Livepatching FreeBSD kernel

Maciej Grochowski
Maciej.Grochowski[at]protonmail.com

EuroBSDcon 2018
University Politehnica of Bucharest
Bucuresti, Romania
September 22 – 23, 2018
Outline

• Problem statement
• Some background
• Why we need that?
• Existing solutions for other OSes
• FreeBSD implementation
What are we trying to achieve?

static char * get_version() {
    return VERSION;
}

Change the function inside kernel without bringing host down
Not always possible to replace the machine code

```c
@@ -60,10 +60,14 @@

    zfs_init_vattr(vattr_t *vap, uint64_t mask, uint64
{
    vap->va_mask = (uint_t)mask;
-   vap->va_type = IFTOVT(mode);
-   vap->va_mode = mode & MODEMASK;
-   vap->va_uid = (uid_t)uid;
-   vap->va_gid = (gid_t)gid;
+   if (mask & AT_TYPE)
+      vap->va_type = IFTOVT(mode);
+   if (mask & AT_MODE)
+      vap->va_mode = mode & MODEMASK;
+   if (mask & AT_UID)
+      vap->va_uid = (uid_t)uid;
+   if (mask & AT_GID)
+      vap->va_gid = (gid_t)gid;
```
1. Allocate memory for new function
2. Put new function body in allocated area
3. Calculate offset to new function
4. Put jump inside old code to new function when is safe to do
Why do we need that?! 

- Apply fixes to severe issues quickly and without planning downtime
  Stability and Security fixes

- Large in-memory databases or FS Cache
  Savings as re-reading data from disk can take hours

- Datacenters and large on-prem deployments
  Rebooting thousands or tens of thousands of machines in a controlled way without affecting business can be hard
  Users don’t want a downtime but want to be patched and secure
Other Live patching techniques

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Solution name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>kSplice (Oracle)</td>
</tr>
<tr>
<td></td>
<td>Kpatch (Red Hat)</td>
</tr>
<tr>
<td></td>
<td>kGraft (SUSE)</td>
</tr>
<tr>
<td></td>
<td>Linux Livepatching (upstream solution)</td>
</tr>
<tr>
<td>Xen</td>
<td>Xen Livepatch</td>
</tr>
<tr>
<td>AIX</td>
<td>Live Update</td>
</tr>
</tbody>
</table>
Linux livepatching

- Using ftrace to take over control (insert call instruction)
- Kernel compiled with –pg flag
- Kernel Linker involved to put replacement function
Xen livepatch

- Xen didn’t have a kernel linker
- Replacement function hooked by livepatch
- 5 bytes required to put trampoline (4 on ARM)

```
<arch_do_domctl>[NEW]:
  55  push %rbp
  48 89 e5  mov %rsp,%rbp
  41  57  push %r15
...
  48 c7 45 b8 00 00 00 00  movq $0x0,-0x48(%rbp)
  48 c7 45 c0 00 00 00 00  movq $0x0,-0x40(%rbp)
  48 c7 45 c8 00 00 00 00  movq $0x0,-0x38(%rbp)
  48 89 e0  mov %rsp,%rax
  48 25 00 80 ff ff  and $0xfffffffffffffff8000,%rax
```
AIX Live update

- Require additional disk
- Create Surrogate Logical partition
- Migrate existing processes to Surrogate
Consistency model STOP

- **Patching Process**: Add Ftrace entry
  - **Process1**: Call old func1
  - **Process2**: Call old func2
  - **Process3**: Call old func2

- **Call Stop Machine**: Safeness Check, Update hash table
  - Walk through the all Thread and check Old func2 on stacks

- **Return**: Call New func1, Call New func2

All running process and interrupts are stopped

Now switch to New functions
**XEN STOP MACHINE**

Spin until

Semaphore count is 3

Set semaphore to 0

set->ready to 1

Spin until

semaphore=3

· Disable IRQs
· Disable WP
· Patch!
· Enable WP
· Flush CPU pipeline
· Enable IRQs

→ do_work set to 0

IPI all other CPUs, causing them to VMEXIT

Each increments semaphore and spins until →ready is set (1)

Each increments semaphore disable IRQs

Spin until

→do_work is cleared (0)

Each CPU flushes its CPU pipeline Restores IRQs, And resumes guests

Time (from IPI to patching timeout is set to 30 ms)
Incremental UPDATE
FreeBSD “Hot patching”

- Using DTrace to take over execution control
- Hot-patch is DTrace provider, using similar technique to FBT
- Stop Machine approach to achieve consistency
Hot patching Design Principles

- Be useful for the community
- Touch as less as possible other kernel components
  Especially stuff like DTrace and kernel linker
- Reuse as much as possible from existing solutions
- Provide solution to allow patch all the code that provide functions
Open questions

- Highly dependent on DTrace framework
  Advantage or disadvantage?
- Possible need of changing module loader code in order to use local symbols
- Additional security mechanisms avoid failure at any cost
- Is performance impact significant?
Conclusions

- Security and stability fixes are common reason of scheduling servers updates/downtime
- Users can get benefit by patching the system without a downtime
- Live patching is common technique used by other kernels
- FreeBSD kernel did not implemented this feature so far
- Initial implementation based on common known practices, community feedback required, not fully functional yet.
Resources

[1] A design proposal for Xen hotpatching Martin Pohlack 2014-10-17


[4] kpatch Have your security and eat it too! Josh Poimboeuf LinuxCon North America August 22, 2014

[5] kGraft Live patching of the Linux kernel


[7] Linux livepatching kernel documentation

Illustration Credit: @FableMode
Q&A