-bits='64' elf-symbol-id='launch_missiles'>
      <return type-id='95e97e5e'/>
    </function-decl>
  </abi-instr>
</abi-corpus>
```

ABI Representations are Graphs

- ABI entities (nodes) include symbols, structs, function types and various other kinds of type.
- ABI relationships (edges) include: has-type, has-member, has-return-type and many more.

ABI Representations are Graphs

- ABI entities (nodes) include symbols, structs, function types and various other kinds of type.
- ABI relationships (edges) include: has-type, has-member, has-return-type and many more.

And so are ABI Comparisons

- Nodes are pairs of ABI entities.
- Edges are matching pairs of ABI relationships.

Why Graphs?

- some entities have no natural (stable) identifiers
  - entities don't live in a simple flat namespace
- topology and connectivity are significant
  - dependency chains
  - cyclic dependencies

What kind of graphs?

- directed, possibly cyclic graphs
- with labelled nodes
  - with distinctly labelled edges
- and a distinguished root node
  - that has an edge to each symbol

Proposition

- ABI representations and comparisons are graphs.
- A small number of graph algorithms and graph transformations can solve most of the difficult parts of ABI monitoring
- They can be independently researched, developed, tested and proven.

Graph Comparison Algorithm

- uses a **local** equivalence of node labels
- matches edge labels and follows them
- tracks connectivity
- obtains a **global** equivalence of nodes
- complexity is O(V + E)
- solves ABI equivalence checking and diff generation
- partly solves ABI graph deduplication

Graph Transformations

What if we want...?

- to restrict an ABI to a subset of symbols
- to resolve forward declarations to their definitions
- volatile const ≡ const volatile
- typedef int foo ≡ int
- void(const int) ≡ void(int)
- to support qualified typedef of array types

Other things we want...

- text format suitable for version control
  - minimal ABI changes result in minimal textual diffs
  - friendly to human readers and diff / merge tools
- natural language descriptions of nodes and diffs
- query expressions for diff selection / suppression
- comparison graph transformations?
- anything else?

Linux Plumbers Conference 2022

>> Dublin, Ireland / September 12-14, 2022

STG - Symbol-Type Graph

Done

- BTF → STG
- libabigail XML (including C++) → STG
- STG comparison algorithm
- STG diff reporting (multiple formats)

To do

- ELF & DWARF → STG
- STG transformations (including deduplication)
- STG ↔ text format suitable for version control

https://android.googlesource.com/platform/external/stg/