Livepatching

The Good, The Bad, and The Ugly

What does it look like today?

Let's take a look at Canonical's offering:
Livepatch service

Getting started

- Need to sign up for Ubuntu Pro (and attach your key to the machine using sudo pro attach)
- Run the command sudo pro enable livepatch

Caveats

- Only certain kernels are supported

What is a kernel module

- Code that can be plugged into a running system
- Usually written for drivers (unless compiled directly into the kernel)
  - uvcvideo $\rightarrow$ Webcam drivers
- Runs in kernel mode (obviously)

How are live patches different?

- Patches are just a special kernel modules that are able to dynamically replace calls to functions
- Has additional consistency guarantees

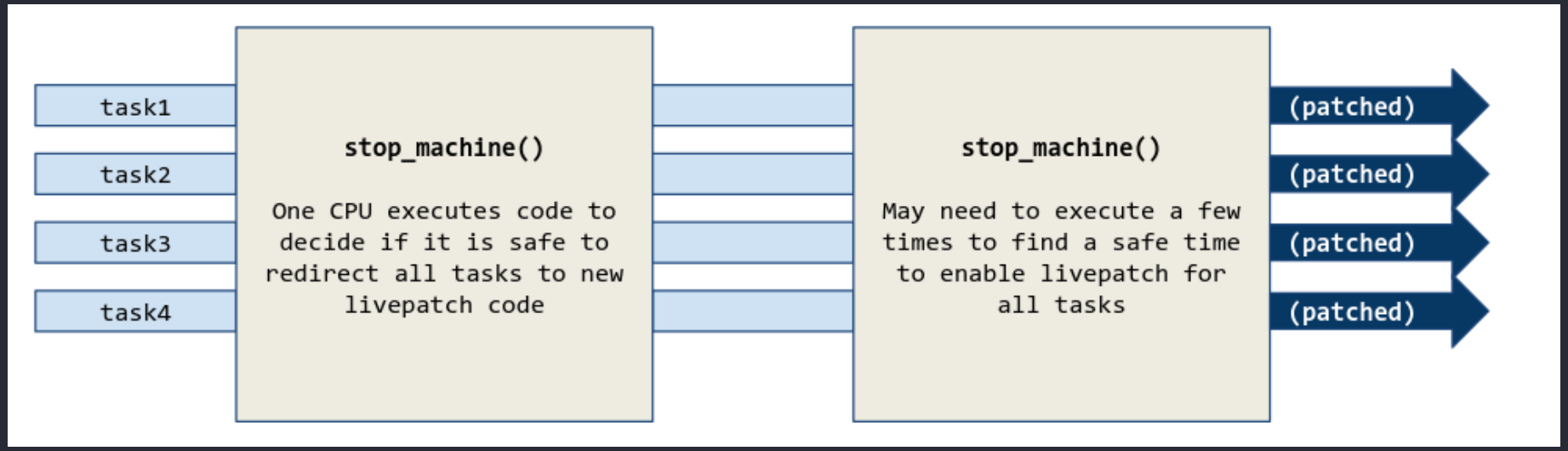
Origins

- First ideas of live patching came ~2008-ish with KSplice
  - Developed within MIT as a master's thesis
  - Spun off into KSplice Inc.

- Acquired by Oracle
  - Closed source in 2011
- Offered for free on Ubuntu desktop around 2015
- Currently available for RHEL and Oracle Linux

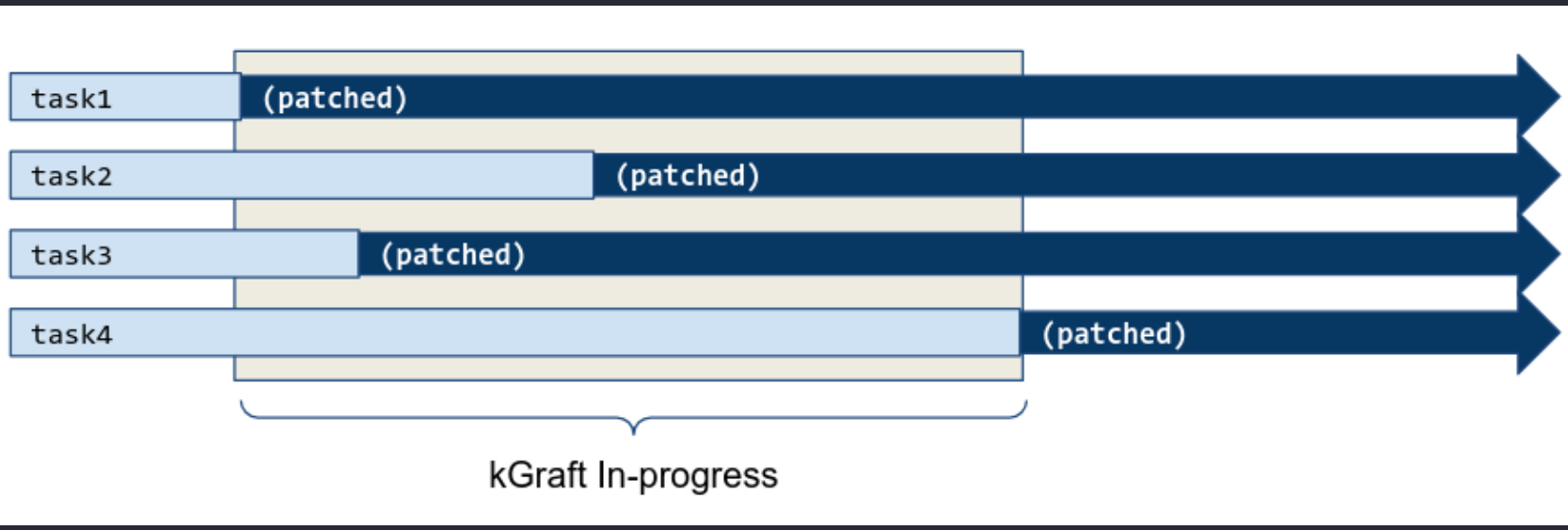
Red Hat

- Released in 2014, Red Hat created kPatch under GPL2
- Submitted for inclusion in the kernel in 2015


task1
task2
task3
task4
stop_machine()
One CPU executes code to decide if it is safe to redirect all tasks to new livepatch code
stop_machine()
May need to execute a few times to find a safe time to enable livepatch for all tasks
(patched)
(patched)
(patched)
(patched)

SUSE

- Released in 2014, SUSE created kGraft
  - Kernel part GPL2
  - Userspace part GPL3


task1
(patched)
task2
(patched)
task3
(patched)
task4
(patched)
kGraft In-progress

Linux 4.0

- A common livepatching core was implemented in Linux 4.0
  - Donated by both Red Hat and SUSE engineers
  - Only had the basic stuff (injection into functions pre-execution)

Consistency Model

- Uses a combination of kpatch and kGraft
  - Uses per-task consistency
  - Uses kpatch's stack trace switching model

What is the purpose of this?

- Uptime, mainly
  - "New CVE has been released but we cannot reboot this machine!"

What are the alternatives?

- Regular reboot
  - Slow, requires downtime
- Kexec
  - Faster, still requires downtime
  - Can leave the system in a weird state

Demo

What just happened?

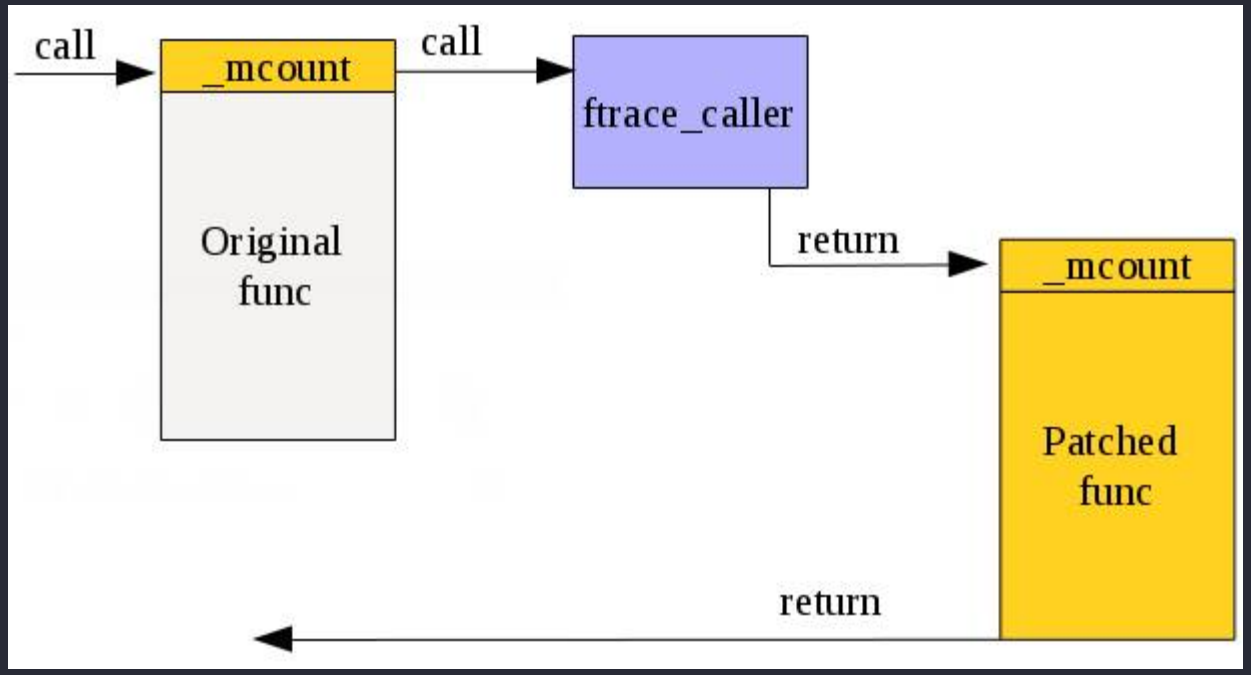
- The cmdline_proc_show function was dynamically replaced with our version
- ...But how?

Ftrace

- Facility in the kernel that allows developers to trace kernel function calls
- Used by Kprobes
  - Utility that allows developers to add/ remove code in existing functions

- Adds nop's to the beginning of every (supported) function call
  - These nops usually do nothing, but can be hooked into

```
crash> dis sysrq_handle_crash
0xffffffff8c41d930 <sysrq_handle_crash>:        nop    DWORD PTR [rax+rax*1+0x0]
0xffffffff8c41d935 <sysrq_handle_crash+5>:      push   rbp
0xffffffff8c41d936 <sysrq_handle_crash+6>:      mov    DWORD PTR [rip+0x13637a8],0x1        # 0xffffffff8d7810e8
0xffffffff8c41d940 <sysrq_handle_crash+16>:     mov    rbp,rsp
0xffffffff8c41d943 <sysrq_handle_crash+19>:     sfence
0xffffffff8c41d946 <sysrq_handle_crash+22>:     mov    BYTE PTR ds:0x0,0x1
0xffffffff8c41d94e <sysrq_handle_crash+30>:     pop    rbp
0xffffffff8c41d94f <sysrq_handle_crash+31>:     ret
crash>
```


call
_mcount
call
ftrace_caller
Original
func
return
_mcount
Patched
func
return

What does a patch look like?

```
#define pr_fmt(fmt) KBUILD_MODNAME ": " fmt

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/livepatch.h>
#include <linux/seq_file.h>

static int livepatch_cmdline_proc_show(struct seq_file *m, void *v)
{
    seq_printf(m, "%s\n", "hello RITLUG!");
    return 0;
}

static struct klp_func funcs[] = {
    {
```

Replacements and additions

- Patches can be stacked on top of one another
  - Not recommended
- Patches can also be a cumulative replacement
  - .replace property in module

Going further

- This is difficult
  - The function we patched before is remarkably simple. Short, no data changes, no internal calls.

/proc/meminfo

- Patches are usually built with modified function only
- This handler contains multiple calls to internal (not exported) functions
- When building, the linker doesn't know where these exist!

- Solution?
  - Use the kernel's module loader infrastructure
  - Apply relocations to the module ELF so functions can be linked at runtime

```
$ readelf --relocs livepatch-meminfo-string.ko
...
Relocation section '.klp.rela.vmlinux..text.meminfo_proc_show' at offset 0xac
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000003f  0054400000004 R_X86_64_PLT32    0000000000000000 .klp.sy
000000000046  0055000000002 R_X86_64_PC32     0000000000000000 .klp.s
...
```

- This is why tools such as kpatch exist
- Given a patch file and kernel source tree:
  - Build original kernel
  - Apply patch
  - Build patched object files
  - Run diffs on the ELF objects generated to find address of relocation
  - Generate patch with generated relocation offsets

Data changes

- Not necessarily recommended to livepatch at this point, but possible
  - Kernel **Shadow Variables**
  - Allows you to store *new* data without modifying kernel structures, which is unsafe/impossible.

```
#define KPATCH_SHADOW_NEWPID 0

...
newpid = klp_shadow_get_or_alloc(p, KPATCH_SHADOW_NEWPID,
        sizeof(*newpid), GFP_KERNEL,
        NULL, NULL);
...
```

```
#define KPATCH_SHADOW_NEWPID 0

newpid = klp_shadow_get(p, KPATCH_SHADOW_NEWPID);
if (newpid)
    seq_printf(m, "newpid:\t%d\n", *newpid);
...
```

```
diff --git a/net/mac80211/sta_info.h b/net/mac80211/sta_info.h
index d77ff709063038..d3a6d8208f2f85 100644
--- a/net/mac80211/sta_info.h
+++ b/net/mac80211/sta_info.h
@@ -267,6 +267,7 @@ struct ieee80211_tx_latency_stat {
  * @drv_unblock_wk: used for driver PS unblocking
  * @listen_interval: listen interval of this station, when we're acting as AP
  * @_flags: STA flags, see &enum ieee80211_sta_info_flags, do not use directly
+ * @ps_lock: used for powersave (when mac80211 is the AP) related locking
  * @ps_tx_buf: buffers (per AC) of frames to transmit to this station
  *     when it leaves power saving state or polls
  * @tx_filtered: buffers (per AC) of frames we already tried to
@@ -356,10 +357,8 @@ struct sta_info {
        /* use the accessors defined below */
        unsigned long _flags;

-       /*
-        * STA powersave frame queues, no more than the internal
-        * locking required.
-        */
+       /* STA powersave lock and frame queues */
+       spinlock_t ps_lock;
        struct sk_buff_head ps_tx_buf[IEEE80211_NUM_ACS];
        struct sk_buff_head tx_filtered[IEEE80211_NUM_ACS];
        unsigned long driver_buffered_tids;
```

Other functions

- klp_shadow_free()
- klp_shadow_alloc()
- klp_shadow_free_all()

Other Limitations

- Cannot patch notrace functions
  - This includes any functions with the inline annotation
- Very limited support for patching syscalls
  - Most are implemented with inline, but workarounds can be done
- Cannot really change the function declaration
  - No parameter changes, only additional variables and such

Current State of Livepatching

- Mostly enterprise only
  - This is due to cost

Current State of Livepatching

- Canonical: Free up to 5 machines; $500/yr/sys
- KernelCare: $45/yr/sys
- Oracle: $2299/yr
- SUSE: $2198/yr
- RedHat: $1299/yr

Note: Oracle, SUSE, and RedHat may not be accurate

Is it worth it?

- If you happen to have a system that cannot afford downtime and need to apply a fix immediately, sure.
  - For 99% of people, no.
  - Follow proper development practices and you won't get stuck in a situation where you need this.

Is it worth it?

- There's a reason it is as expensive as it is
  - Hard to apply (pretty much need to compile against the exact kernel you are running)
  - Even harder to write (it's very much an art)
  - Not really worth it unless you need it, in which case you will spend the money

Is it worth it?

- Very limited at the end of the day
  - Cannot really change caller, so no new parameters
  - Difficult to change structs
  - Not everything can be patched

Good Sources

- https://ruffell.nz/programming/writeups/2020/04/20/everything-you-wanted-to-know-about-kernel-livepatch-in-ubuntu.html
- https://developer.ibm.com/tutorials/live-patching-the-linux-kernel/
- https://www.redhat.com/en/topics/linux/what-is-linux-kernel-live-patching
- https://www.kernel.org/doc/html/v6.14-rc4/livepatch/index.html

- https://mkyong.com/linux/an-introduction-to-kernel-live-patching-on-linux/
- https://github.com/dynup/kpatch/blob/master/doc/patch-author-guide.md
- https://events.linuxfoundation.org/wp-content/uploads/2024/05/Linux-Livepatch_-Introduction-Mentorship-Webinar-5-22-24.pdf
- https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules.html#

- https://opensource.com/article/21/7/linux-kernel-ftrace
- https://www.linkedin.com/pulse/linux-kernel-module-relocation-process-david-zhu-tdb3c/