User Space Live Patching

João Moreira
SUSE Labs

User Space Live Patching

João Moreira
(formerly at) SUSE Labs
joao.moreira@lsc.ic.unicamp.br

Software has bugs,
and bugs have to be fixed

+ security issues
+ execution degradation
+ undefined behavior

Fixing bugs

+ kill the process
+ replace the respective binary with a fixed version
+ restart the process
+ wait until process is ready
+ re-establish services

Fixing bugs: **downtime**

+ kill the process
+ replace the respective binary with a fixed version
+ restart the process
+ 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 live software without restart
+ Already a thing in the Linux kernel

Libpulp

+ User Space Live Patching Library
+ Actually... not only a library, but a full framework

Quiessence

+ Changes should not lead to inconsistent states
+ Patches must be applied atomically
+ Functions cannot be patched while running

Kernel Consistency model

+ Execution boundary between user and kernel space
+ Hold new kernel threads and wait all others to finish
+ Safe to patch

+ Stack unwinding
+ Identify that to-be-patched functions are not running
+ Safe to patch

Consistency model

Kernel

+ Execution boundary between user and kernel space
+ Hold new kernel threads and wait all others to finish
+ Safe to patch

+ Stack walking
+ Identify that no-be-patched functions are not running
+ Safe to patch

HOW?!

Consistency model

Kernel

+ Execution boundary between user and kernel space
+ Hold new kernel threads and wait all workers to finish
+ Safe to patch

+ Stack walking
+ Identify that to-be-patched functions are not running
+ Safe to patch

libpulp to the RESCUE!!1!

libpulp Consistency Model

+ Uses shared libs model to identify quiescent states
+ If lib was **not** entered, all its functions can be patched
+ Before patch is applied, check if library was entered

libpulp Consistency Model

+ Uses shared library to manage different states
+ If lib was **not** entered, all its functions can be patched
+ Before patch is applied, check if library was entered

HOW?!

For now, imagine that we...

+ can magically change the functions in a process
+ just need to ensure that these functions aren't running

libpulp Consistency Model

+ Entry points to the library are its exported functions
+ Referenced in the ELF dynamic symbol table (.dynsym)

libpulp Consistency Model

+ Linker emits .ulp section with entries for exp. functions

libpulp Consistency Model

+ Linker emits .ulp section with entries for exp. functions
+ .dynsym symbols modified to point to .ulp entries
    - relocations are now resolved to .ulp entry

libpulp Consistency Model

+ Linker emits .ulp section with entries for exp. functions
+ .dynsym symbols modified to point to .ulp entries
    - relocations are now resolved to .ulp entry
+ .ulp saves function reference and jumps to ulp_entry

libpulp Consistency Model

+ Linker emits .ulp section with entries for exp. functions
+ .dynsym symbols modified to point to .ulp entries
    - relocations are now resolved to .ulp entry
+ .ulp saves function reference and jumps to ulp_entry
+ ulp_entry flags entrance, realigns stack, calls function

libpulp Consistency Model

+ Linker emits .ulp section with entries for exp. functions
+ .dynsym symbols modified to point to .ulp entries
    - relocations are now resolved to .ulp entry
+ .ulp saves function reference and jumps to ulp_entry
+ ulp_entry flags entrance, realigns stack, calls function
+ Function returns to ulp_entry

libpulp Consistency Model

+ Linker emits .ulp section with entries for exp. functions
+ .dynsym symbols modified to point to .ulp entries
  - relocations are now resolved to .ulp entry
+ .ulp saves function reference and jumps to ulp_entry
+ ulp_entry flags entrance, realigns stack, calls function
+ Function returns to ulp_entry
+ ulp_entry flags exit, restores return address, returns


a.out
<foo>:
...
mov %rax, %rbx
call fprintf@GLIBC
mov %rax, %rbx
...
glibc
section .ulp:
...
lea &fprintf, %r11
push %r11
jmp &ulp_entry
...
section .ulp.track:
<ulp_entry>:
glibc_flag = 1
synch_t_universe();
save original_return
fix_stack
pop %r11
call *%r11
load original_return
glibc_flag = 0
retq
section .text:
...
<fprintf>:
...
retq

Thread-local Universes

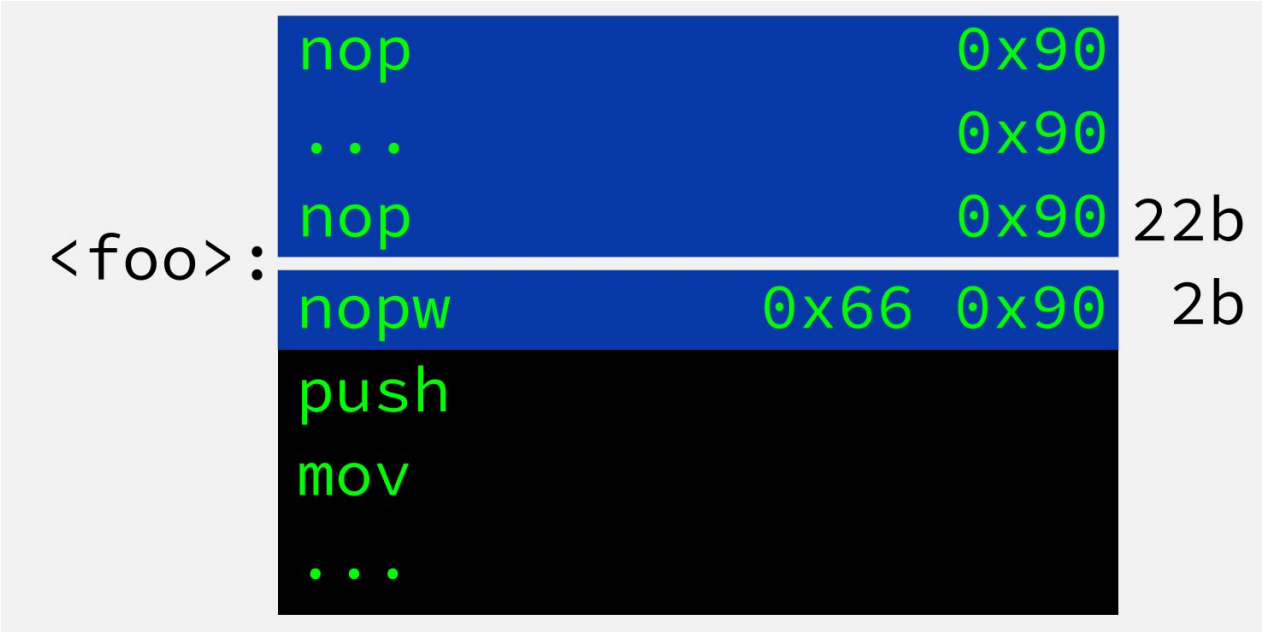
+ We don't want to wait for all threads to leave the library
+ Some may never leave the library
+ libpulp keeps per-thread patching states, or universes

Thread-local Universes

+ One global universe counter
    - Updated upon patching
+ Per-thread universe counters
    - Synchronized to the global universe in ulp_entry
    - When a patch is effectively applied to a thread

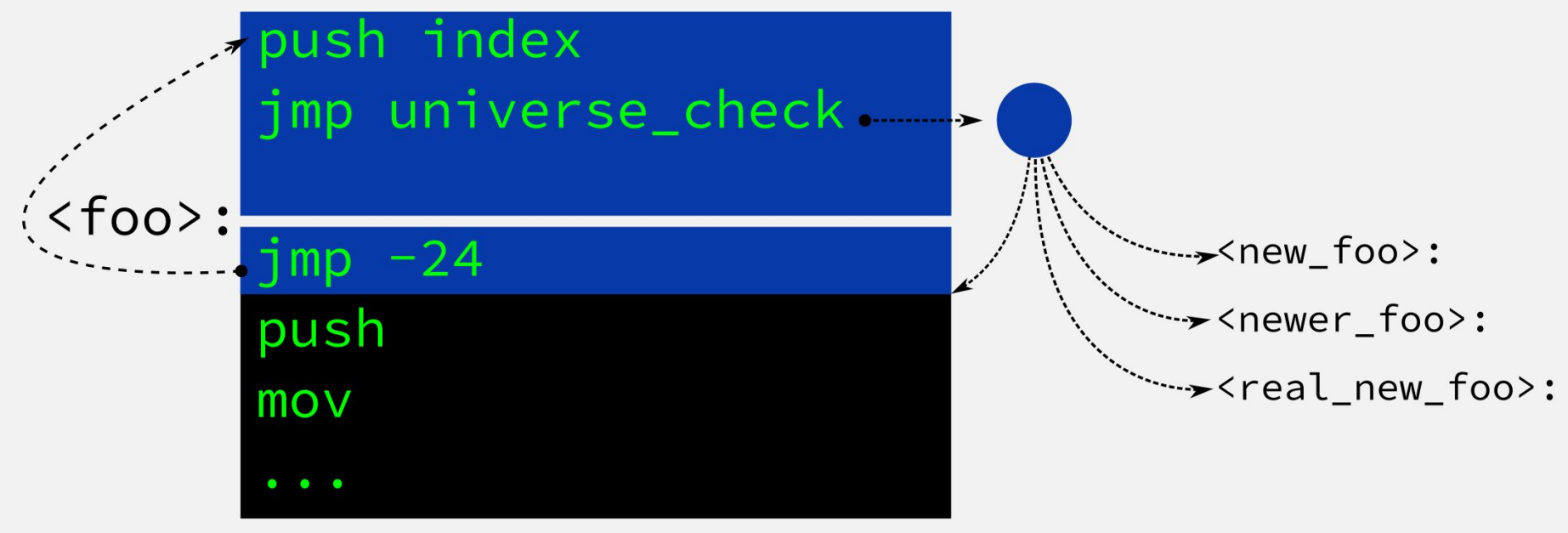
Thread-local Universes

+ Functions are emitted with padding nops area



Thread-local Universes

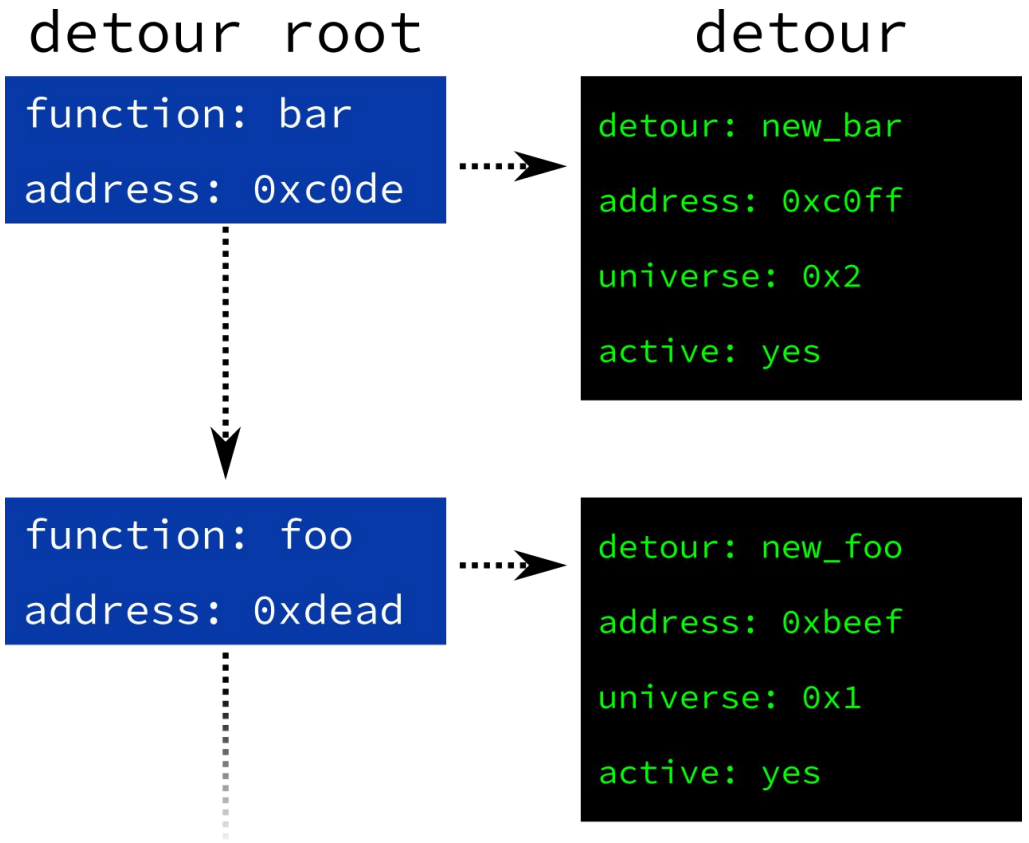
+ Nops modified into universe checker when patched



Thread-local Universes

+ Libpulp keeps a list of patched functions
+ Each node contains another list of function versions
+ Universe checking routine selects which detour to take

Thread-local Universes



libpulp

+ Library that can be LD_PRELOAD'ed
+ Provides self-modifying capabilities
+ Keeps needed data structures
+ Activated from the outside, through ptrace

libpulp

+ Library that can be LD_PRELOAD'ed
+ Provides self-modifying capabilities
+ Keeps needed data structures
+ Activated from the outside, through ptrace
    - This is the magic

All Together Now!

+ **P** is running process that LD_PRELOAD'ed libpulp
+ **P** uses specially compiled libs
+ We need to fix function **F** in lib **L**, but we can't kill **P**

+ A ptrace-based tool called **T** (trigger) attaches to **P**

All Together Now!

+ **T** stops **P**, parses its memory and saves its context
+ Redirects a thread to a *patch_apply* routine in libpulp
+ Redirects all other threads to a infinite loop routine
+ Restarts **P**

All Together Now!

+ patch_apply:
    - Modifies the to-be-patched functions
    - Loads .so file with function replacements
    - Updates data structures and increments universe
    - Interrupts, returning the control to **T**
+ **T** restores the original context and restarts **P**

All Together Now!

+ **P** calls **F** in **L**, which is being entered by the thread
+ Control-flow goes through ulp_entry
+ Thread-local universe counter is updated
+ **F** first runs the universe checking routine
+ New version of **F** is executed

All Together Now!

+ **P** calls **F** in **L**, from a thread which was already in **L**
+ Control-flow goes through ulp_entry
+ Thread-local universe update is bypassed
+ **F** first runs the universe checking routine
+ Thread-local universe is obsolete
+ Previous version of **F** is executed

The Trigger

+ Fully based on ptrace
+ Uses original binary to map all symbols within the process
+ Checks if libpulp was loaded into the process memory
+ Hijacks control-flow of threads to invoke libpulp routines

Live patch anatomy

+ Two separate parts
+ Compiled .so file that contains replacement functions
+ Metadata file with data required for applying the patch
    - Names of functions that will be replaced
    - Names of replacement functions
    - Sanity check: dependencies, target build-ids

Metadata Generation

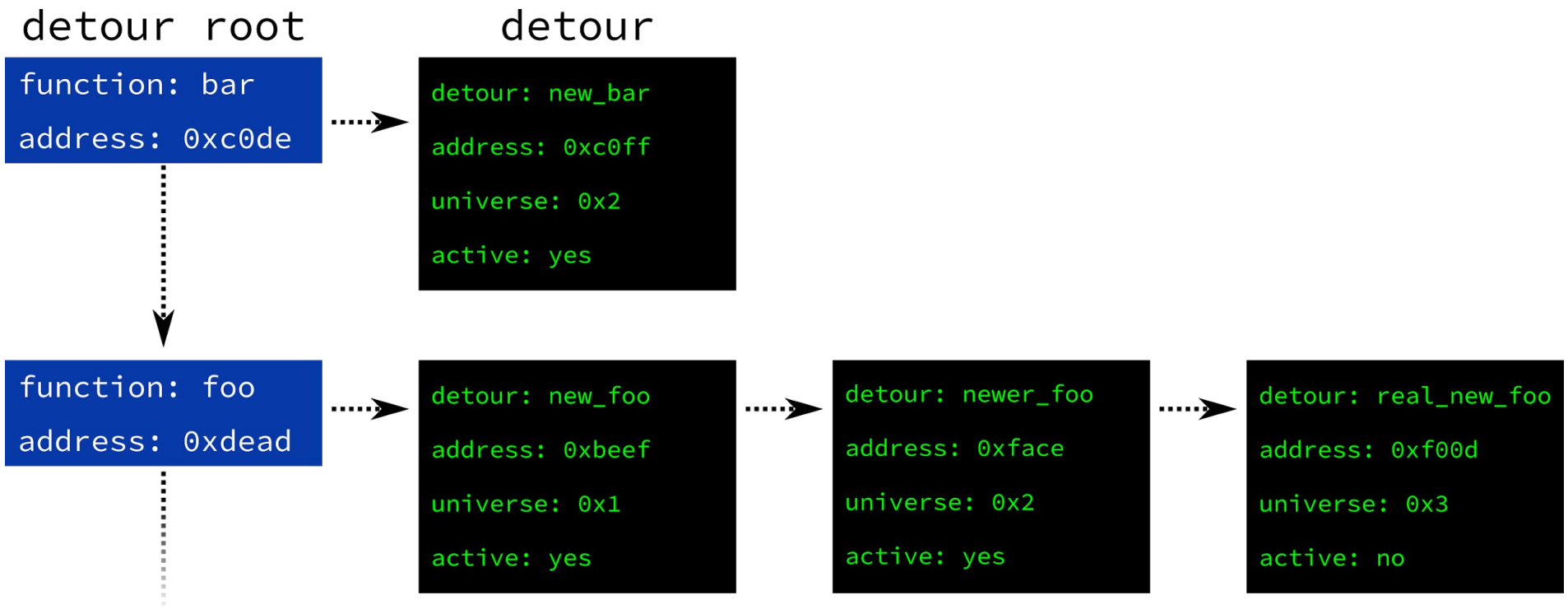
+ There is also a **packer tool**
+ Gets patch description text file and all objects involved
+ Generates metadata and reverse patches automatically

Stacked Patches

+ Multiple patches can be applied to the same process
+ Universe may be higher than the universes of available detours for given functions
+ Detour with higher universe below the compared universe is picked

Unpatching

+ Unpacthing is similar to patching
+ Global universe is incremented
+ Doesn't load .so, only marks detours as inactive
+ Inactive detour picked if its universe matches exactly

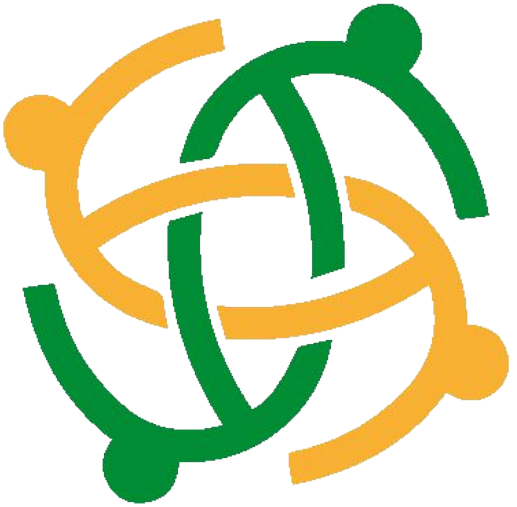

detour root
detour
function: bar
address: 0xc0de
detour: new_bar
address: 0xc0ff
universe: 0x2
active: yes
function: foo
address: 0xdead
detour: new_foo
address: 0xbeef
universe: 0x1
active: yes
detour: newer_foo
address: 0xface
universe: 0x2
active: yes
detour: real_new_foo
address: 0xf00d
universe: 0x3
active: no

Overheads

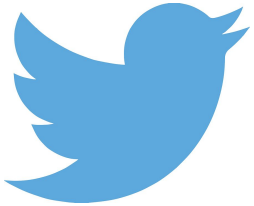
+ ~2% for libpulp-prepared glibc on SPEC

+ Worst case scenario for a process with a patch-applied
  - Recursive fibonacci sequence computation
  - Similar to having all called functions patched
  - Up to 50x overhead

github.com/SUSE/libpulp



linuxdev-br



twitter.com/linuxdevbr

instagram.com/linuxdevbr

t.me/linuxdevbr

User Space Live Patching

João Moreira
(formerly at) SUSE Labs
joao.moreira@lsc.ic.unicamp.br