LIVE PATCHING THE XEN PROJECT HYPERVISOR

Ross Lagerwall (Citrix)

FOSDEM 2017

Introduction

Xen supports live migration — so why live patch?

- VM is using a pass-through device.
- Live migration downtime is too long.
- To avoid host downtime.
- Not enough spare resources to move all VMs off the host.

There are several good reasons to live patch the hypervisor!

Patching basics

Inline replacement of hypervisor code would be possible but not necessarily practical.

We replace old functions with new functions instead.

Xen Live Patch has a function-level granularity.

Payloads

A payload is a set of replacement functions plus metadata.

It is packaged in a relocatable object file, like a kernel module.

Since the module is relocatable, all the symbols need to be resolved.

Xen has a simple linker to do this while loading the payload.

Plus a whole lot more: perform relocations, apply alternative instructions, and parse the replacement table, hook functions, and bug and exception frames.

Stacking payloads

Payloads are specific to a particular build of the hypervisor.

They depend on the exact hypervisor internal ABI — not stable.

The hypervisor and each live patch contains a build-id.

Each payload is designed to be applied on top of a specific build-id to prevent applying a patch on the wrong base.

This model allows loading a stack of patches.

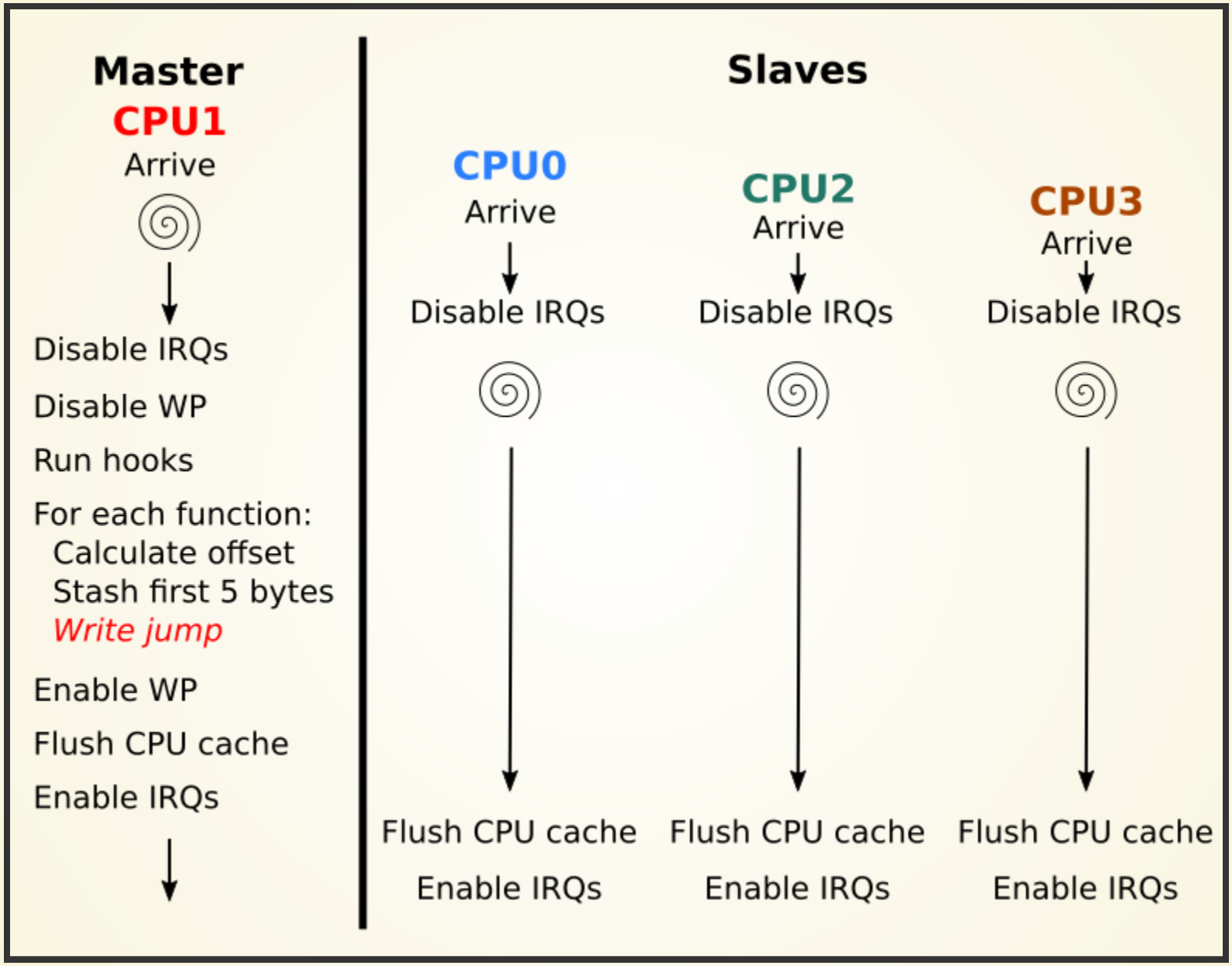
Applying payloads

A loaded payload needs to be applied to take effect.

To avoid modifying code while it is being executed, the system is quiesced.

We check in the return-to-guest path if any live patches need to be applied.

At this point, there is a fixed, small subset of functions that could be on the stack.


Master
CPU1
Arrive
Disable IRQs
Disable WP
Run hooks
For each function:
Calculate offset
Stash first 5 bytes
Write jump
Enable WP
Flush CPU cache
Enable IRQs
Slaves
CPU0
Arrive
Disable IRQs
Flush CPU cache
Enable IRQs
CPU2
Arrive
Disable IRQs
Flush CPU cache
Enable IRQs
CPU3
Arrive
Disable IRQs
Flush CPU cache
Enable IRQs

Hypervisor interface

The sysctl hypercall was extended with four new sub-operations for:

- Uploading a payload
- Listing payloads
- Querying a payload state
- Sending a command to apply, revert or unload a payload

The new sub-operations are controllable through XSM.

The xen-livepatch tool is used to access this functionality.

Compared with Linux live patching

- Latency for VMs is not usually that critical so we don't use kGraft's model.
- Xen's model is largely similar to kPatch but there is no need to perform stack checking.
- Xen does not have ftrace and is not compiled with `-pg` to get `mcount()` calls so the start of the function is overwritten.
- Can patch any function which is at least 5 bytes long.
- Overhead is a single unconditional jump.

Building live patches

How are these created?

Build them by hand? — **NO!**

Enter livepatch-build-tools!

http://xenbits.xen.org/gitweb/?p=livepatch-build-tools.git

livepatch-build-tools is based on kpatch-build

Building live patches: Inputs

```
$ livepatch-build -s xen -c orig.config \
    --depends 55776af8c7377e7191d733797543b87a59631c50 \
    -p xsa182.patch -o outdir
```

Takes as input:

- The exact source tree from the running Xen.
- The .config from the original build of Xen.
- A build-id onto which the livepatch will be applied.
- A source patch.

Building live patches: Process

livepatch-build does:

1. Build Xen
2. Apply the source patch
3. Build Xen with "-ffunction-sections -fdata-sections"
4. Revert the source patch
5. Build Xen again with "-ffunction-sections -fdata-sections"
6. Create a livepatch from the changed object files.

Building live patches: Diff

For each pair of changed objects, ‘original’ and ‘patched’, run `create-diff-tool`:

- Load objects and check that the headers match.
- Adjust the ELFs to make them easier to process.
- Correlate sections, symbols, and static locals.
- Compare and mark as SAME, CHANGED or NEW.
- For each CHANGED function or NEW global, include it and its references recursively.

Building live patches: Diff

- Handle special sections (bug frames, altinstructions, exception tables).
- For each CHANGED function, create an entry in a special livepatch section (`.livepatch.funcs`).
- Write out the new object file.
- Link each object file together into a relocatable module.

Handling live patches with data

- New data and read-only data is handled correctly.
- Changing initialized data or existing data structures is hard so such changes are prevented.
- Hook functions allow code to be executed at various stages during the patch apply (or revert) process.
  - Allows data to be transformed during patch apply, even if the data is dynamically allocated
  - Allows once-off initializations.
- Use shadow variables to attach new members to existing data structures.

Compared with Linux live patching

- Linux live patches are contained in modules.
- Xen does not support loadable modules which avoids many issues.
- kGraft does not use a tool to automatically create live patches.
- kpatch is moving towards not using kpatch-build.

Future Work

- Support for handling NMIs and MCEs while applying/reverting payloads.
- Support for signing payloads similarly to how kernel modules can be signed.
- This would make it more difficult to load a malicious module.
- Add OSStest support.
- Remove the experimental tag.