



Live Patching: The long road from Kernel to User Space

João Moreira

Toolchain Engineer - SUSE Labs

jmoreira@suse.de

Software has bugs, and bugs have to be fixed

- + security issues
- + execution degradation
- + undefined behavior



Fixing bugs:

- + replace broken software in disk
- + shutdown running software instance
- + start software based on fixed version
- + wait until process is ready
- + re-establish services



Fixing bugs: downtime

- + replace broken software in disk
- + shutdown running software instance
- + start software based on fixed version
- + wait until process is ready
- + re-establish services



Downside of downtime

- + Some services may take very long to restart
- + Active connections will drop
- + Interruption of large computations



Live Patching

- + Fixing bugs in the **OS** without rebooting
- + Already a thing in the Linux Kernel
- + Fixing bugs in **applications** without restart
- + A wild challenge in user space



Kernel Live Patching

- + Ksplice, kGraft, kpatch... **upstream**
 - x86_64, s390x, powerpc (arm, arm64 soon)
- + Patch code is loaded in the kernel as a module



Kernel Live Patching

+ **ftrace** framework:

- GCC emits ftrace calls on function prologues
- **calls** are overwritten with **nops** during boot
- **nops** can be reverted to **calls** during runtime
- patched function calls handler which does the...



Kernel Live Patching

+ **ftrace** framework:

- GCC emits ftrace calls on function prologues
- **calls** are overwritten with **nops** during boot
- **nops** can be reverted to **calls** during runtime
- patched function calls handler which does the...

★★**MAGIC**★★

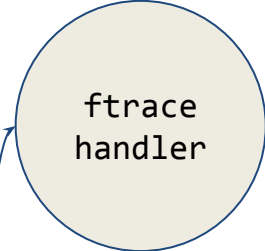


```
<foo>:  
...  
call bar  
...
```

```
<bar>:  
nop1  
push  
push  
...  
pop  
pop  
ret
```

```
<foo>:  
...  
call bar  
...
```

```
<bar>:  
call ftrace_handler  
push  
push  
...  
pop  
pop  
ret
```



```
<new_bar>:  
nop1  
push  
push  
...  
pop  
pop  
ret
```

**LIVE
PATCH!**



Call redirection

- + Functions are patched one at a time
- + Scheme works fine for patching single functions
- + ...but many function patching requires consistency
 - + Start using **all** new versions at the same time
 - + Switch happens when it is safe
 - + Patched functions cannot be active



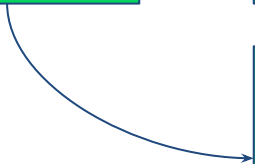
```
<dog>:  
...  
call cat
```

```
<new_cat>:  
...  
call fish
```

```
<cat>:  
PATCHED
```

```
<fish>:  
NOT YET  
PATCHED
```

```
<new_fish>:
```



```
<dog>:  
...  
call cat
```

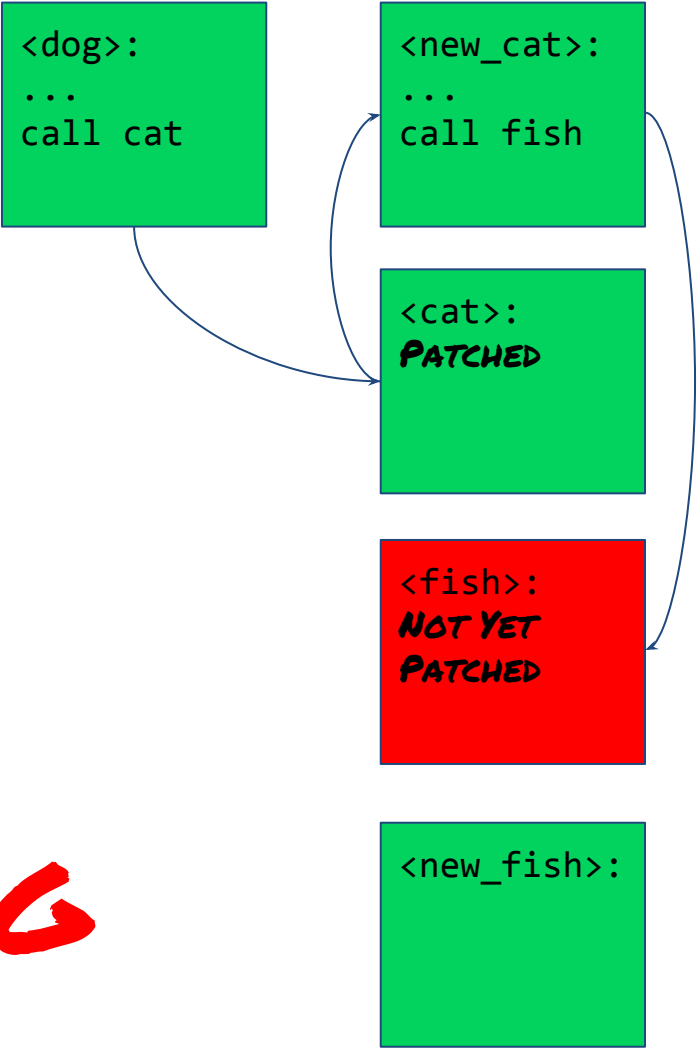
```
<new_cat>:  
...  
call fish
```

```
<cat>:  
PATCHED
```

```
<fish>:  
NOT YET  
PATCHED
```

```
<new_fish>:
```





WRONG



```
<dog>:  
...  
call cat
```

```
<new_cat>:  
...  
call fish
```

```
<cat>:  
NOT  
PATCHED  
call fish
```

```
<fish>:  
NOT  
PATCHED
```

```
<new_fish>:  
...
```

RIGHT!

```
<dog>:  
...  
call cat
```

```
<new_cat>:  
...  
call fish
```

```
<cat>:  
PATCHED
```

```
<fish>:  
PATCHED
```

```
<new_fish>:  
...
```

RIGHT!



Hybrid Consistency Model

- + **kGraft**: Per-task consistency and syscall barrier
- + **kpatch**: Stack trace switching
- + Plus some fallback options to make it flexible



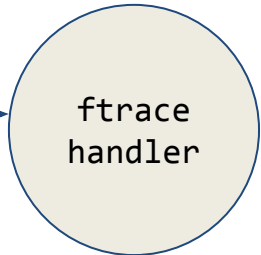
Per-task Consistency

- + Threads in kernel are migrated one by one
- + Switching done by the ftrace handler



```
<foo>:  
...  
call bar  
...
```

```
<bar>:  
call ftrace_handler  
push  
push  
...  
pop  
pop  
ret
```



```
<new_bar>:  
nop1  
push  
push  
...  
pop  
pop  
ret
```



Syscall Barrier

- + Kernel exit switch
- + Safe to patch when thread returns to user space
 - + After syscalls, user space IRQ or signal



Stack Checking

- + Enables patching of sleeping tasks
- + Checks if affected functions are active
 - + If yes, retry later
- + Requires config **HAVE_RELIABLE_STACKTRACE**
 - + rarely the case...
 - + Recent merge of ORC



Open challenges

- + Static or non-exported symbols in patches
 - + Use of kallsyms
 - + klp-convert: automatic resolution of symbols



Open challenges

+ GCC Optimizations

- + Inlining and partial inlining
 - + Removing unused parameters
 - + Changing parameters from reference to value
 - + Creating many variants of the same function
 - + Eliminating calls to pure const functions
 - + Caller not saving registers when supposed to
- + and more of crazy stuff...



Open challenges

+ Very hard to deal with data structure changes



ok ok, Kernel is **burning hot...**
... but what about **user space?**



User Space Live Patching

- + Currently lacks a complete implementation
- + Under development at SUSE



Is it Different?

- + No ftrace
- + Harder to identify when it is safe to patch
 - + Unable to use syscall barriers
 - + Stack checking is also unreliable
- + Not every application can load code dynamically



Libulp

- + User Space Live Patching Library
- + Can be statically linked to application
- + Can also be LD_PRELOAD'ed
- + Signals and checkpoints used to start live patching
- + Loads code using glibc's dlopen



Libulp Consistency

- + Checkpoints in code synchronize the patching process
- + Check trigger before applying the patch
- + Functions stacked under the checkpoint can't be patched
- + Multi-threaded applications:
 - All threads need to reach checkpoint
 - After patched, threads leave checkpoint together
 - No need to track versions



Detour-based Patching

- + Just like in the Kernel
- + Fixed functions are loaded elsewhere in memory
- + Bad function's prologue is patched with a **jmp**
- + **jmp** target is the patched function



```
<foo>:  
...  
call bar  
...
```

```
<bar>:  
nopl  
push  
push  
...  
pop  
pop  
ret
```

```
<foo>:  
...  
call bar  
...
```

```
<bar>:  
jmp new_bar  
push  
push  
...  
pop  
pop  
ret
```

```
<new_bar>:  
nopl  
push  
push  
...  
pop  
pop  
ret
```

LIVE
PATCH!



Libulp Consistency

- + But this model is not that good...
 - + Multi-threaded requirement would lead to dead-locks
 - + Tailoring checkpoints inside code is undesirable



So we keep searching :-)





We adapt. You succeed.



Live Patching: The long road from Kernel to User Space

João Moreira

Toolchain Engineer - SUSE Labs

jmoreira@suse.de