Shredding Your Garbage

Reducing Data Lifetime Through Secure Deallocation

Jim Chow, Ben Pfaff, Tal Garfinkel, Mendel Rosenblum

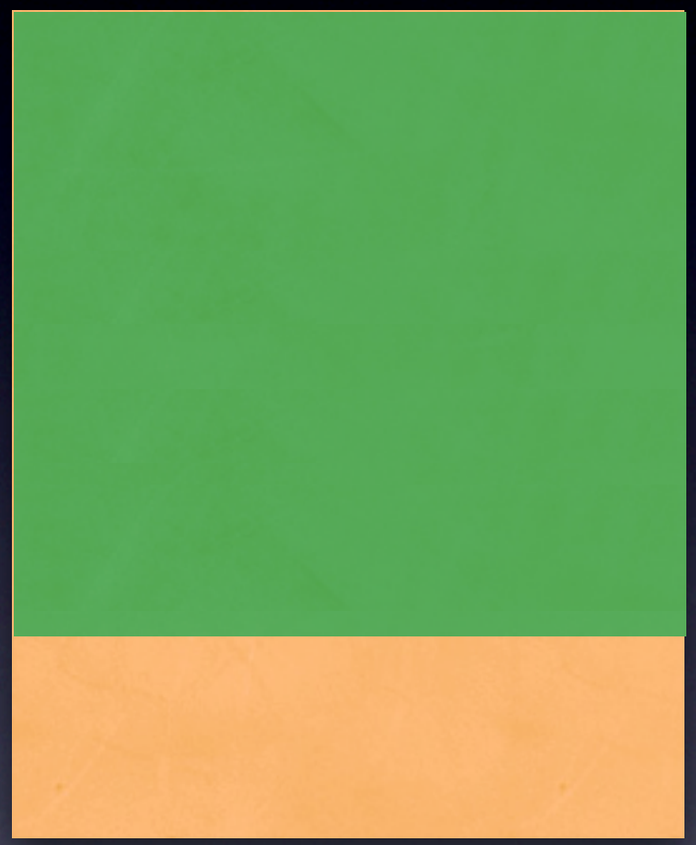
presented by Michael Roitzsch

Problem

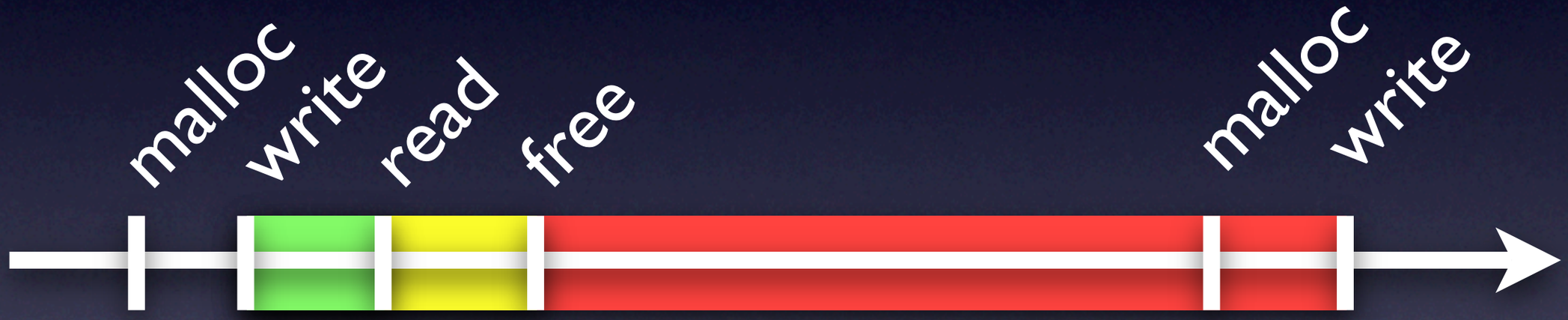
Most applications take no steps to minimize the amount of sensitive data in memory.

- scrubbing memory is used only for credentials like keys or passwords
- data may spill out via core dumps, logs, swapping, driver buffers, window manager
- most of this is outside application control

Holes



Data Life Cycle

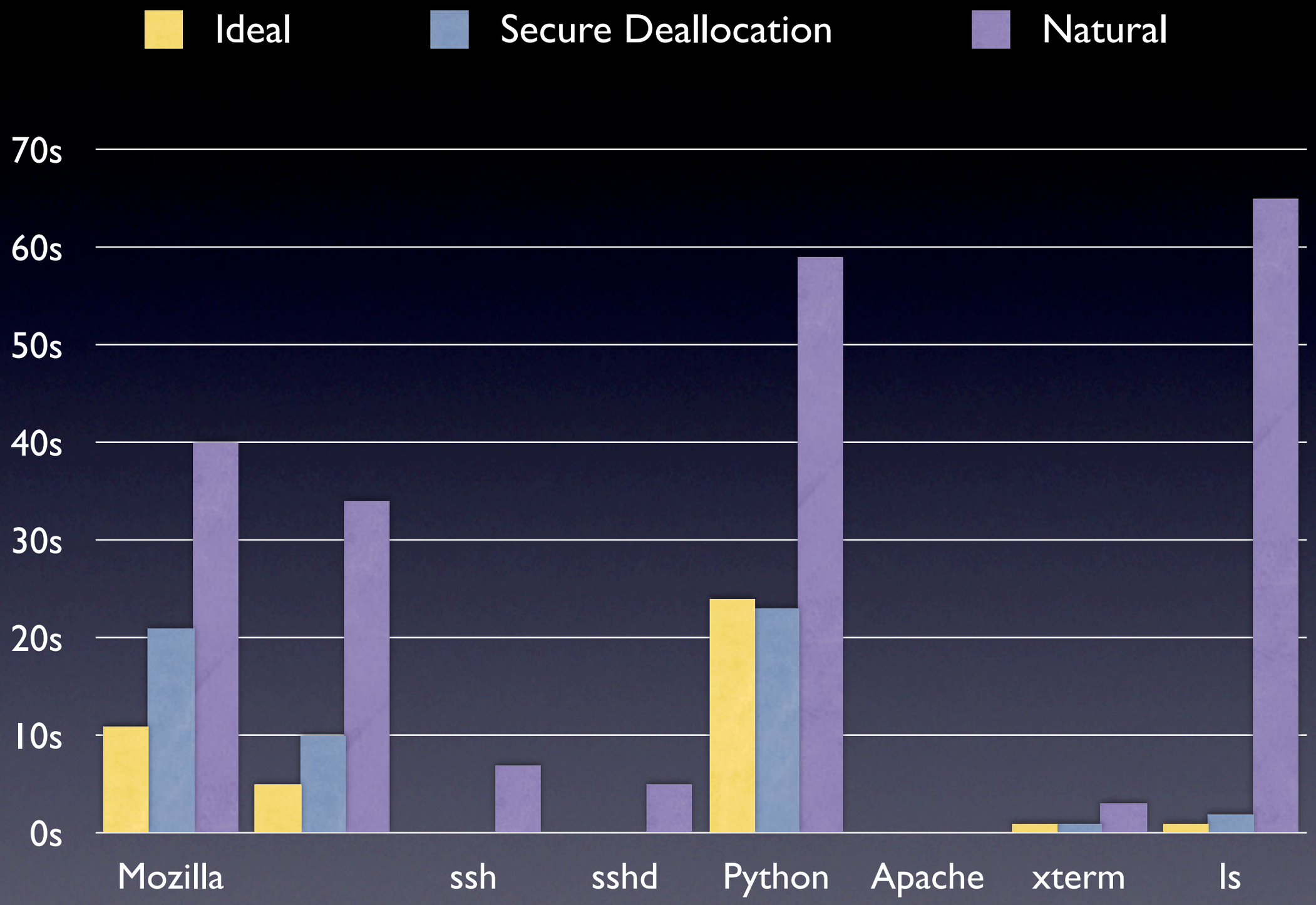


Secure Deallocation

- scrub on deallocate as a conservative heuristic
  - time of last use is generally unknown
  - should not introduce errors
- requires no a priori knowledge about data use in an application

Layered Clearing

- applicable at many levels
  - application-level
  - compiler-level
  - library-level
  - kernel-level
- clearing only on one level is insufficient


Ideal
Secure Deallocation
Natural
70s
60s
50s
40s
30s
20s
10s
0s
Mozilla
ssh
sshd
Python
Apache
xterm
ls

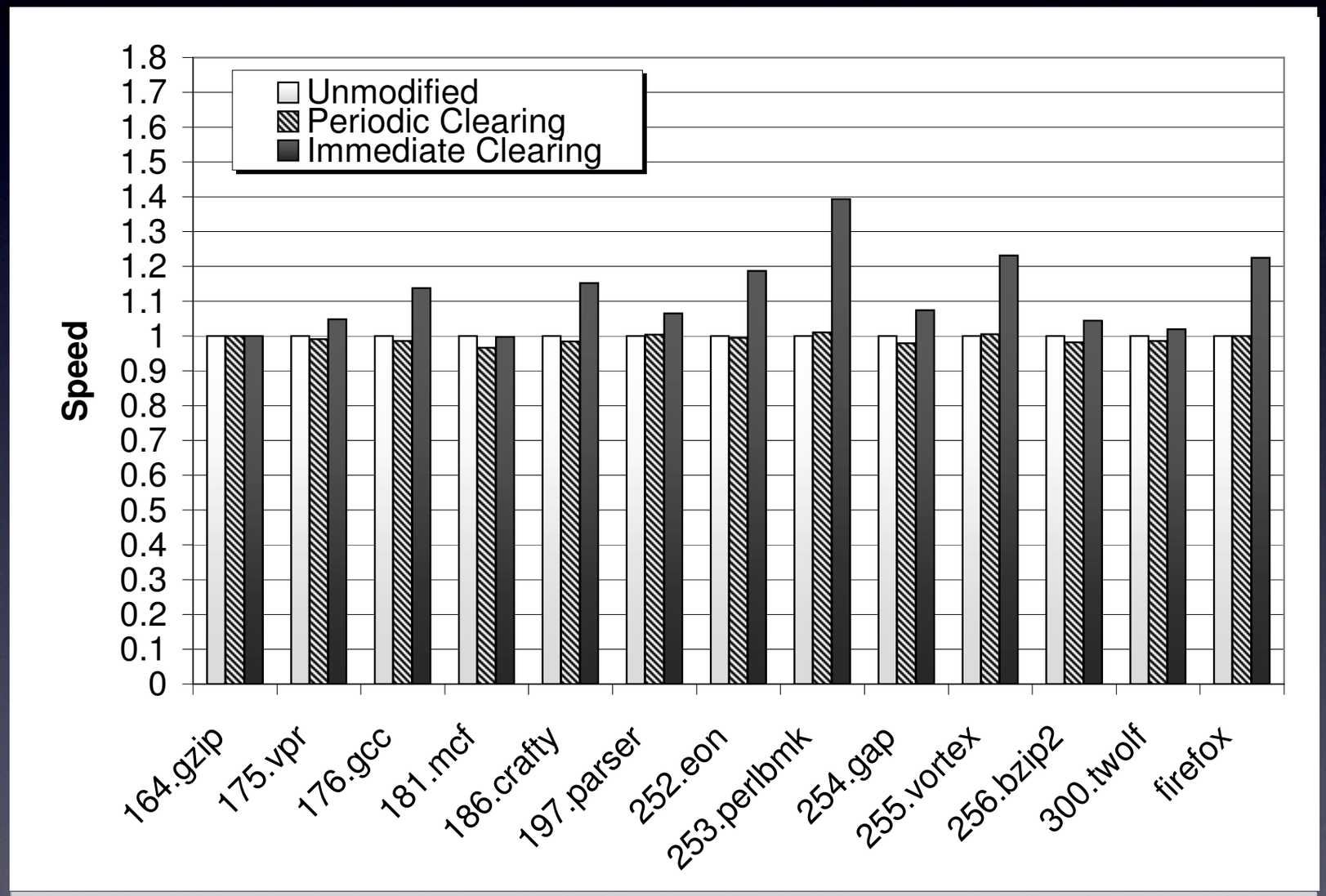
Kernel Clearing

- free pages divided into three pools
  - not-zeroed pool
  - zeroed pool
  - polluted pool
- zeroing daemon ensures pages do not stay in polluted pool longer than 5 seconds

Results

- Apache & Perl workload
  - CGI script checks a password
    - all tainted memory is cleared with some benign exceptions
- Emacs workload
  - all taints cleared except entropy pool

Performance Impact



Caveats

- some data is never deallocated
  - long-living data
  - memory leaks
- custom allocators

Discussion

- There is still a gap to the ideal lifetime.
- The data is still in memory while it is used.
- There is always a window of opportunity.
- You have to trust your memory anyway.
- Why bother shrinking the window when you can never close it for good?