redhat.

LIVEPATCH MODULE CREATION

LPC 2016

Josh Poimboeuf

Overview

- Review current approaches
  - kpatch-build (out-of-tree)
  - kbuild (in-tree)
- Propose common in-tree approach

redhat.

kpatch-build

```
$ kpatch-build dirtycow.patch
Fedora/Red Hat distribution detected
Downloading kernel source for 3.10.0-327.36.2.el7.x86_64
Unpacking kernel source
Testing patch file
checking file mm/memory.c
Reading special section data
Building original kernel
Building patched kernel
Extracting new and modified ELF sections
memory.o: changed function: __get_user_pages
memory.o: changed function: follow_page_mask
Patched objects: vmlinux
Building patch module: kpatch-dirtycow.ko
SUCCESS
```


redhat.

But...

- Complex
  - Kernel special sections
    - .altinstructions, __ex_table, etc
  - gcc optimizations, quirks
    - switch statement jump tables, static local variables, etc…
- x86_64 only
- Brittle; maintenance nightmare!
- Is there a simpler option?

redhat.

kbuild

- Ordinary OOT module
- Copy/paste/modify function:

```
int livepatch_cmdline_proc_show(struct seq_file *m, void *v)
{
        seq_printf(m, "%s livepatch=1\n", saved_command_line);
        return 0;
}
```

- Register function with klp API
- Easy!

redhat.

But…

```
int livepatch_cmdline_proc_show(struct seq_file *m, void *v)
{
        seq_printf(m, "%s livepatch=1\n", saved_command_line);
        return 0;
}
```

- ‘saved_command_line’ not exported!

redhat.

Hack it together…

```
int livepatch_cmdline_proc_show(struct seq_file *m, void *v)
{
        static char *my_cmdline = NULL;

         if (!my_cmdline)
                 my_cmdline = kallsyms_lookup_name(“saved_command_line”);

        seq_printf(m, "%s livepatch=1\n", my_cmdline);
        return 0;
}
```

- Yucky, error prone, slow
- Doesn’t use .klp.relas, .klp.arch.*, etc

redhat.

Proposal: kbuild + klp-convert

- Post-processing tool
- Runs after .ko has been built and linked
- Integrated with kernel build
  - Triggers on MODULE_INFO(livepatch, "Y")
- Allows patch author to specify livepatch-specific metadata in module source file
- Converts metadata to format needed by livepatch kernel code

redhat.

struct klp_module_reloc

```
struct klp_module_reloc {
        void *sym;
        unsigned int sympos;
} __attribute__((packed));

#define KLP_MODULE_RELOC(obj) \
        klp_module_reloc \
        __attribute__((__section__(".klp.module_relocs." #obj)))
```


redhat.

klp_module_reloc usage

```
extern char *saved_command_line;

struct KLP_MODULE_RELOC(vmlinux) vmlinux_relocs[] = {
        {
                .sym = &saved_command_line,
                .sympos = 0,
        },
};

int livepatch_cmdline_proc_show(struct seq_file *m, void *v)
{
        seq_printf(m, "%s livepatch=1\n", saved_command_line);
        return 0;
}
```


redhat.

klp-convert

Before:

```
Relocation section [12] '.rela.klp.module_relocs.vmlinux' for section [11] '.klp.module_relocs.vmlinux' at
offset 0x28418 contains 1 entry:
  Offset      Type              Value   Addend  Name
  0           X86_64_64         0       +0      saved_command_line
```

After:

```
Relocation section [31] '.klp.rela.vmlinux..text' for section [ 2] '.text' at offset 0x491e8 contains 1 entry:
  Offset      Type              Value   Addend Name
  9           X86_64_PC32       0       -4     .klp.sym.vmlinux.saved_command_line,0
```


redhat.

Potential improvements

- Support for .klp.arch.altinstructions, .klp.arch.parainstructions
  - May need to rework klp API
- Warn on unresolved symbols
- Automatically detect and convert relocations (for unique symbols only)?
- Automatically detect and warn about gcc optimizations which would affect the patch?

redhat.

THE END

redhat.