# Livepatch module Elf format

This document outlines the Elf format requirements that livepatch modules must follow.

## 1. Background and motivation

Formerly, livepatch required separate architecture-specific code to write relocations. However, arch-specific code to write relocations already exists in the module loader, so this former approach produced redundant code. So, instead of duplicating code and re-implementing what the module loader can already do, livepatch leverages existing code in the module loader to perform the all the arch-specific relocation work. Specifically, livepatch reuses the apply_relocate_add() function in the module loader to write relocations. The patch module Elf format described in this document enables livepatch to be able to do this. The hope is that this will make livepatch more easily portable to other architectures and reduce the amount of arch-specific code required to port livepatch to a particular architecture.

Since apply_relocate_add() requires access to a module's section header table, symbol table, and relocation section indices, Elf information is preserved for livepatch modules (see section 5). Livepatch manages its own relocation sections and symbols, which are described in this document. The Elf constants used to mark livepatch symbols and relocation sections were selected from OS-specific ranges according to the definitions from glibc.

### Why does livepatch need to write its own relocations?

A typical livepatch module contains patched versions of functions that can reference non-exported global symbols and non-included local symbols. Relocations referencing these types of symbols cannot be left in as-is since the kernel module loader cannot resolve them and will therefore reject the livepatch module. Furthermore, we cannot apply relocations that affect modules not yet loaded at patch module load time (e.g. a patch to a driver that is not loaded). Formerly, livepatch solved this problem by embedding special "dynrela" (dynamic rela) sections in the resulting patch module Elf output. Using these dynrela sections, livepatch could resolve symbols while taking into account its scope and what module the symbol belongs to, and then manually apply the dynamic relocations. However this approach required livepatch to supply arch-specific code in order to write these relocations. In the new format, livepatch manages its own SHT_RELA relocation sections in place of dynrela sections, and the symbols that the relas reference are special livepatch symbols (see section 2 and 3). The arch-specific livepatch relocation code is replaced by a call to apply_relocate_add().

## 2. Livepatch modinfo field

Livepatch modules are required to have the "livepatch" modinfo attribute. See the sample livepatch module in samples/livepatch/ for how this is done.

Livepatch modules can be identified by users by using the 'modinfo' command and looking for the presence of the "livepatch" field. This field is also used by the kernel module loader to identify livepatch modules.

### Example:

**Modinfo output:**

```
% modinfo livepatch-meminfo.ko
filename:               livepatch-meminfo.ko
livepatch:              Y
license:                GPL
depends:
vermagic:               4.3.0+ SMP mod_unload
```

## 3. Livepatch relocation sections

A livepatch module manages its own Elf relocation sections to apply relocations to modules as well as to the kernel (vmlinux) at the appropriate time. For example, if a patch module patches a driver that is not currently loaded, livepatch will apply the corresponding livepatch relocation section(s) to the driver once it loads.

Each "object" (e.g. vmlinux, or a module) within a patch module may have multiple livepatch relocation sections associated with it (e.g. patches to multiple functions within the same object). There is a 1-1 correspondence between a livepatch relocation section and the target section (usually the text section of a function) to which the relocation(s) apply. It is also possible for a livepatch module to have no livepatch relocation sections, as in the case of the sample livepatch module (see samples/livepatch).

Since Elf information is preserved for livepatch modules (see Section 5), a livepatch relocation section can be applied simply by passing in the appropriate section index to apply_relocate_add(), which then uses it to access the relocation section and apply the relocations.

Every symbol referenced by a rela in a livepatch relocation section is a livepatch symbol. These must be resolved before livepatch can call apply_relocate_add(). See Section 3 for more information.

## 3.1 Livepatch relocation section format

Livepatch relocation sections must be marked with the SHF_RELA_LIVEPATCH section flag. See include/uapi/linux/elf.h for the definition. The module loader recognizes this flag and will avoid applying those relocation sections at patch module load time. These sections must also be marked with SHF_ALLOC, so that the module loader doesn't discard them on module load (i.e. they will be copied into memory along with the other SHF_ALLOC sections).

The name of a livepatch relocation section must conform to the following format:

```
.klp.rela.objname.section_name
^        ^^     ^ ^          ^
|________||_____| |__________|
   [A]      [B]        [C]
```

[A]
: The relocation section name is prefixed with the string ".klp.rela."

[B]
: The name of the object (i.e. "vmlinux" or name of module) to which the relocation section belongs follows immediately after the prefix.

[C]
: The actual name of the section to which this relocation section applies.

### Examples:

**Livepatch relocation section names:**

```
.klp.rela.ext4.text.ext4_attr_store
.klp.rela.vmlinux.text.cmdline_proc_show
```

**`readelf –sections` output for a patch module that patches vmlinux and modules 9p, btrfs, ext4:**

```
Section Headers:
[Nr] Name                          Type                    Address          Off    Size   ES Flg Lk Inf Al
[ snip ]
[29] .klp.rela.9p.text.caches.show RELA                    0000000000000000 002d58 0000c0 18 AIo 64   9  8
[30] .klp.rela.btrfs.text.btrfs.feature.attr.show RELA     0000000000000000 002e18 000060 18 AIo 64  11  8
[ snip ]
[34] .klp.rela.ext4.text.ext4.attr.store RELA              0000000000000000 002fd8 0000d8 18 AIo 64  13  8
[35] .klp.rela.ext4.text.ext4.attr.show RELA               0000000000000000 0030b0 000150 18 AIo 64  15  8
[36] .klp.rela.vmlinux.text.cmdline.proc.show RELA         0000000000000000 003200 000018 18 AIo 64  17  8
[37] .klp.rela.vmlinux.text.meminfo.proc.show RELA         0000000000000000 003218 0000f0 18 AIo 64  19  8
[ snip ]                                        ^                                             ^
                                                |                                             |
                                               [*]                                           [*]
```

[*]
: Livepatch relocation sections are SHT_RELA sections but with a few special characteristics. Notice that they are marked SHF_ALLOC ("A") so that they will not be discarded when the module is loaded into memory, as well as with the SHF_RELA_LIVEPATCH flag ("o" - for OS-specific).

**`readelf –relocs` output for a patch module:**

```
Relocation section '.klp.rela.btrfs.text.btrfs_feature_attr_show' at offset 0x2ba0 contains 4 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name + Addend
000000000000001f  0000005e00000002 R_X86_64_PC32          0000000000000000 .klp.sym.vmlinux.printk,0 - 4
0000000000000028  0000003d0000000b R_X86_64_32S           0000000000000000 .klp.sym.btrfs.btrfs_ktype,0 + 0
0000000000000036  0000003b00000002 R_X86_64_PC32          0000000000000000 .klp.sym.btrfs.can_modify_feature.isr
000000000000004c  0000004900000002 R_X86_64_PC32          0000000000000000 .klp.sym.vmlinux.snprintf,0 - 4
[ snip ]                                                                   ^
                                                                           |
                                                                          [*]
```

[*]
: Every symbol referenced by a relocation is a livepatch symbol.

## 4. Livepatch symbols

Livepatch symbols are symbols referred to by livepatch relocation sections. These are symbols accessed from new versions of functions for patched objects, whose addresses cannot be resolved by the module loader (because they are local or unexported global syms). Since the module loader only resolves exported syms, and not every symbol referenced by the new patched functions is exported, livepatch symbols were introduced. They are used also in cases where we cannot immediately know the address of a symbol when a patch module loads. For example, this is the case when livepatch patches a module that is not loaded yet. In this case, the relevant livepatch symbols are resolved simply when the target module loads. In any case, for any livepatch relocation section, all livepatch symbols referenced by that section must be resolved before livepatch can call apply_relocate_add() for that reloc section.

Livepatch symbols must be marked with SHN_LIVEPATCH so that the module loader can identify and ignore them. Livepatch modules keep these symbols in their symbol tables, and the symbol table is made accessible through module->symtab.

## 4.1 A livepatch module's symbol table

Normally, a stripped down copy of a module's symbol table (containing only "core" symbols) is made available through module->symtab (See layout_symtab() in kernel/module.c). For livepatch modules, the symbol table copied into memory on module load must be exactly the same as the symbol table produced when the patch module was compiled. This is because the relocations in each livepatch relocation section refer to their respective symbols with their symbol indices, and the original symbol indices (and thus the symtab ordering) must be preserved in order for apply_relocate_add() to find the right symbol.

For example, take this particular rela from a livepatch module::

```
Relocation section '.klp.rela.btrfs.text.btrfs_feature_attr_show' at offset 0x2ba0 contains 4 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name + Addend
000000000000001f  0000005e00000002 R_X86_64_PC32          0000000000000000 .klp.sym.vmlinux.printk,0 - 4

This rela refers to the symbol '.klp.sym.vmlinux.printk,0', and the symbol index is encoded
in 'Info'. Here its symbol index is 0x5e, which is 94 in decimal, which refers to the
symbol index 94.
And in this patch module's corresponding symbol table, symbol index 94 refers to that very symbol:
[ snip ]
94: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT OS [0xff20] .klp.sym.vmlinux.printk,0
[ snip ]
```

## 4.2 Livepatch symbol format

Livepatch symbols must have their section index marked as SHN_LIVEPATCH, so that the module loader can identify them and not attempt to resolve them. See include/uapi/linux/elf.h for the actual definitions.

Livepatch symbol names must conform to the following format:

```
.klp.sym.objname.symbol_name,sympos
^       ^^     ^ ^         ^ ^
|_______||_____| |_________| |
   [A]     [B]        [C]    [D]
```

[A]
: The symbol name is prefixed with the string ".klp.sym."

[B]
: The name of the object (i.e. "vmlinux" or name of module) to which the symbol belongs follows immediately after the prefix.

[C]
: The actual name of the symbol.

[D]
: The position of the symbol in the object (as according to kallsyms) This is used to differentiate duplicate symbols within the same object. The symbol position is expressed numerically (0, 1, 2...). The symbol position of a unique symbol is 0.

### Examples:

#### Livepatch symbol names:

```
.klp.sym.vmlinux.snprintf,0
.klp.sym.vmlinux.printk,0
.klp.sym.btrfs.btrfs_ktype,0
```

#### `readelf –symbols` output for a patch module:

```
Symbol table '.symtab' contains 127 entries:
   Num:    Value          Size Type    Bind   Vis     Ndx         Name
   [ snip ]
    73: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT OS [0xff20] .klp.sym.vmlinux.snprintf,0
    74: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT OS [0xff20] .klp.sym.vmlinux.capable,0
    75: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT OS [0xff20] .klp.sym.vmlinux.find_next_bit,0
    76: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT OS [0xff20] .klp.sym.vmlinux.si_swapinfo,0
  [ snip ]                                               ^
                                                         |
                                                        [*]
```

[*]
: Note that the 'Ndx' (Section index) for these symbols is SHN_LIVEPATCH (0xff20). "OS" means OS-specific.

## 5. Symbol table and Elf section access

A livepatch module's symbol table is accessible through module->symtab.

Since apply_relocate_add() requires access to a module's section headers, symbol table, and relocation section indices, Elf information is preserved for livepatch modules and is made accessible by the module loader through module->klp_info, which is a klp_modinfo struct. When a livepatch module loads, this struct is filled in by the module loader. Its fields are documented below:

```
struct klp_modinfo {
        Elf_Ehdr hdr; /* Elf header */
        Elf_Shdr *sechdrs; /* Section header table */
        char *secstrings; /* String table for the section headers */
        unsigned int symndx; /* The symbol table section index */
};
```