kpatch

Have your security and eat it too!

Josh Poimboeuf
Senior Software Engineer, Red Hat
LinuxCon North America
August 22, 2014

Agenda

- What is kpatch?
- Why use kpatch?
- Demo
- How it works
- Features & Limitations
- Try it!
- Questions?

What is kpatch?

- Live kernel patching framework
- Patch a running kernel
- No reboots
- No disruption to applications
- Used for security and stability fixes
    - Not for major kernel updates

Open source

- Started as internal Red Hat project
- Feb 2014: Released on github
- Goal: merge into upstream Linux
- Already stable and useful
- 100% self-contained
- Works on many distributions
    - Fedora, Ubuntu, Debian, Arch, RHEL7*, CentOS7, OL7

* Use at your own risk

Why kpatch?

Kernel bugs are problematic

- Many security bugs waiting to be found
  - Large attack surface
  - Huge code base
- System-level impact -> high priority
- Many high-priority security fixes
- Kernel update = reboot
- Kernel updates are often delayed

Why is rebooting a problem?

- Disruption to users/applications
- Sysadmins don't always have control of users or applications
- Many applications aren't distributed
    - Re-architecting can be expensive or impractical
- Distributed systems need to reboot too
- (Up)time is money
- Hardware reboot failures

Security vs business factors

- Security doesn't exist in a vacuum
- Judgment calls / business decisions
- Risk of getting hacked vs reboot costs
- Reboot now? Or risk it and wait?

Security at the expense of flexibility comes at the expense of security

kpatch to the rescue

- Remove security / flexibility trade-offs
- No more risk analysis, judgment calls, business decisions, etc.
- Apply security fixes immediately
- No disruption to users/applications
- Can wait for a better time to reboot
- Scheduled reboots

kpatch benefits

- Security-focused
  - Flexibility and predictability
- Uptime-focused
  - Security
- The rest of us
  - All of the above
- Decouple (arbitrary) security fix schedule from reboot schedule

“But this sounds crazy...”

- Integrated with kernel (not a Band-Aid)
  - Uses ftrace to do the patching
  - Replacement functions are first class functions
  - Compatible with oops, ftrace, kprobes, kdump, perf, etc.
  - Taint flag
- Patching process is deterministic
- Simple design
  - Code is 100% self-contained

Is it safe?

*if you're very careful with your patch selection

Demo

How it works

How it works

1. Build the patch module
   - kpatch-build foo.patch
2. Patch the kernel
   - kpatch load kpatch-foo.ko

Building the patch module

- Much harder than patching the kernel!
- Compile kernel with/without patch
- Compare binaries
- Detect which have functions changed
- Extract object code of changed functions into patch module
- Edge cases...
  - Compiler optimizations, kernel special ELF sections

Determining patch safety

- Some patches are inherently unsafe
  - Data structure changes
  - Data semantic changes
- Tooling does *some* safety analysis
- Impossible for a program to definitively determine whether a patch is safe

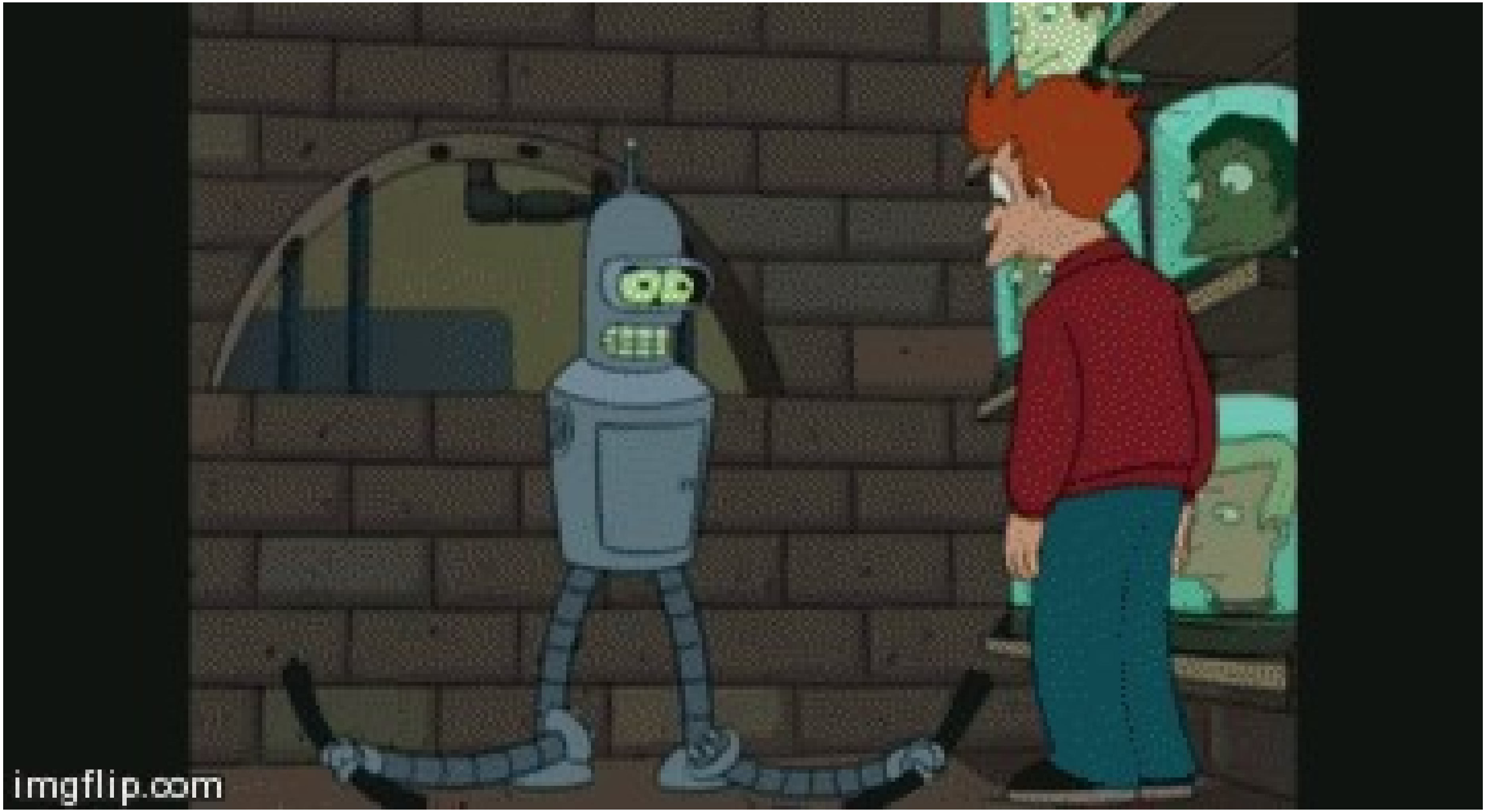
CAUTION
DO NOT TOUCH THE EDGES OF THIS SIGN
ALSO, THE BRIDGE IS OUT AHEAD

A human must analyze each patch to determine whether it's safe to apply in a live patching context!

Human patch analysis

- What function does
- What patch does
- How patch changes data interactions
- Modify patch if needed
- Kernel expert recommended
    - Or get your Linux distribution to do it

Patching the kernel

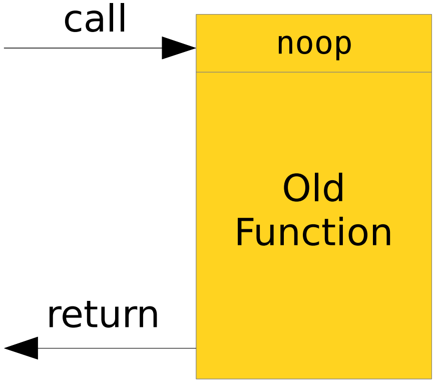


Patching the kernel

1. Load new functions into memory
2. Link new functions into kernel
   - Allows access to unexported kernel symbols
3. Activeness safety check
   - Prevent old & new functions from running at same time
   - stop_machine() + stack backtrace checks
4. Patch it!
   - Uses ftrace

Patching with ftrace

Before patching:



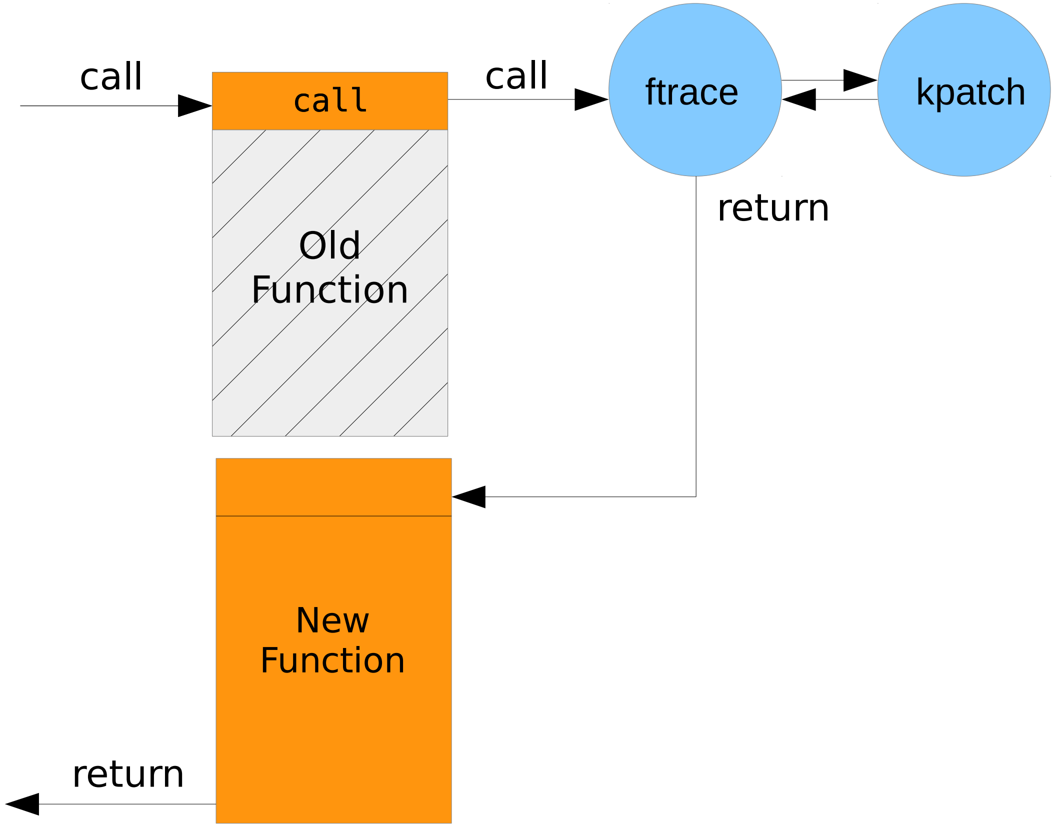
call

noop

Old Function

return

After patching:



call

call

Old Function

call

ftrace

kpatch

return

New Function

return

Features & Limitations

Features

- Patch rollback
- Patch on reboot
- Multiple patches
- Atomic patch upgrade
- Module patching (and deferred)
- User load/unload hook functions
- Skip backtrace safety check

Limitations

- Human safety analysis required!
- Not a general purpose upgrade tool
- ~80% of all CVE patches currently supported
  - Data structure changes, edge cases
  - Goal: 99%
- stop_machine() latency: 1ms – 40ms
- Currently x86_64 only

kpatch on RHEL 7

- Not supported at this time
- Working with small customer group to get early operational feedback
- Goal: get it (or something like it) merged upstream first

Try it!

Feedback wanted

- We've built the “car”
  - Kicked the tires
  - Many test drives
  - Not many long family road trips or daily commuters yet?
- Looking for brave users to solve real-world problems with it
- Help influence the direction of kpatch

Try it!

- See the README on github
  - Quick start guide
  - More in-depth information
- Open github issues
- Join the mailing list
- Ping us on IRC
- Contributors welcome!

Reference

- Github repository
  - https://github.com/dynup/kpatch
- Mailing list
  - https://www.redhat.com/mailman/listinfo/kpatch
- IRC channel: #kpatch on freenode
- Contact me: jpoimboe@redhat.com

Questions?