# Monitoring and Stabilizing the In-Kernel ABI

Linux Plumbers 2019

Matthias Männich



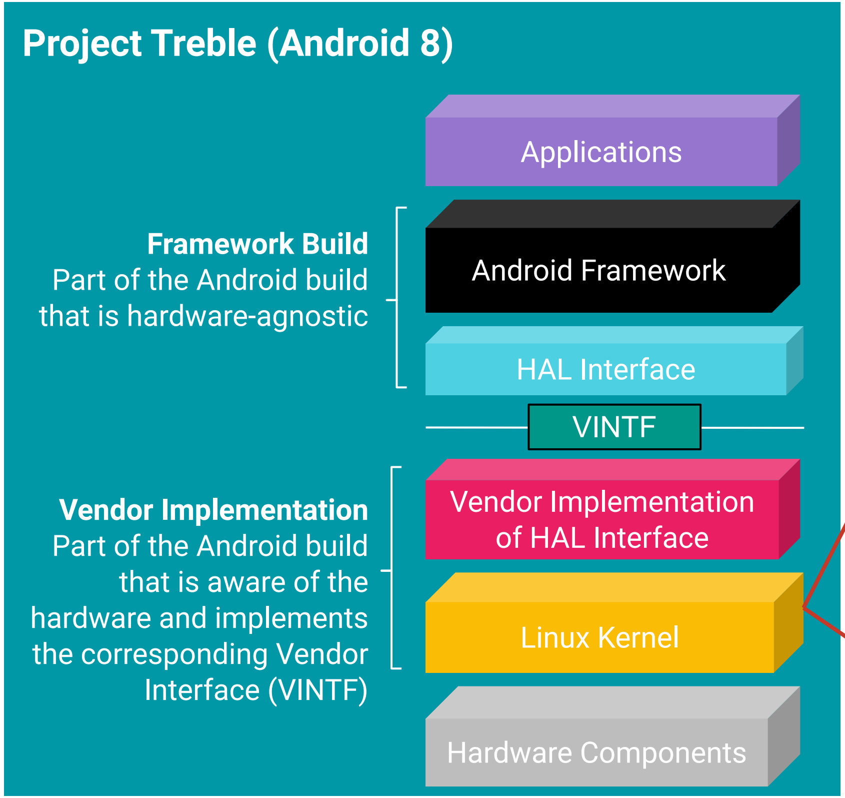
android

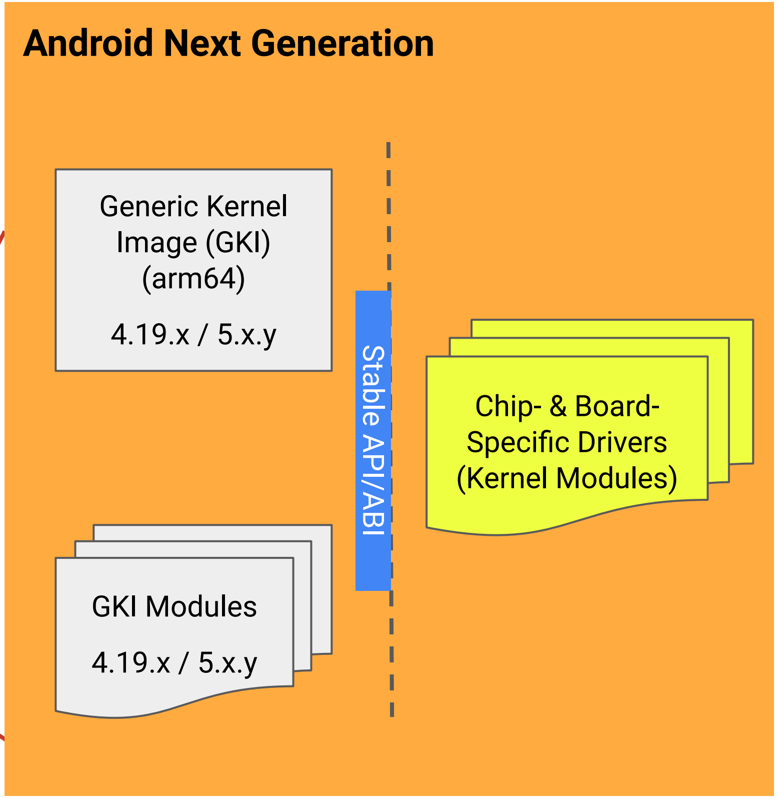
# Why stabilize the in-kernel ABI at all?

**Or: Why ignore** *Documentation/process/**stable-api-nonsense**.rst* ?

- Decouple development of the Kernel and its Modules
- Provide a single Kernel ABI / API for ecosystem of vendor modules
- Reduce fragmentation (multiple Kernel versions per Android version, one Kernel per device)

# Stable ABIs for Android Kernels



# Stable ABI within Boundaries

## and how Android implements that*

| Branches | Configuration | Toolchain | Scope |
| --- | --- | --- | --- |
| ○ Only keep ABI stable within major upstream branch<br>○ E.g.<br>LTS 4.9, 4.14, 4.19, 5.x | ○ Single Kernel Configuration<br>○ Suitable for all vendors<br>○ Configuration changes allowed if they don't break ABI | ○ Single Toolchain<br>○ Hermetic Build | ○ Define what is part of the ABI<br>○ Whitelist<br>○ Suppression |
| ○ android-4.19<br>○ android-5.x * | ○ Generic Kernel Image (GKI) configuration | ○ Clang Build (only)<br>○ Hermetic Toolchain*<br>(enforced by build wrapper) | ○ Observable ABI<br>○ Whitelists *<br>○ **Symbol Namespaces** * |

* some of these are still work in progress and to be finalized / implemented

# Integration into the Android Kernel Build

| build_abi.sh | | |
|---|---|---|
| build.sh | | ABI Tooling |
| ○ Setup (hermetic) build environment<br>  ○ Toolchain<br>  ○ Cross Compiler<br>  ○ Build Dir / Dist Dir | ○ `make mrproper`<br>○ `make <defconfig>`<br>○ `make`<br><br>○ create Android Kernel distribution | ○ `abidw --linux-tree out/`<br><br>○ `abidiff abi.xml abi-base.xml`<br><br>○ create abi report |

```
$ repo init -u <url> -b <branch>     # initialize workspace

$ repo sync                          # get sources, toolchain, dependencies, etc.

$ build/build_abi.sh                 # build and validate ABI
```

# Monitoring and Enforcement

- Define a baseline ABI
- Keep it along with your sources
- Establish ABI checking (e.g. `build_abi.sh`) as mandatory test before merging
- Changes targeting Android Common Kernels have to pass this test in AOSP Gerrit

```
--- a/include/linux/utsname.h
+++ b/include/linux/utsname.h
@@ -22,6 +22,7 @@ struct user_namespace;
 extern struct user_namespace init_user_ns;

 struct uts_namespace {
+        int dummy;
         struct kref kref;
         struct new_utsname name;
         struct user_namespace *user_ns;
```

M include/linux/utsname.h +1 -0

+36368 -36143

**Presubmit Lint Checks**

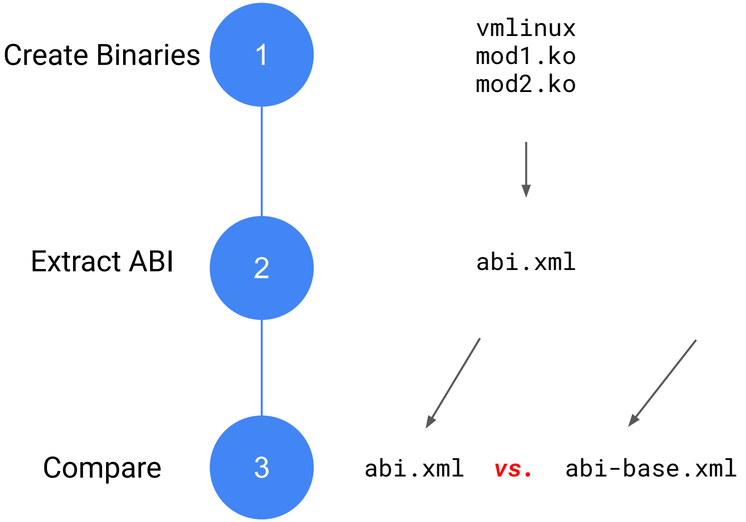
| Status | Category |
| --- | --- |
| | CommonTypos |
| | KernelCommit |
| | KernelABI |

| Line | Col | Filepath | Message | Subcategory |
| --- | --- | --- | --- | --- |
| | | | ABIs broken! : rc=4, output=Leaf changes summary: 3 artifacts changed<br>Changed leaf types summary: 1 leaf type changed<br>Removed/Changed/Added functions summary: 0 Removed, 1 Changed, 0 Added function<br>Removed/Changed/Added variables summary: 0 Removed, 0 Changed, 0 Added variable<br><br>1 function with some sub-type change:<br><br>'struct uts_namespace at utsname.h:24:1' changed:<br>type size hasn't changed<br>1 data member insertion:<br>'int uts_namespace::dummy', at offset 0 (in bits) at utsname.h:25:1<br>there are data member changes:<br>'kref uts_namespace::kref' offset changed from 0 to 32 (in bits) (by +32 bits)<br>'new_utsname uts_namespace::name' offset changed from 32 to 64 (in bits) (by +32 bits)<br><br>6244 impacted interfaces:<br>Qdisc_ops bfifo_qdisc_ops | |

# Libabigail

*"Application Binary Interface Generic Analysis and Instrumentation Library"*
[https://sourceware.org/libabigail/](https://sourceware.org/libabigail/)

- Library and set of tools to analyze ABIs of binaries
- Allows serializing, deserializing and comparing of ABI representation
- Considers ELF symbols along with DWARF information
- Linux Kernel Support is fairly new, but works pretty good (considers ***ksymtab*** instead of ELF symbol table )
- Support for 4.19+ Kernels is almost completed





android

# Libabigail

## ABI Representation

```c
struct my_type {
  char str[1024];
} my_type;

struct my_struct {
  int a;
  struct my_type b;
} my_struct;

enum my_enum {
  A, C, B
};

void func_str(struct my_struct   s) {}
void func_ptr(struct my_struct * s) {}
void func_enu(enum   my_enum     e) {}
```

```xml
<abi-corpus path='test.o' architecture='elf-amd-x86_64'>

  <elf-function-symbols>
    <elf-symbol name='func_enu' type='func-type'/>
    <elf-symbol name='func_ptr' type='func-type'/>
    <elf-symbol name='func_str' type='func-type'/>
  </elf-function-symbols>
  <elf-variable-symbols>
    <elf-symbol name='my_struct' size='1028' type='object-type'/>
    <elf-symbol name='my_type' size='1024' type='object-type'/>
  </elf-variable-symbols>

[...]
```

# Libabigail

## ABI Representation

```c
struct my_type {
  char str[1024];
} my_type;

struct my_struct {
  int a;
  struct my_type b;
} my_struct;

enum my_enum {
  A, C, B
};

void func_str(struct my_struct   s) {}
void func_ptr(struct my_struct * s) {}
void func_enu(enum   my_enum     e) {}
```

```xml
[...]

  <abi-instr version='1.0' address-size='64' language='LANG_C99'>

    <type-decl name='void' id='type-id-1'/>

    <type-decl name='enum-int' size-in-bits='32' id='type-id-2'/>

    <enum-decl name='my_enum' line='10' column='1' id='type-id-3'>
      <underlying-type type-id='type-id-2'/>
      <enumerator name='A' value='0'/>
      <enumerator name='C' value='1'/>
      <enumerator name='B' value='2'/>
    </enum-decl>

    <function-decl name='func_enu' line='16' column='1'
                   size-in-bits='64' elf-symbol-id='func_enu'>
      <parameter type-id='type-id-3' name='e' line='16' column='1'/>
      <return type-id='type-id-1'/>  <!-- void ->
    </function-decl>

[...]
```



android

# Libabigail

## ABI Representation

```c
struct my_type {
  char str[1024];
} my_type;

struct my_struct {
  int a;
  struct my_type b;
} my_struct;

enum my_enum {
  A, C, B
};

void func_str(struct my_struct   s) {}
void func_ptr(struct my_struct * s) {}
void func_enu(enum   my_enum     e) {}
```

```xml
[...]
    <class-decl name='my_struct' size-in-bits='8224' is-struct='yes'
                  line='5' column='1' id='type-id-4'>
      <data-member layout-offset-in-bits='0'>
        <var-decl name='a' type-id='type-id-5' line='6' column='1'/>
      </data-member>
      <data-member layout-offset-in-bits='32'>
        <var-decl name='b' type-id='type-id-6' line='7' column='1'/>
      </data-member>
    </class-decl>
    <type-decl name='int' size-in-bits='32' id='type-id-5'/>
    <class-decl name='my_type' size-in-bits='8192' [...]</class-decl>


    <pointer-type-def type-id='type-id-4' size-in-bits='64' id='type-id-11'/>
    <function-decl name='func_ptr' line='15' column='1'
                   size-in-bits='64' elf-symbol-id='func_ptr'>
      <parameter type-id='type-id-11' name='s' line='15' column='1'/>
      <return type-id='type-id-1'/>
    </function-decl>

[...]
```

# Libabigail

## Typical Breakages -- Removed Function

```
struct my_type {
  char str[1024];
} my_type;

struct my_struct {
  int a;
  struct my_type b;
} my_struct;

enum my_enum {
  A, C, B
};

void func_str(struct my_struct   s) {}
void func_ptr(struct my_struct * s) {}
-void func_enu(enum   my_enum     e) {}
```

```
Functions changes summary: 1 Removed, 0 Changed, 0 Added function
Variables changes summary: 0 Removed, 0 Changed, 0 Added variable

1 Removed function:

  'function void func_enu(my_enum)'    {func_enu}
```

| | |
|---|---|
| ABI Breakage | yes |
| Mitigation | Add back the symbol (maybe forward if it was a rename). |
| Notes | |



android

# Libabigail

## Typical Breakages -- Added Function

```
struct my_type {
  char str[1024];
} my_type;

struct my_struct {
  int a;
  struct my_type b;
} my_struct;

enum my_enum {
  A, C, B
};

void func_str(struct my_struct   s) {}
void func_ptr(struct my_struct * s) {}
void func_enu(enum   my_enum     e) {}

+void new_func() {}
```

```
Functions changes summary: 0 Removed, 0 Changed, 1 Added function
Variables changes summary: 0 Removed, 0 Changed, 0 Added variable

1 Added function:

  'function void new_func()'    {new_func}
```

| | |
|---|---|
| ABI Breakage | yes, but ABI still compatible |
| Mitigation | n/a |
| Notes | `new_func()` becomes part of the ABI. An update of the ABI representation is required to catch future breakages of `new_func()`. |



android

# Libabigail

## Typical Breakages -- Struct member name change

```
 struct my_type {
   char str[1024];
 } my_type;

 struct my_struct {
   int a;
-  struct my_type b;
+  struct my_type c;
 } my_struct;

 enum my_enum {
   A, C, B
 };

 void func_str(struct my_struct   s) {}
 void func_ptr(struct my_struct * s) {}
 void func_enu(enum   my_enum     e) {}
```

```
Functions changes summary: 0 Removed, 0 Changed (2 filtered out),
                           0 Added functions
Variables changes summary: 0 Removed, 0 Changed, 0 Added variable
```

| | |
|---|---|
| ABI Breakage | no, but API breakage |
| Mitigation | From an ABI point of view this is sane. |
| Notes | The rename is detected, but **considered harmless**.<br><br>Most likely the rename is not essential for the intent of the actual patch. So, reverting the rename is probably a good idea. |



android

# Libabigail

## Typical Breakages -- Add struct member

```
struct my_type {
  char str[1024];
} my_type;

struct my_struct {
  int a;
  struct my_type b;
+ int new_member;
} my_struct;

enum my_enum {
  A, C, B
};

void func_str(struct my_struct   s) {}
void func_ptr(struct my_struct * s) {}
void func_enu(enum   my_enum     e) {}
```

```
Functions changes summary: 0 Removed, 2 Changed, 0 Added functions
Variables changes summary: 0 Removed, 0 Changed, 0 Added variable

2 functions with some indirect sub-type change:

  [C]'function void func_str(my_struct)' at test.c:15:1 has indirect sub-type changes:
    parameter 1 of type 'struct my_struct' has sub-type changes:
      type size changed from 8224 to 8256 (in bits)
      1 data member insertion:
        'int my_struct::new_member', at offset 8224 (in bits) at test.c:8:

  [C]'function void func_ptr(my_struct*)' at test.c:16:1 has indirect sub-type changes:
    parameter 1 of type 'my_struct*' has sub-type changes:
      in pointed to type 'struct my_struct' at test.c:5:1:
```

| | |
|---|---|
| ABI Breakage | yes, but strictly only for `func_str()` |
| Mitigation | Add padding to data structures potentially being affected. |
| Notes | `func_ptr()` does not actually suffer from an ABI breakage at that very definition. It might be affected when the parameter gets dereferenced or if its size is taken. Libabigail diagnoses this because it can. It would not if the full definition of `my_struct` would not be available. |



android

# Libabigail

## Typical Breakages -- Type change (with identical memory layout)

```
  struct my_type {
    char str[1024];
  } my_type;

  struct my_struct {
    int a;
-   struct my_type b;
+   char b[1024];
  } my_struct;

  enum my_enum {
    A, C, B
  };

  void func_str(struct my_struct   s) {}
  void func_ptr(struct my_struct * s) {}
  void func_enu(enum   my_enum     e) {}
```

```
Functions changes summary: 0 Removed, 2 Changed, 0 Added functions
Variables changes summary: 0 Removed, 0 Changed, 0 Added variable

2 functions with some indirect sub-type change:

  [C]'function void func_ptr(my_struct*)' at test.c:15:1 has indirect sub-type changes:
    parameter 1 of type 'my_struct*' has sub-type changes:
      in pointed to type 'struct my_struct' at test.c:5:1:
        type size hasn't changed
        1 data member change:
         type of 'my_type my_struct::b' changed:
           entity changed from 'struct my_type' to 'char[1024]'
[...]
```

| ABI Breakage | no, but API breakage |
|---|---|
| Mitigation | Just don't. |
| Notes | Again, this is not a strict ABI breakage, but still discovered as breakage. In this case, the change is not considered harmless. |



android

# Libabigail

## Typical Breakages -- Sort enum values

```
struct my_type {
  char str[1024];
} my_type;

struct my_struct {
  int a;
  struct my_type b;
} my_struct;

enum my_enum {
-  A, C, B
+  A, B, C
};

void func_str(struct my_struct   s) {}
void func_ptr(struct my_struct * s) {}
void func_enu(enum   my_enum     e) {}
```

```
Functions changes summary: 0 Removed, 1 Changed, 0 Added function
Variables changes summary: 0 Removed, 0 Changed, 0 Added variable

1 function with some indirect sub-type change:

  [C]'function void func_enu(my_enum)' at test.c:16:1 has indirect sub-type
changes:
    parameter 1 of type 'enum my_enum' has sub-type changes:
      type size hasn't changed
      2 enumerator changes:
        'my_enum::C' from value '1' to '2' at test.c:10:1
        'my_enum::B' from value '2' to '1' at test.c:10:1
```

| | |
|---|---|
| ABI Breakage | no, but API breakage |
| Mitigation | Just don't. |
| Notes | Again, this is not a strict ABI breakage, but still discovered as breakage. In this case, the change is not considered harmless. |

# Unhandled Cases

## Untagged enums

*include/linux/mm.h*

```
enum {
    REGION_INTERSECTS,
    REGION_DISJOINT,
    REGION_MIXED,
};

/* returns one of the above values */
int region_intersects(resource_size_t offset,
                      size_t size,
                      unsigned long flags,
                      unsigned long desc);
```

How to handle sorting of such an enum now?

```
enum {
-   REGION_INTERSECTS,
    REGION_DISJOINT,
+   REGION_INTERSECTS,
    REGION_MIXED,
};
```

Generate ABI capturing data structures
(code generation or compiler plugin)

```
enum abi_enum {
    abi_REGION_INTERSECTS = REGION_INTERSECTS,
    abi_REGION_DISJOINT = REGION_DISJOINT,
    abi_REGION_MIXED = REGION_MIXED,
};

void abi_func(enum abi_enum e) { }
```

Captured as ELF symbol

# Unhandled Cases

## Defines

*include/linux/sched.h*

```
/* Used in tsk->state: */
#define TASK_RUNNING            0x0000
#define TASK_INTERRUPTIBLE      0x0001
#define TASK_UNINTERRUPTIBLE    0x0002
[...]
#define TASK_WAKEKILL           0x0100
[...]

/* Convenience macros for the sake of
set_current_state: */
#define TASK_KILLABLE                       \
   (TASK_WAKEKILL | TASK_UNINTERRUPTIBLE)
```

How to handle changes to #defined values?

```
 #define TASK_RUNNING           0x0000
 #define TASK_INTERRUPTIBLE     0x0001
-#define TASK_UNINTERRUPTIBLE   0x0002
+#define TASK_UNINTERRUPTIBLE   0x0004

 [...]
#define TASK_KILLABLE  (TASK_WAKEKILL | TASK_UNINTERRUPTIBLE)
```

Deduct defines and create trackable data structures.

```
enum abi_enum {
    abi_TASK_RUNNING = 0x0000,
    abi_TASK_INTERRUPTIBLE = 0x0001,
    abi_TASK_UNINTERRUPTIBLE = 0x0002,
    abi_TASK_WAKEKILL = 0x0100,
    abi_TASK_KILLABLE = 0x0102
};

void abi_func(enum abi_enum e) { }
```

Captured as ELF symbol



android

# Questions?

Maybe interesting:

- *Is capturing the observable ABI / API enough to avoid breakages?*
- *How about runtime checks? (ABI checksum?, libabigail at runtime?)*
- *How about upstream patches that transform #define into tagged enums?*
- *Would such an approach be valuable for (enterprise) distributions?*