Virtuozzo

Painless Live Userspace Patching

https://github.com/skinsbursky/presentations/blob/master/lup-devconf.odp

Stanislav Kinsburskiy
stanislav.kinsburskiy@gmail.com

Summary

- What is "Live Patching"?
- Why it's needed?
- Types of binary code
- Process state
- What can be patched?
- Binary patch creation
- When binary patch can be applied?
- Where to place new code?
- Userspace live patching in pictures
- Why is it called "painless"?

What is "Live Patching"?

- Change a piece of code in a process
- Preserve process state
- Do it safe

Why it's needed?

➢ Get rid of heavy services restart

➢ Reduce service downtime in case of critical vulnerabilities, because of:

  ➢ No need in application restart

  ➢ **No need in migration**

Types of binary code

➢ Statically-linked

➢ Dynamically-linked

   ➢ Load-time relocation

   ➢ Position independent code (PIC)

Process state

➢ Process state is like a cards castle:
  ➢ strongly associated
  ➢ accurately verified
➢ Contains:
  ➢ statically allocated variables
  ➢ dynamically allocated variables
  ➢ stack content
➢ Changes during runtime

What can be patched?

- Not any binary patch can be applied
- Fundamental Limitations:
  - Static data of different size
  - Dynamically allocated objects of different size
- Source code review is required :(

Binary patch creation

➢ Information about binary is required

➢ Patch code on function basis:

    ➢ Reliable

    ➢ Relatively simple

➢ Information about symbols required:

    ➢ Name

    ➢ Size

    ➢ Address

When binary patch can be applied?

➤ Process external stop/resume is essential

➤ Not at any moment of time:

   ➤ Process can execute the code to patch

   ➤ Code to patch can be referenced in the call stack

➤ Stack unwinding is essential:

   ➤ Need to catch the process outside old code

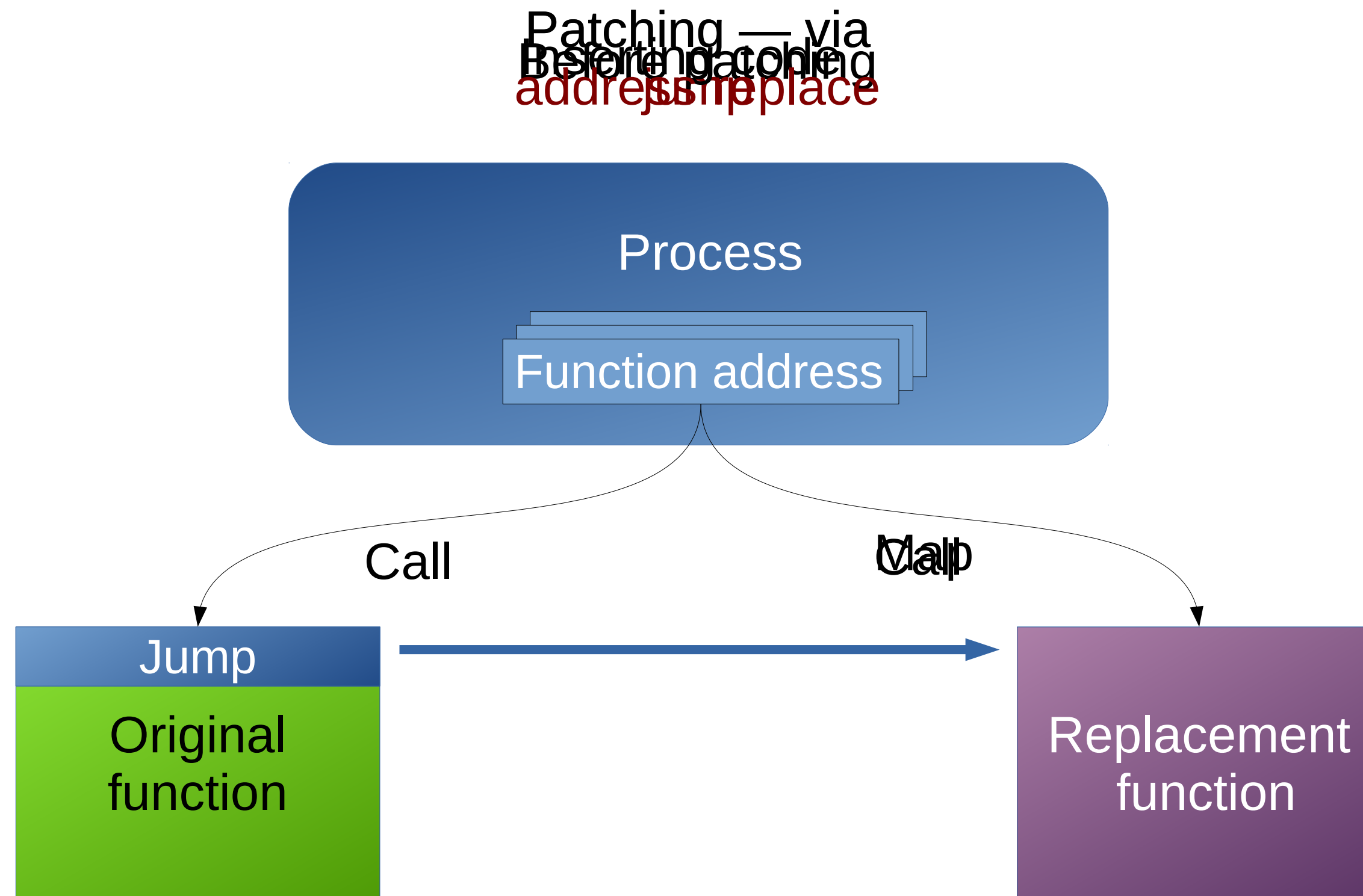   ➤ Or at least outside functions to patch

Where to place new code?

➢ Has to be placed into process address space:

  ➢ Either as binary blob

  ➢ Or as a file mapping

➢ Static code has to be "fixed" in place

➢ PIC code can be mapped as is, but:

  ➢ Initialization required: external and global symbols

  ➢ Corresponding data has to be copied

➢ Need to redirect execution to the new code

Userspace live patching in pictures

Patching — via
Before patching
Restoring code
address jumpplace

Process

Function address

Call   Map Call

Jump

Original function

Replacement function

Why is it called "painless"?

- ➤ Live patching can't be painless :)
- ➤ No kernel changes required!
- ➤ Binary patch creation can be based on ELF parsing
- ➤ Libcompel can be used for:
  - ➤ Task stop
  - ➤ Task resume
  - ➤ Code insertion
  - ➤ Source: https://github.com/xemul/criu/tree/criu-dev
- ➤ Libunwind can be used for stack unwinding
  - ➤ Source: http://git.savannah.gnu.org/cgit/libunwind.git