Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

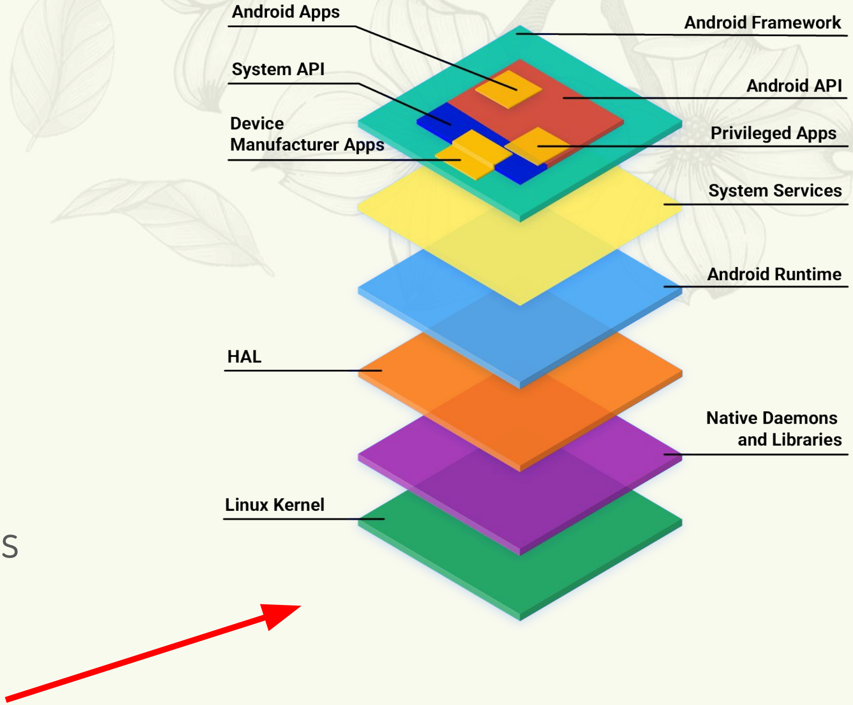
Graph-based ABI analysis for fun and profit

Matthias Männich <maennich@android.com> | Giuliano Procida <gprocida@android.com>

Who are we?

Android Systems Team @ Google

- Linux Kernel in Android
- Generic Kernel Image (GKI)
- Systems libraries and components

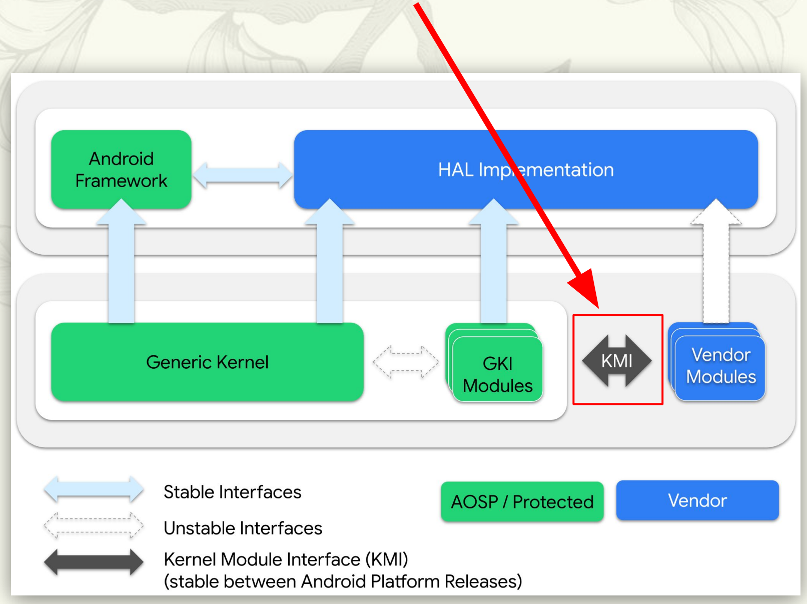


https://source.android.com/

Why do we care?

Android system software - safer upgrades

- Android (Common) Kernels
  - Generic Kernel Image (GKI) with Kernel-Module-Interface (KMI)
  - ABI stability guarantees within releases
- Userspace Libraries
  - API versions
  - Android NDK



Mission: Capture exposed ABI / API surfaces and detect changes at build time.

https://source.android.com/docs/core/architecture/kernel
https://source.android.com/docs/core/architecture/kernel/abi-monitor

What do we mean by an ABI?

ABI stands for "application binary interface". **ABI compatibility** means that software components can work together at runtime – link and type compatibility

- library + binary
- kernel + module

ABI representations capture enough symbol and type information to be able to determine compatibility.

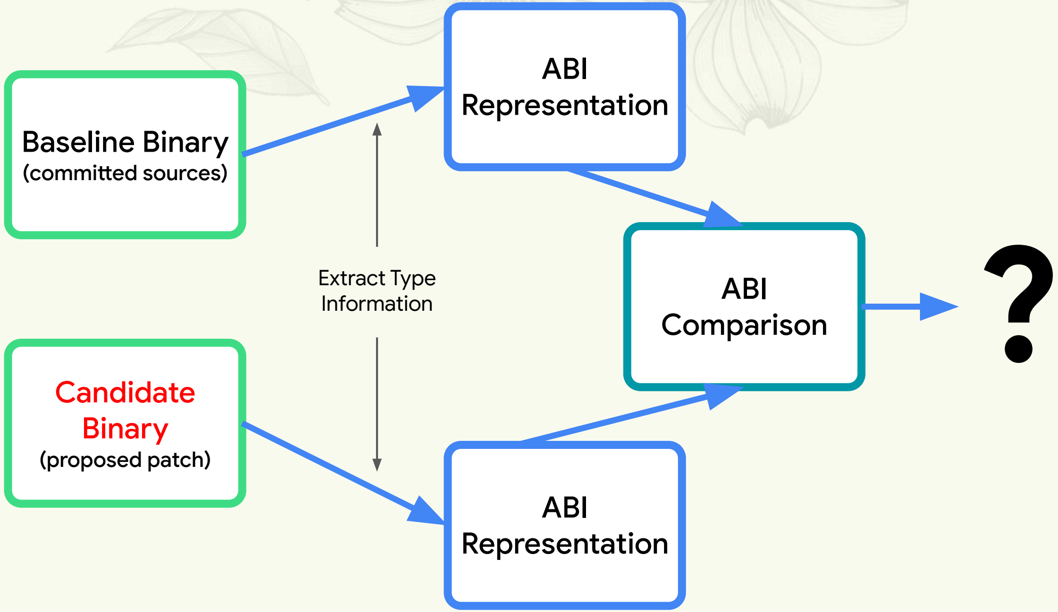
Other aspects of ABI compatibility may or may not be represented:

- machine type
- ISA
- calling convention

ABI Stability Monitoring through Static Analysis

Comparison of Binaries against ABI baseline

1. Extraction of an ABI representation from ELF +
   - ○ DWARF
   - ○ CTF
   - ○ BTF
   - ○ ?
2. Comparison
3. Reporting



Prior Art - libabigail

libabigail (https://sourceware.org/libabigail/)

- de facto ABI monitoring tool
- open source
- actively maintained
- broad compiler, language and package support
- packaged for RedHat, Debian, ...

Tools

- **`abidw`** - ABI extraction - used until Android 13
- **`abidiff`** - ABI comparison - used until Android 12
- and others (e.g. package comparison)

History: BTF and btdiff

BTF is the eBPF Type Format

- alternative to DWARF for type information
- much smaller and potentially shippable in production binaries (e.g. 124MB DWARF vs 1.5MB BTF)
- limited non-GCC toolchain support

btfdiff - intern project to explore using BTF

- BTF reader, building a graph
- comparison algorithm with memoisation
  - incomplete handling of graph cycles
  - in-line diff reporting
- memoised type name generation

https://facebookmicrosites.github.io/bpf/blog/2018/11/14/btf-enhancement.html
https://docs.kernel.org/bpf/btf.html#bpf-type-format-btf
https://android.googlesource.com/platform/external/stg/+/refs/heads/main/btf_reader.cc

History: Cycle-Aware Graph Comparison Algorithm

ABI comparison is graph comparison, traversing an abstract comparison graph whose nodes are pairs of concrete nodes.

Requirements

- be easily extensible
- efficiently handle repeated comparisons
- correctly handle cycles

The development of this required research into classic graph algorithms and experimentation with much simpler graphs.

In a comparison graph SCC (strongly connected component), differences between any node pair imply differences between all pairs.

Implementation

- simple recursive comparison
- with memoisation
- and SCC handling

SCC detection and the comparison logic must happen simultaneously to avoid actually building a concrete comparison graph.

The path-based strong component algorithm is a DFS plus some state management logic.

STG's SCC finder encapsulates this state and logic and can be added to any recursive algorithm which might encounter graph cycles.

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/doc/SCC.md

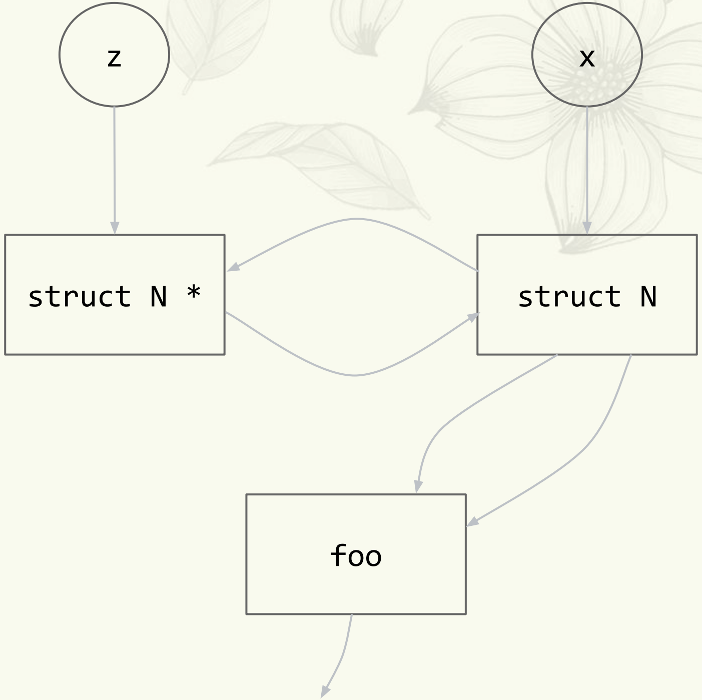
Linux Plumbers Conference | Richmond, VA | Nov. 13-15, 2023

History: Cycle-Aware Graph Comparison Algorithm

```
typedef … foo;

struct N {
  struct N * next;
  foo left;
  foo right;
};

struct N x;
struct N * z;
```



How to compare?

Cycles?

Result caching?

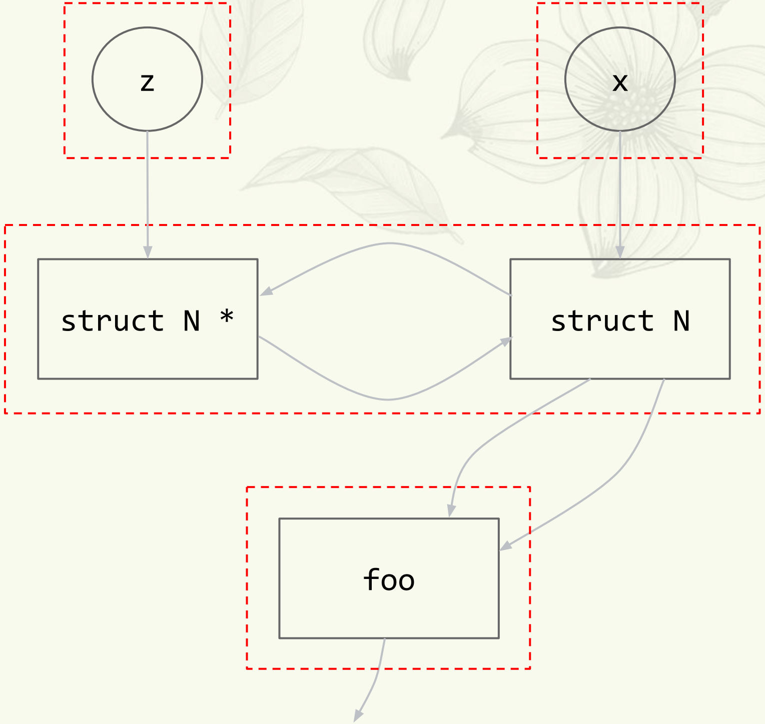
https://android.googlesource.com/platform/external/stg/+/refs/heads/main/doc/SCC.md

History: Cycle-Aware Graph Comparison Algorithm

```
typedef … foo;

struct N {
  struct N * next;
  foo left;
  foo right;
};

struct N x;
struct N * z;
```



Strongly Connected

Components (SCC)!

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/doc/SCC.md

History: btfdiff → stgdiff

From research vehicle to Android Common Kernel ABI monitoring

- separate diff graph creation and report generation
- multiple report formats, including:
  - flat: break the diff graph into smaller subgraphs
  - small: flat + omit uninteresting subgraphs
- libabigail XML reader
  - `abidw` + `stgdiff` = ABI monitoring solution
- CLI supports all combinations of input and output formats

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/doc/DIFFS.md

History: STG (Symbol Type Graph)

- C++17 code base
- Hosted within AOSP (Android Open Source Project)
- Open Source / Apache2 + LLVM Exception
- Built with CMake, standalone prebuilts available

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

diff example - changed node distance from root node ~8

```
@@ -1,13 +1,13 @@
 struct A {
-  int x;
+  unsigned int x;
 };

 struct B {
   struct A a;
 };

 struct C {
   struct B b;
 };

 struct C c;
```

diff format: **plain** – gives full context but does not scale

```
@@ -1,13 +1,13 @@
 struct A {
-  int x;
+  unsigned int x;
 };

 struct B {
   struct A a;
 };

 struct C {
   struct B b;
 };

 struct C c;
```

```
variable symbol 'struct C c' changed
  type 'struct C' changed
    member 'struct B b' changed
      type 'struct B' changed
        member 'struct A a' changed
          type 'struct A' changed
            member changed from 'int x' to 'unsigned int x'
              type changed from 'int' to 'unsigned int'
```

diff format: **flat** – chops up the diff graph

```
@@ -1,13 +1,13 @@
 struct A {
-  int x;
+  unsigned int x;
 };

 struct B {
   struct A a;
 };

 struct C {
   struct B b;
 };

 struct C c;
```

```
variable symbol 'struct C c' changed
  type 'struct C' changed

type 'struct C' changed
  member 'struct B b' changed
    type 'struct B' changed

type 'struct B' changed
  member 'struct A a' changed
    type 'struct A' changed

type 'struct A' changed
  member changed from 'int x' to 'unsigned int x'
    type changed from 'int' to 'unsigned int'
```

diff format: **small** – discards chunks with no internal differences

```
@@ -1,13 +1,13 @@
 struct A {
-  int x;
+  unsigned int x;
 };

 struct B {
   struct A a;
 };

 struct C {
   struct B b;
 };

 struct C c;
```

```
type 'struct A' changed
  member changed from 'int x' to 'unsigned int x'
    type changed from 'int' to 'unsigned int'
```

History: C++ Support for Userspace ABI Monitoring

intern project to **model C++ type system** features

- aim: model C++ ABIs, read full libabigail XML syntax
- design decisions
  - access specifiers (public, protected, private) are not modelled
  - methods modelled with a new kind of node
  - references modelled by tweaking pointer nodes
  - ...
- parsing, comparison and reporting code changes
  - with full test coverage

History: native format

Requirements

- version control friendly
  - text
  - minimal graph changes cause minimal representation changes
  - rebase and merge usually work
- fast to read and write
- debuggable without special tools
- extensible

Implementation

- protocol buffer definition
  - corresponding to the graph model
  - heavily tested technology
  - wide support
- standard protobuf text format
  - with minor output tweaks
- stable output
  - external IDs generated as stable hashes
  - deterministic node order
- versioned
  - reader supports older versions

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/proto_writer.cc

Example

```c
struct A {
  int x;
};

struct B {
  struct A a;
};

struct C {
  struct B b;
};

struct C c;
```

```
version: 2
root_id: 0x84ea5130

primitive {
  id: 0x6720d32f
  name: "int"
  encoding: SIGNED_INTEGER
  bytesize: 4
}

member {
  id: 0x801a8063
  name: "a"
  type_id: 0xc1147dbd
}

member {
  id: 0x4cb80257
  name: "b"
  type_id: 0x207acb9f
}

member {
  id: 0xa0d54b05
  name: "x"
  type_id: 0x6720d32f
}
```

```
struct_union {
  id: 0xc1147dbd
  kind: STRUCT
  name: "A"
  definition {
    bytesize: 4
    member_id: 0xa0d54b05
  }
}
struct_union {
  id: 0x207acb9f
  kind: STRUCT
  name: "B"
  definition {
    bytesize: 4
    member_id: 0x801a8063
  }
}
struct_union {
  id: 0xc0318865
  kind: STRUCT
  name: "C"
  definition {
    bytesize: 4
    member_id: 0x4cb80257
  }
}
```

```
elf_symbol {
  id: 0x2230fb28
  name: "c"
  is_defined: true
  symbol_type: OBJECT
  type_id: 0xc0318865
  full_name: "c"
}

interface {
  id: 0x84ea5130
  symbol_id: 0x2230fb28
}
```

History: `stg`

New driver for ABI extraction

- reads any supported input
  - BTF, XML, ELF/DWARF, STG
- merges multiple inputs with type unification
  - used to obtain a single ABI from vmlinux + *.ko
- optionally applies a symbol filter
- deduplicates nodes by identifying equal subgraphs and rewriting the graph
- optionally outputs the resulting graph in the native format

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/doc/stg.md

History: ELF / DWARF reader

Support for:

- Clang-compiled Linux kernel and modules
- ELF: ksymtab, symbol CRCs and namespaces
- DWARF: versions 4 and 5, C language
- DWARF: C++ and GCC added incrementally

With:

- type unification, type normalisation, graph rewriting and more

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/dwarf_processor.cc

G is for Graph – the STG data structure

Aims:

- clear semantics
- maintainability
- high performance

Achieved via:

- simplicity
- separation of code and data
- powerful abstractions
- carefully-selected concrete data structures

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/graph.h

Graph Nodes - and edges

Nodes

- are identified by a numerical ID
- have attributes (name, size, ...)
- refer to other nodes (type, ...) by ID

And that's it!

- nodes are boring values
- no inheritance or methods
- no parent-child containment relationships

Concrete graph representation (currently)

- a vector per node kind
- an indirection vector (ID → (kind, offset))
- all hidden behind access abstractions

Node Kinds

- Special
- Pointer / Reference
- Pointer to Member
- Typedef
- Qualified
- Primitive
- Array
- Base Class
- Method
- Member
- Struct / Union
- Enumeration
- Function
- Elf Symbol
- Interface

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/graph.h

Graph Algorithms - node kind polymorphic functions

Naive recursive algorithms

- straightforward to write
- C++ function objects
- node kind overloaded function application
- arbitrary internal state

Examples:

- native format writer
- stable ID generation for the native format
- memoised type name generation for diff reporting

Cycle-aware recursive algorithms

- relatively easy to write
- just add:
  - SCC finder object
  - calls to SCC open / close
  - pending / completed node handling

Examples:

- efficient ABI difference graph generation
- memoised equality / inequality check
- node fingerprinting (bucketing) for faster deduplication

stg: Usage Examples (simple workflow)

```
# Compile a source file with debugging information
$ cc -c -g test.c -o test.o


# Extract ABI representation
$ stg --elf test.o -o test.stg


# Compare against baseline
$ stgdiff -s test.stg expected.stg -o -
```

https://android.googlesource.com/platform/external/stg/+/refs/heads/main/doc/stg.md

Performance

Some things already mentioned, plus:

- concrete graph representation tuned for space and time - 10 variations tested
- *jemalloc* or *tcmalloc* are very cheap constant-factor wins for large hash tables
- very simple dense data structures represent node sets and node mappings
- range reduction optimisation to avoid quadratic time costs processing multiple inputs
- fingerprinting optimisation to avoid quadratic time costs during node deduplication

Some numbers:

- vmlinux ~18M DIEs converted to ~12M STG nodes and reduced to ~45k STG nodes in ~18s
- → DWARF reader (including type unification and deduplication) ~1M DIEs/s
- vmlinux ABI STG representation read x2 ~175ms, comparison ~33ms
- → graph read ~0.5M nodes/s
- → graph comparison (excluding report serialisation) ~1.37M pairs/s

Demo

Where are we going?

Features

- more control over type definitions
  - implementation types can leak into ABIs
- named type filtering
- macro definition model
- improved CLI ergonomics (`stgdiff` → `stg diff`)

Improved DWARF Support

- more compiler / language / DWARF versions
- Support for different ABI models (e.g. Rust)
- backed by a comprehensive test suite

Test Suite Publication

- extraction and comparison tests

Other Inputs

- CTFv3
- archives (.rpm, .deb etc.)
- Compiler-generated representation targeting ABI inspection (e.g. CTF, but with support for more languages (like C++) and compilers (like Clang)) ?

Packaging STG for Distributions

- ArchLinux - done (AUR)
- Debian
- Fedora / RedHat ...
- SuSE
- ...

Resources

STG

- Source code: https://android.googlesource.com/platform/external/stg/
- LPC 2022 > Android MC > ABI Graphs: https://lpc.events/event/16/contributions/1335/
- Contact: kernel-team@android.com

Languages and Type Systems

- C and C++: https://cppreference.com/

Specifications

- ABI: https://itanium-cxx-abi.github.io/cxx-abi/abi.html
- BTF: https://www.kernel.org/doc/html/latest/bpf/btf.html
- CTF: https://github.com/oracle/binutils-gdb/wiki
- DWARF: https://dwarfstd.org/
- ELF: https://refspecs.linuxfoundation.org/elf/elf.pdf

Graph-based ABI analysis for fun and profit

Questions?

Matthias Männich <maennich@android.com> | Giuliano Procida <gprocida@android.com>