Paper Reading Group

Ksplice: Automatic Rebootless Kernel Updates

Eurosys 2009 Paper
by
Jeff Arnold and M. Frans Kaashoek

Use Case

- Reboots are risky

  → Security updates are postponed
- Security patches often touch only few lines of code
- How about patching this code *while it runs*?

Current Approaches

- require specially written software
- constrain mechanisms software can use
- can only update kernels / software that had this feature in mind to begin with
- are constrained to specific programming languages
- Need laborious adaptation work to make patches runtime applicable.

How About Something New

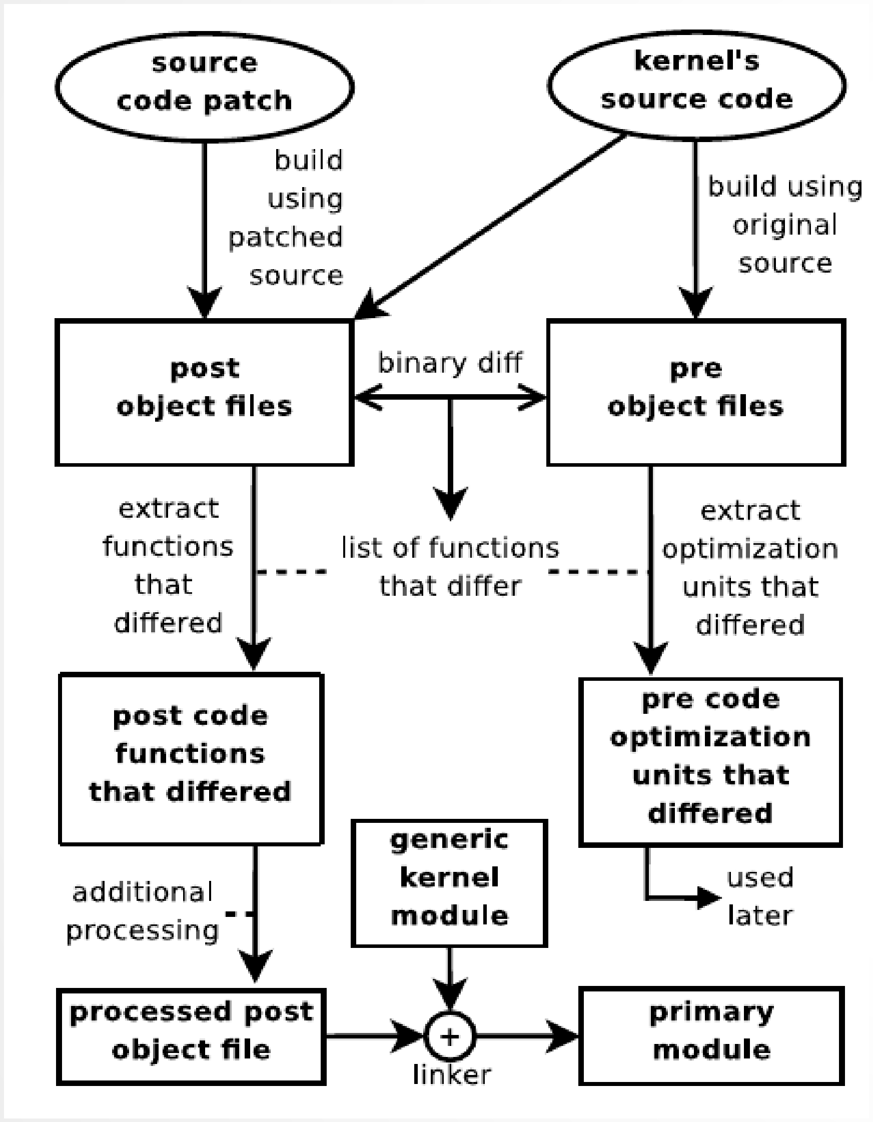
- Binary diffs and patching of the changed code paths only!
  - Does not require changes in to original binary
  - Is not constrained on specific programing languages

Challanges

- resolve symbols in replacement code
- find changes resulting from patch in replacement object code
- avoid obfuscation by compiler optimization
- avoid inter code jumps making function isolation difficult
- ensure safety of code replacement (avoid crashes)

  → avoid missing inlined code leading to an inconsistent binary

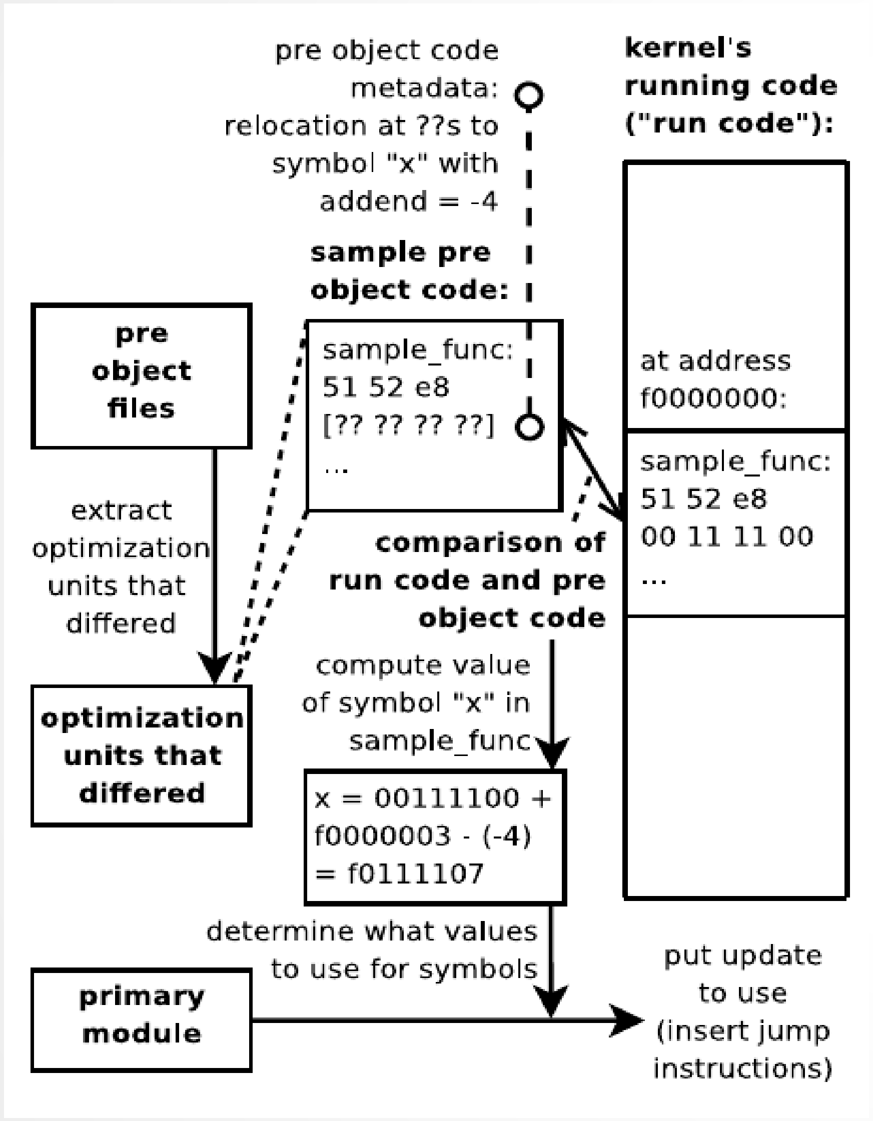
pre-post differencing



source code patch
kernel's source code
build using patched source
build using original source
post object files
binary diff
pre object files
extract functions that differed
list of functions that differ
extract optimization units that differed
post code functions that differed
pre code optimization units that differed
generic kernel module
used later
additional processing
processed post object file
linker
primary module

- Compile both version (patched and unpatched)
- each function within its own data and function section in the binary
- compare sections to find changed functions
- but what about symbol resolution?

run-pre matching


pre object code metadata: relocation at ??s to symbol "x" with addend = -4
kernel's running code ("run code"):
sample pre object code:
pre object files
sample_func: 51 52 e8 [?? ?? ?? ??] ...
at address f0000000:
sample_func: 51 52 e8 00 11 11 00 ...
extract optimization units that differed
comparison of run code and pre object code
compute value of symbol "x" in sample_func
optimization units that differed
x = 00111100 + f0000003 - (-4) = f0111107
determine what values to use for symbols
primary module
put update to use (insert jump instructions)

- Use kallsyms symbol table
- problem for ambiguous symbols
- need to know no-ops (used as function alignment padding, does not imply difference!)
- must know instructions with relative addressing so that different rel. addresses pointing to the same location are not identified as differences.

Ksplice Components

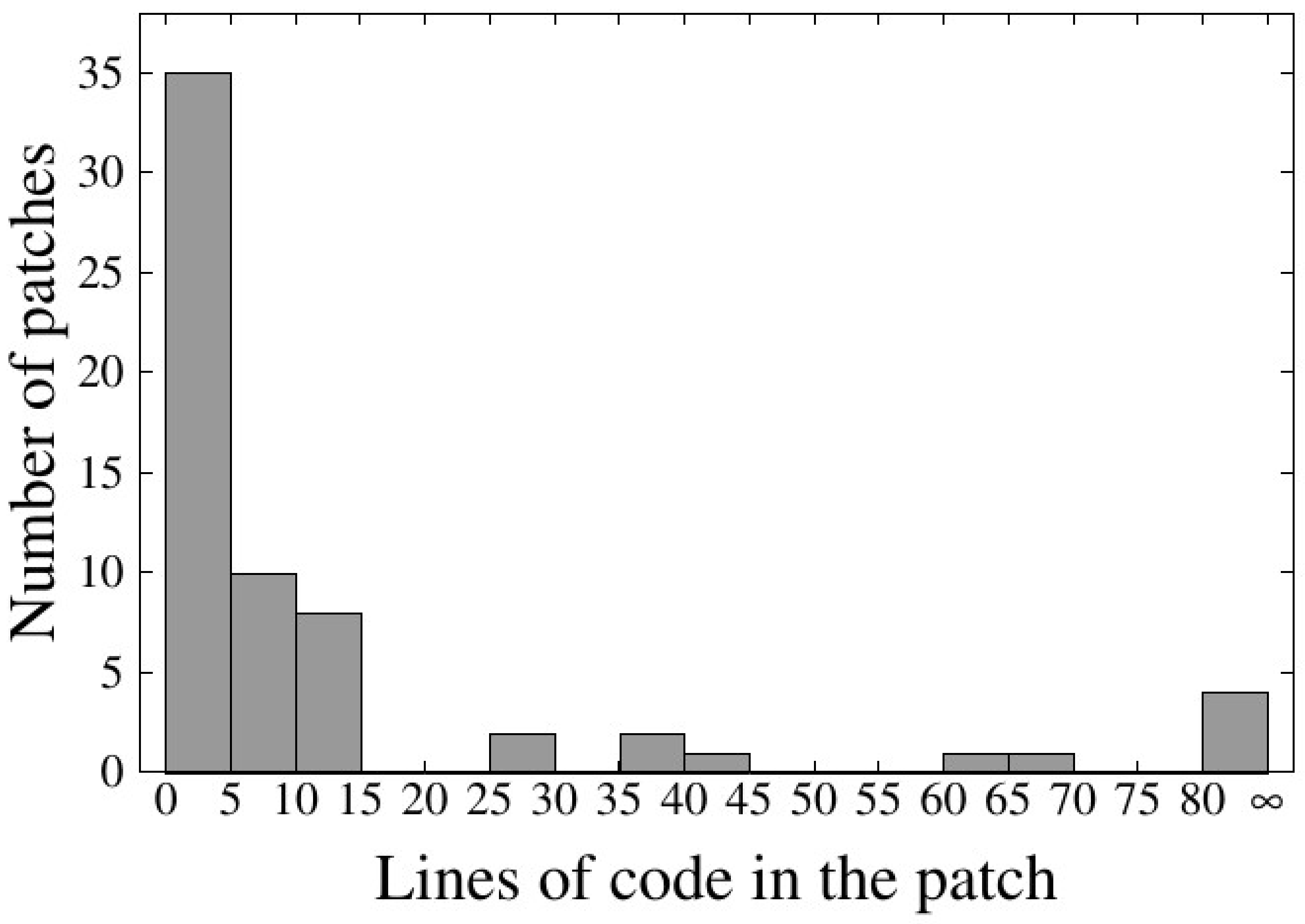
- Ksplice core kernel module (preforms run-pre matching)
- pre/post object code generator in userspace
- helper module for loading pre-code (for comparison)
- primary module (loads new functions)

Quiescent State

- Functions can only be replaced when:
  - they are on no threads kernel stack
  - no threads fp is in the function
- Retry on failure, give up after threshold
- stop_machine takes about 0.7 ms to execute

Evaluation

Figure 3: Number of patches by patch length



Conclusion

- Found a way to patch arbitrary software during runtime
- With few limitations
- independent of programing language
- Safe (checks performed, does abort if assumptions not met)
- Can even cope with changes to data structures
- Eliminates reboots for kernel security updates

Discussion Points

- How about jumps into functions? How are they covered?
- How do they compare *pre* and *run* code?
- Evaluation: 0.7 ms for how many threads?