xSplice

Runtime patching hypervisor

Konrad Rzeszutek Wilk
Oracle
Software Development Director

ORACLE

Agenda:

- Why would you want this?
- Other known patching techniques.
- Patching!
- Tiny details.
- Roadmaps.
- Call to developers!

Why not migrate to another host?

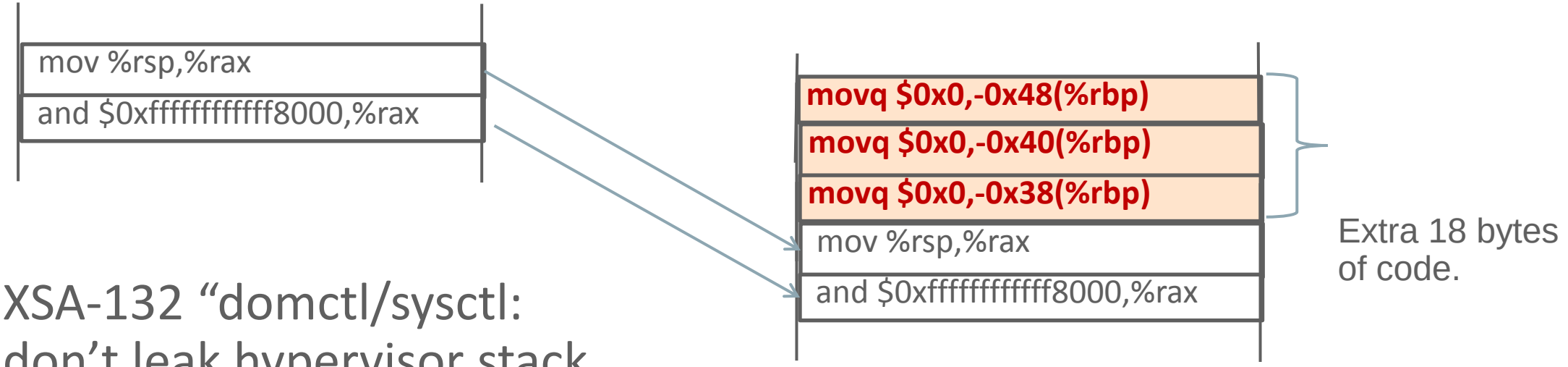
- Local storage (SATA?),
- PCI pass-through (SR-IOV),
- NUMA locality,
- Giant guests (memory or CPU) and cannot fit on other hosts,
- Or system administrator simply does not want to reboot host.

Known patching techniques.

- On Linux:
  - kGraft (SuSE).
  - kPatch (Red Hat).
  - kSplice (Oracle).
  - Linux hot-patching (upstream) – merge of kGraft + kPatch.
- On Xen:
  - Amazon's hotpatching design:
    - http://www.linuxplumbersconf.net/2014/ocw//system/presentations/2421/original/xen_hotpatching-2014-10-16.pdf
  - Oracle's - borrowing ideas/concepts from kSplice git tree (pre acquisition):
    - http://lists.xen.org/archives/html/xen-devel/2015-07/msg04951.html

Patching!

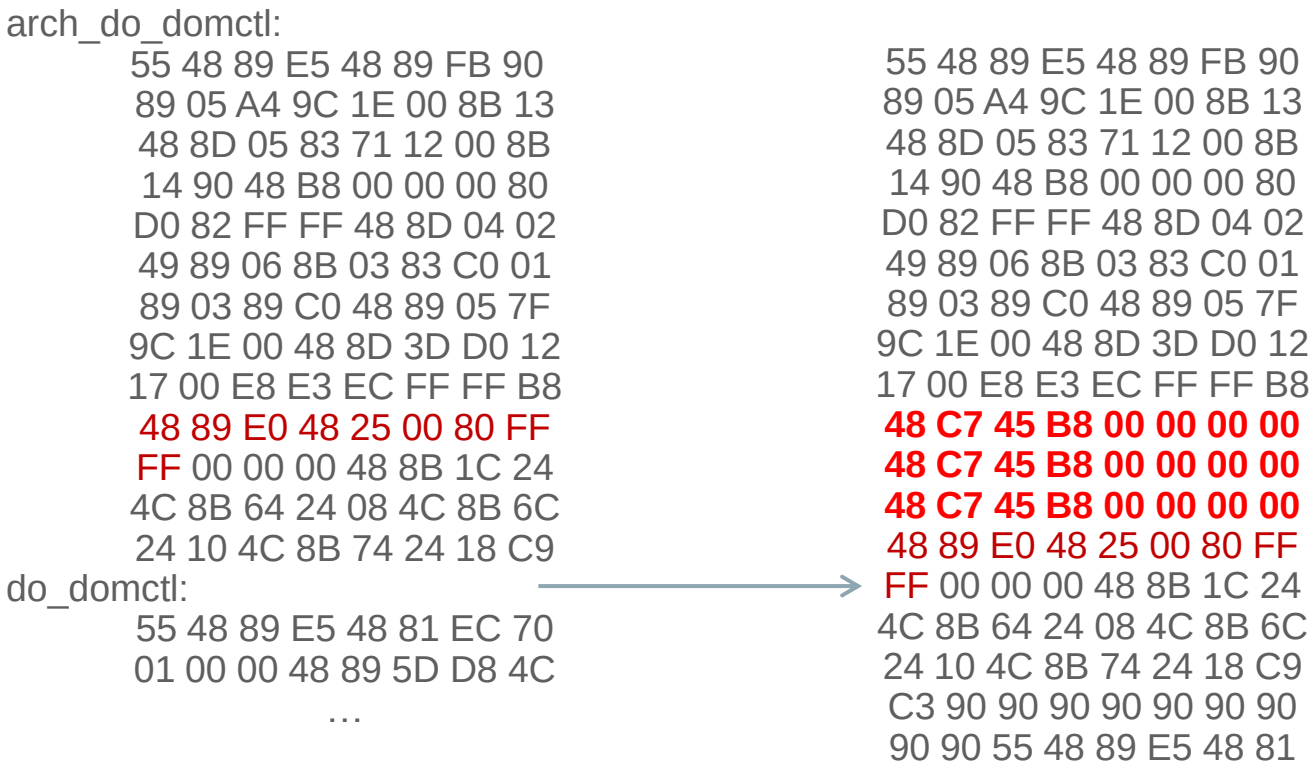
- At first blush this sounds like binary translation – we convert old code to new code:



- XSA-132 “domctl/sysctl: don’t leak hypervisor stack to toolstack” – change inside arch_do_domctl.
- But nobody can translate the code for us. We NEED to change the code in memory while the hypervisor is executing.

Patching: inserting new code.

- But adding in code means moving other code as well:



- Otherwise we end up executing nonsense code at old location!

Patching: Jumping

- We could add padding in all the functions to deal with this. But what if the amount of changes > padding?
- Jump!
  - Allocate new memory.
  - Copy new code in memory.
  - Check that nobody is running old code.
  - Compute offset from old code to new code.
  - Add trampoline jump to new code.

Patching: 1) Allocate + copy new code in

- New **arch_do_domctl** code at newly allocated memory space:

```
<arch_do_domctl>:

    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 $0xffffffffffff8000,%rax
```

Patching: 2) Check code 3) Compute offset

- Check that arch_do_domctl is not being executed.
- Figure out offset from new to old code.

```
<arch_do_domctl>:

    55                      push   %rbp
    48 89 e5                mov    %rsp,%rbp
    41 57                   push   %r15
…
48 89 e0                    mov %rsp,%rax
48 25 00 80 ff ff           and $0xffffffffffff8000,%rax
```

```
<arch_do_domctl>:

    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 $0xffffffffffff8000,%rax
```

Patching: 4) Add trampoline

- Add trampoline:

```
<arch_do_domctl>:

    E9 1A 97 EA FF       jmpq   <arch_do_domctl>[NEW]
…
48 89 e0                 mov %rsp,%rax
48 25 00 80 ff ff        and $0xffffffffffff8000,%rax
```

```
<arch_do_domctl>:

    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 $0xffffffffffff8000,%rax
```

Patching: Conclusion

- For code just need to over-write start of function with:

```
…
E9 1A 97 EA FF      jmpq   <arch_do_domctl>[NEW] …
```

- For data it can be inline replacement (changing in .data values):

```
<opt_noreboot>:
   00 00
    ...
```



```
<opt_noreboot>:
   00 01
    ...
```

That was easy, what is the fuss about?

- Relocation of symbols – data or functions:

```
…
8b 0d 53 80 fb ff       mov    -0x47fad(%rip),%ecx        # ffff82d0802848c0 <pfn_pdx_hole_shift>
…
```

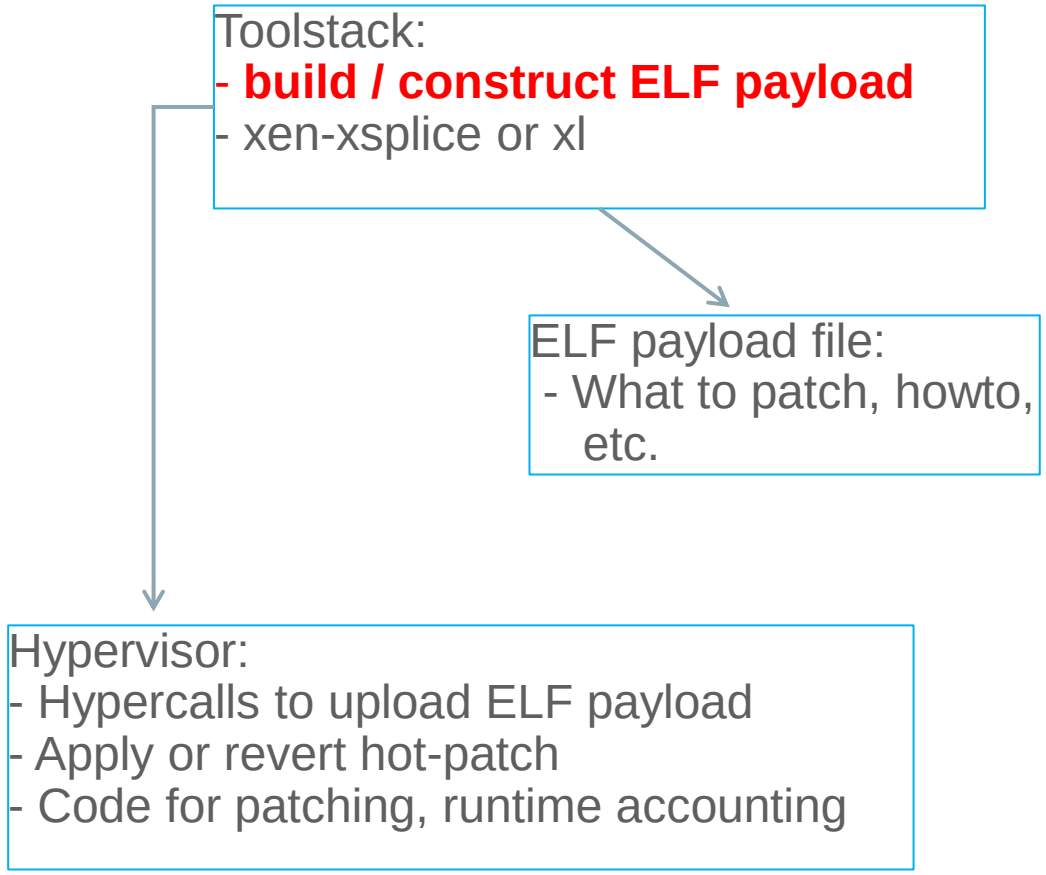
  Need to compute of new code/data the offsets to other functions, data structures, etc. (ELF dynamic linker).

- Correctness: Is the old code the same as what the hot-patch had been based on?
- Stack checking: Cannot patch the function which is in use by another CPU!
- Reverting the hot-patch (updated XSA?).
- Dependency off one hot-patch on another.

Other runtime concerns:

- When to patch:
  - Hypervisor has no threads. Only guests.
  - Can at VMEXIT where the stack is empty – good deterministic point.
  - Use Amazon's global barrier with timeout at VMEXIT point with abort and retry?
  - But what if only running PV guests? Need to be low enough in stack to have the minimum amount of code on the stack – and there is no VMX code.
- Other CPUs
  - Can use IPI for all the other CPUs and all of them can come together at about the same point.

Component: build



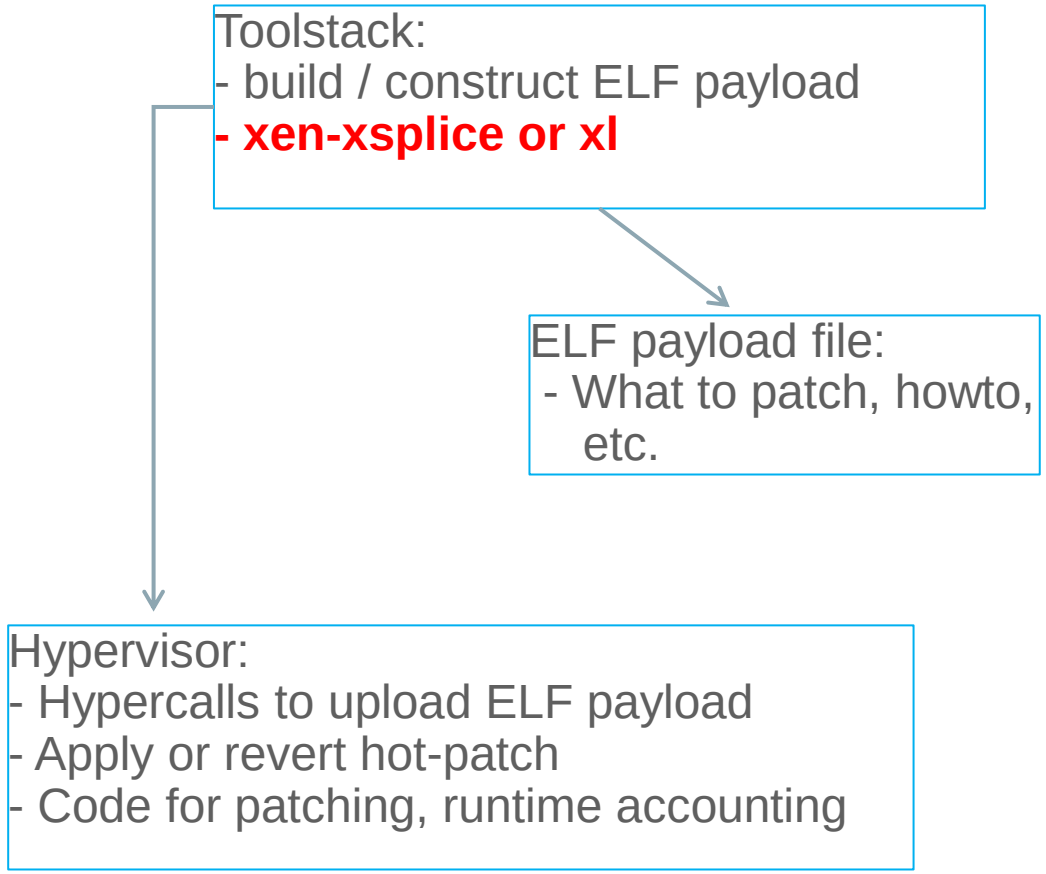
Build-time concerns

- MUST use same gcc when preparing payloads.
- Can use linker **build-id** to create payloads specific to a built-hypervisor, or can have the old code (or data) as part of the payload to compare against.

Build process:

- Source code, build artifacts (.o files) and unmodified hypervisor – baseline.
- Apply patch, build. New objects have a different name (.o.XSPLICE)
- Extract functions – compare and find changed ones.
- Which sections (.text, .data), and what offsets are different, how it should be patched.
- Create an ELF payload with telemetry data, new code and data. Also include **build-id** or old code (or data).
- Apply signature to payload so hypervisor can verify it.

Component: tools



Tool side functionality:

- Query what hot-patches have been loaded and their status (applied, verified, reverted).
- Upload new hot-patches.
- Verify hot-patch – signature and build-id (off old code or data).
- Apply, revert, or unload hot-patch.

Screenshot of xen-xsplice:

```
-bash-4.1# xen-xsplice upload "XSA 131" xsa131.xsplice
Uploading xsa131.xsplice (253 bytes)
-bash-4.1# xen-xsplice list
 ID                                      | status
-----------------------------------------+------------
XSA 131                                  | LOADED
-bash-4.1# xen-xsplice check "XSA 131"
XSA 131: State is 0x1, ok are 0x5. Commencing check:completed!
-bash-4.1# xen-xsplice apply "XSA 131"
XSA 131: State is 0x4, ok are 0x14. Commencing apply:completed!
-bash-4.1# xen-xsplice list
 ID                                      | status
-----------------------------------------+------------
XSA 131                                  | APPLIED
-bash-4.1#
```

Roadmap for tools:

- **xen-xsplice** is low-level (RFC). Integrate the code into **xl.**
- From **Build** slide: “Which sections (.text, .data), and what offsets are different, how it should be patched.”
  - Difficult problem.
  - Objdump and objcopy magic.
  - kSplice tools used as aspiration:
    - http://repo.or.cz/w/ksplice.git

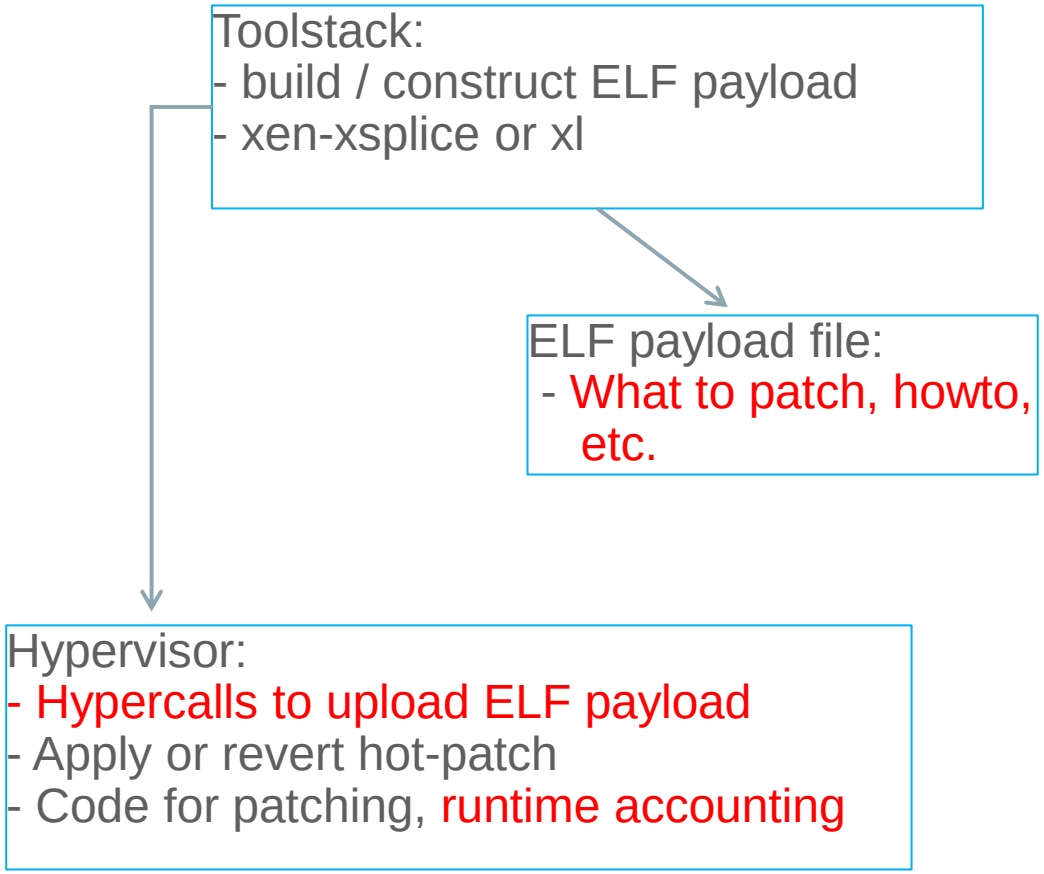
Roadmap for tools – regression tools.

- For v1 – tools for creating hot-patches to patch xen_extra:

```
-bash-4.1# xl info | grep xen_extra
xen_extra              : -unstable
-bash-4.1# xen-xsplice upload regression-test ./regression-test.ksplice
Uploading ./regression-test.ksplice (1171 bytes)
-bash-4.1# xen-xsplice check regression-test
regression-test: State is 0x1, ok are 0x5. Commencing check:completed!
-bash-4.1# xen-xsplice apply regression-test
regression-test: State is 0x4, ok are 0x14. Commencing apply:completed!
-bash-4.1# xl info | grep xen_extra
xen_extra              : -Hello_World
-bash-4.1#
```

(Disclaimer: Doctored image).

Components: hypervisor and ELF payload.



Roadmap – design (and prototype code) on hypervisor level and ELF payload.

- RFC v3.1 posted: http://lists.xen.org/archives/html/xen-devel/2015-07/msg04951.html
  - ELF payload structures (what section to patch, how to patch, safety data).
  - Signature verification.
  - State diagrams for hot-patch life-cycle.
  - Hypercall API – four new hypercalls: UPLOAD, LIST, GET, and ACTION.
  - Tons of implementation gotchas.
  - RFC hypercall code include (no patching, just accounting).
- Birds of Feather discussion (BoF 2) August 18 @11:30 – 12:20 to nail down some questions raised on xen-devel mailing list.

ORACLE

Roadmap – Further work in hypervisor:

- Need in hypervisor ‘dl_sym’ functionality (dynamic ELF linker).
- Computing offset – symbol table calculation.
- Exception table growth.
- Patching and reverting code (and data): inline and trampoline.
- Signature verification code.
- VMEXIT resume call to patching code.
- Other issues as development continues along.

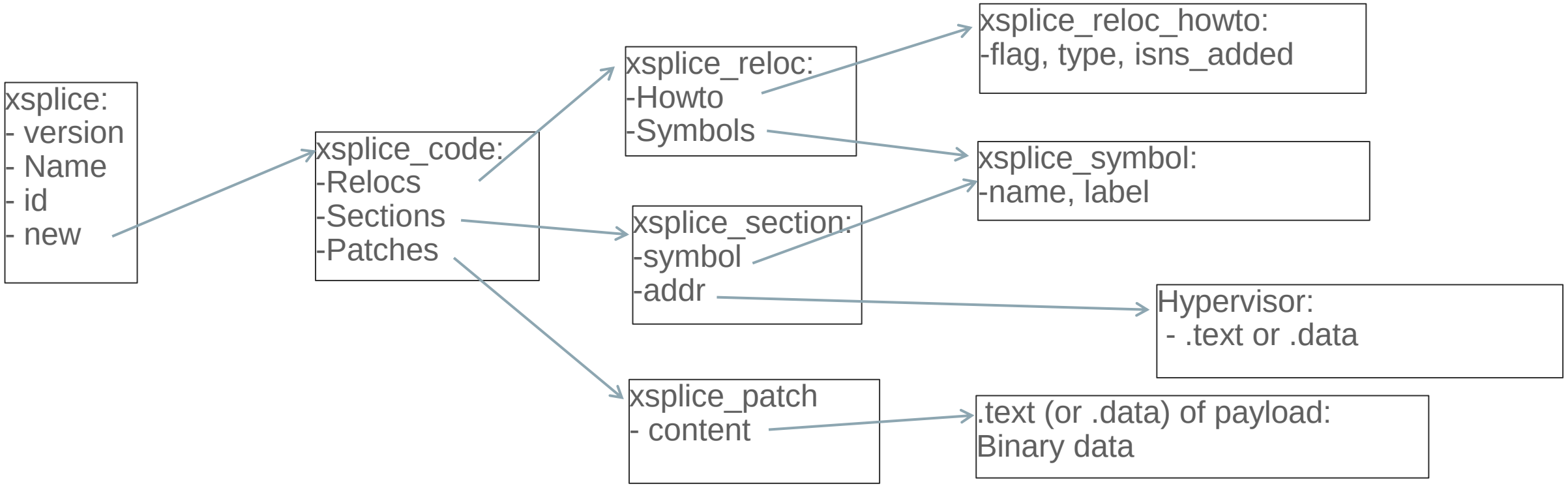
Call to action!

http://wiki.xen.org/wiki/XSplice sign up for what you would be interested in helping with!

Questions and Answer

ELF payload:

- .xsplice_* sections which map to structures.
- .text, .bss, .symtab – included if the .xsplice_* refer to them.



Signature verification:

- The signature is to be appended at the end of the ELF payload prefixed with the string: ~Module signature appended~\n
- Signature header afterwards matches Linux's one.