Paper-Reading-Group

Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation

Nathan Dautenhahn, Theodoros Kasampalis, Will Dietz, John Criswell, and Vikram Adve

Nested Kernel - Motivation

- Monoliths have single large TCB
- How to seperate into multiple protection domains?
  - Microkernels require complete redesign of kernel
  - VMMs have high performance overhead

How can we provide protection domains without the overhead using the existing code base and design principles?

Nested Kernel - Idea

- Use the MMU to isolate the MMU
  - *Nested Kernel* is small and protects MMU structures
  - *Outer Kernel* is de-priviledged and only has checked access to MMU structures
- Keep the monolithic address space
- While still enabling application of intra-kernel security policies
  - Example policies: write logging, write-mediation

Nested Kernel - Design

- Separate policy from mechanism (MMU)
- OS Co-design for security policies
- MMU based privilege separation
- Fine grained resource control
- Negligible performance impact

Nested Kernel - Invariants

Invariant 1:

Active virtual-to-physical mappings for protected data are configured read-only while the outer kernel executes.

Invariant 2:

Write-protection permissions in active virtual-to-physical mappings are enforced while the outer kernel executes.

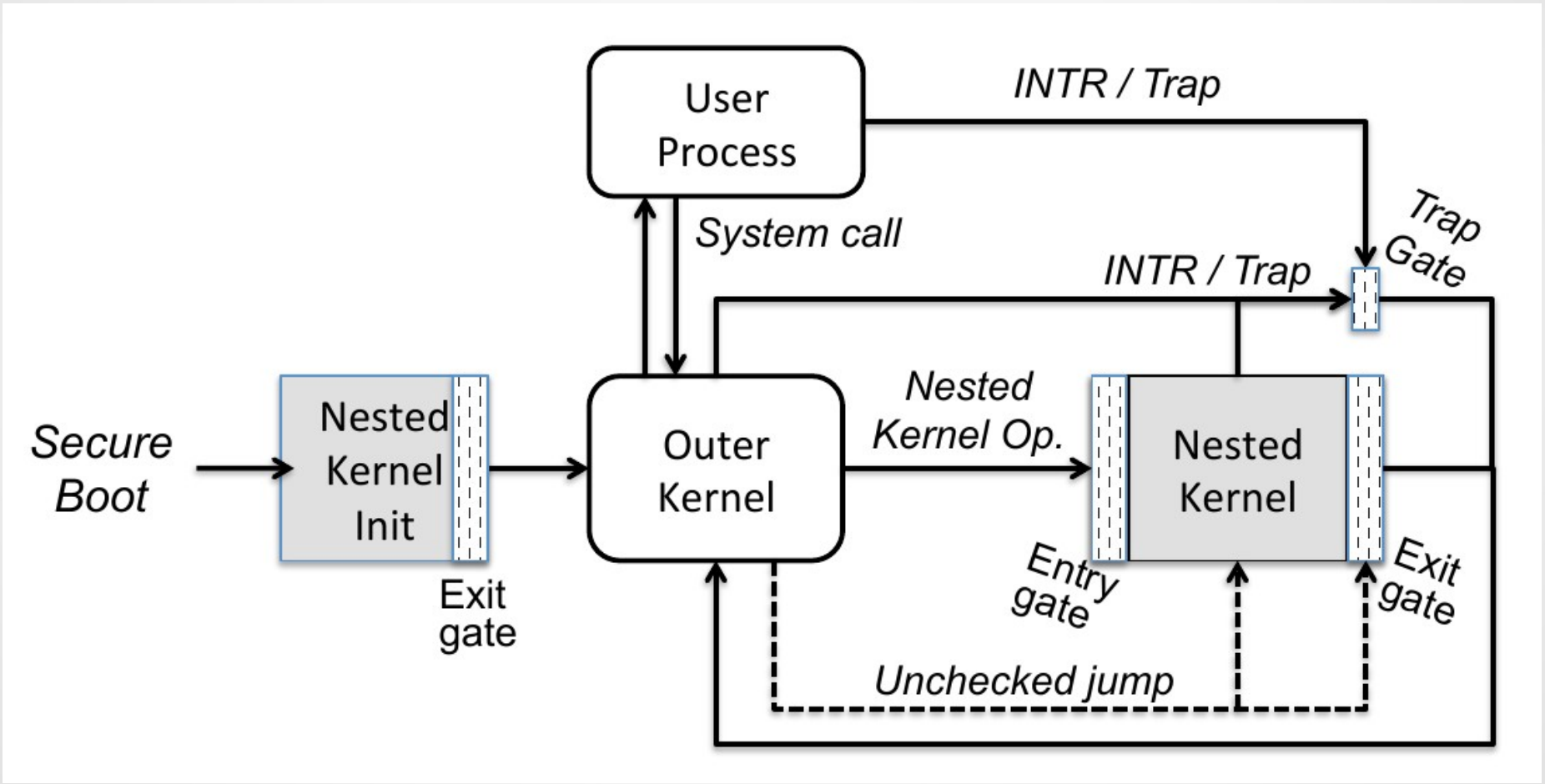
Nested Kernel – write protection

Function	Selected Arguments	Purpose
`nk_declare`	`mem_start, size, mediation_func`	Marks all pages RO; initializes an NK write descriptor `nk_wd`; returns the `nk_wd` and the pointer to the region.
`nk_alloc`	`size, mediation_func, nk_wd_p`	Allocates memory region; invokes nk_declare on it; stores write descriptor in `nk_wd`; returns `nk_wd` and pointer to the region.
`nk_free`	`nk_wd`	Deallocates memory identified by `nk_wd`. Memory must have been allocated by `nk_alloc`. Freed pages can be reused only by a future `nk_alloc`.
`nk_write`	`dest, src, size, nk_wd`	Verifies write bounds; invokes `mediation_func`, if any; then copies `size` bytes from `src` to `dest`.

PerspicuOS – Thread Model

- *Outer kernel* may be under complete attacker control
  - Can attempt to arbitrarily modify CPU state
  - Can modify outer kernel source code
  - Can modify control flow
- *Nested kernel* source code and binary are trusted
  - Including the mediation functions
- Hardware is free of vulnerabilities
- Do not protect against hardware attacks

PerspicuOS – Architecture



PerspicuOS – Invariant 1 Support

Reminder: **Invariant 1:**

Active virtual-to-physical mappings for protected data are configured read-only while the outer kernel executes.

Invariant 3:

Ensure that there are no unvalidated mappings prior to outer kernel execution

Invariant 4:

Only declared PTPs are used in mappings

Invariant 5:

All mappings to PTPs are marked read-only

Invariant 6:

CR3 is only loaded with a pre-declared top-level PTP.

PerspicuOS – Invariant 1 Support

Remin

Wri

whi

Invaria

Invaria

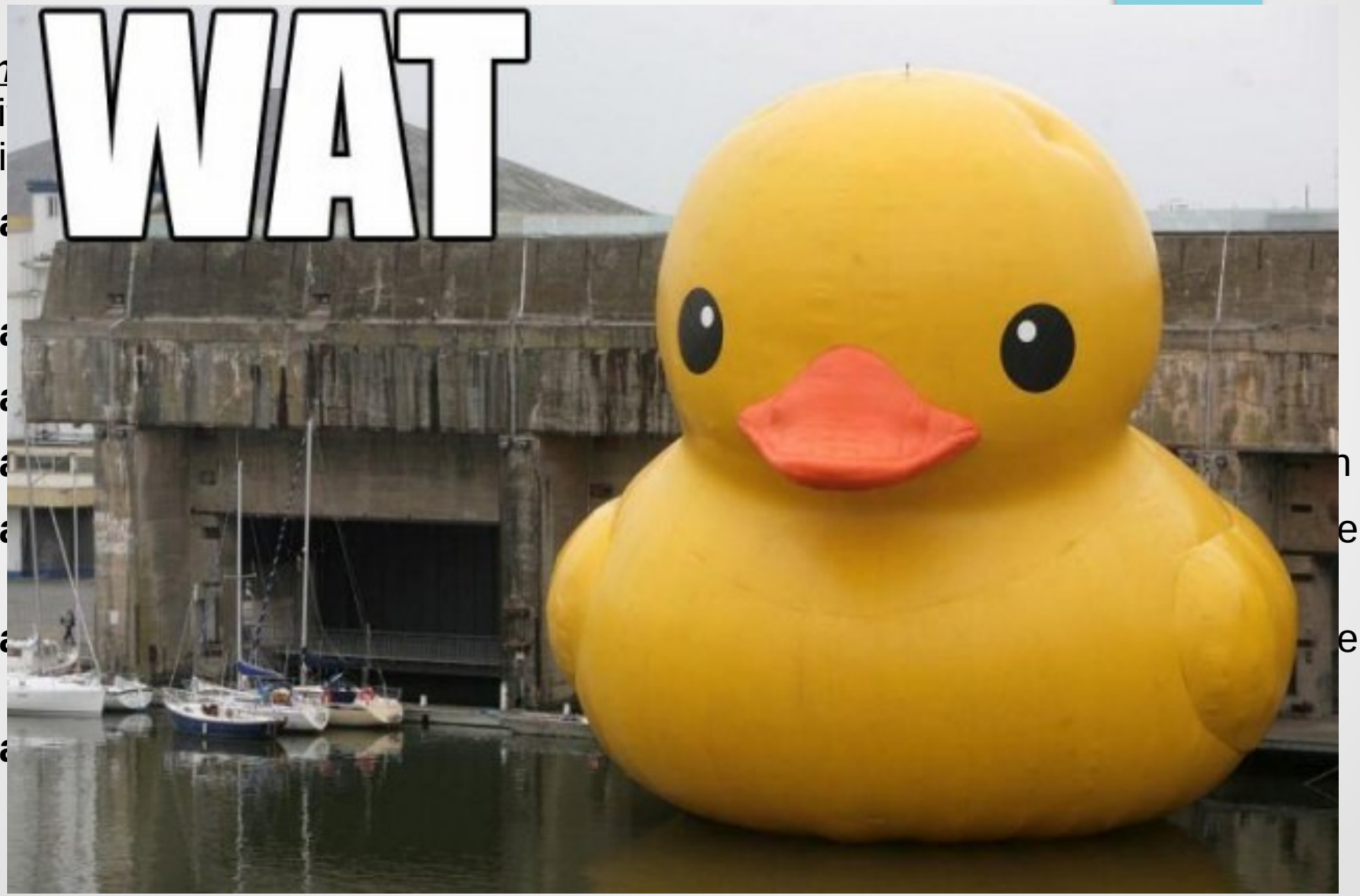
Invaria

Invaria

Invaria

Invaria

Invaria



PerspicuOS – Entry & Exit

```
entry:
  pushfq                          Save current flags
  cli                             Disable interrupts
  mov %rax, -8(%rsp)              Spill regs for temps
  mov %rcx, -16(%rsp)
  mov %rsp, %rcx                  Save stack ptr in rcx
  mov %cr0, %rax                  Get current CR0 value
  and ~CR0_WP,%rax                Clear WP bit in copy
  mov %rax, %cr0                  Write back to CR0
  cli                             Disable interrupts
  mov PerCPUSecureStack,%rsp      Switch to secure stack
  push %rcx                       Save orig stack ptr
  mov -0x8(%rcx), %rax            Restore spilled regs
  mov -0x10(%rcx), %rcx
```

```
exit:
  mov 0(%rsp), %rsp               Restore orig stack ptr
  push %rax                       Spill scratch reg
  mov %cr0, %rax                  Get current CR0 value
1:
  or CR0_WP, %rax                 Set WP in CR0 copy
  mov %rax, %cr0                  Write back to CR0
  test CR0_WP, %eax               Ensure WP set
  je 1b                           If not, loop back
  pop %rax                        Restore clobbered reg
  popfq                           Restore flags
                                  (incl interrupt status)
```

PerspicuOS - Evaluation

Example Mediation Functions:

- Write-only data
- Append-only data
- Write logging

Privilege Boundary	Time (μsecs)	Time / NK Call
NK Call	0.1390	1.00x
Syscall	0.08757	0.62x
VMCALL	0.5130	3.69x

Table 3. Privilege Boundary Crossing Costs.

PerspicuOS – Evaluation (2)

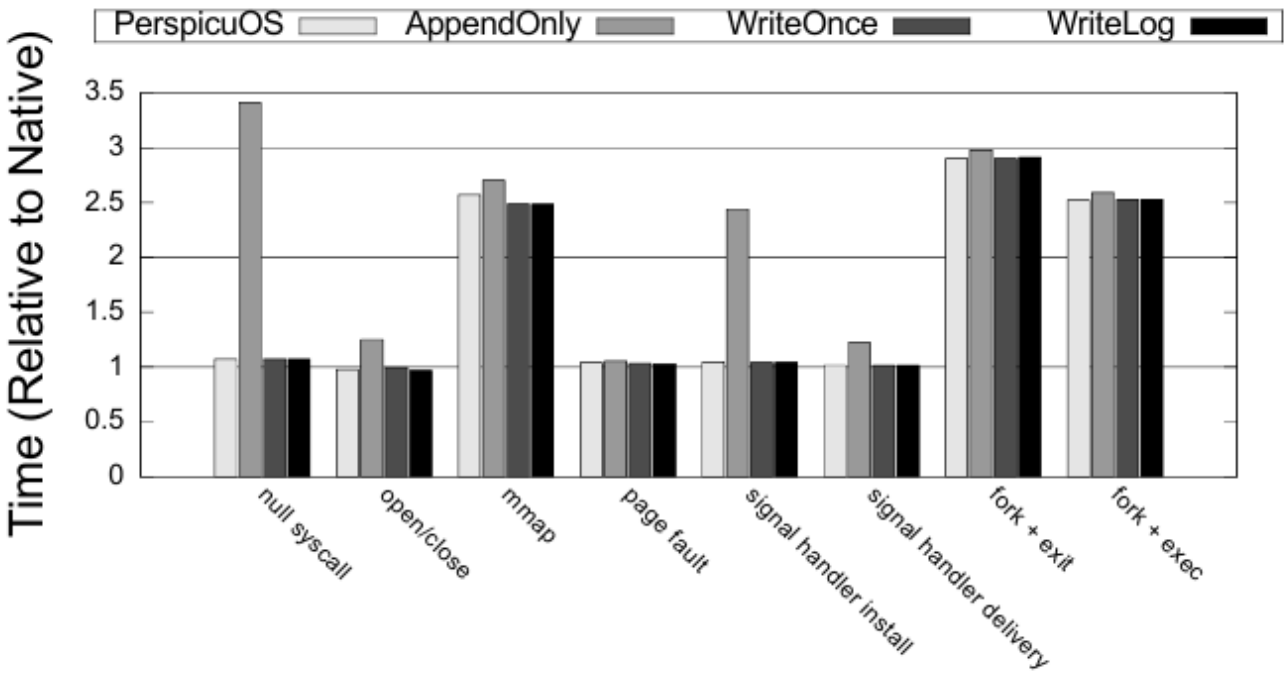


Figure 4. LMBench results.

PerspicuOS – Evaluation (3)

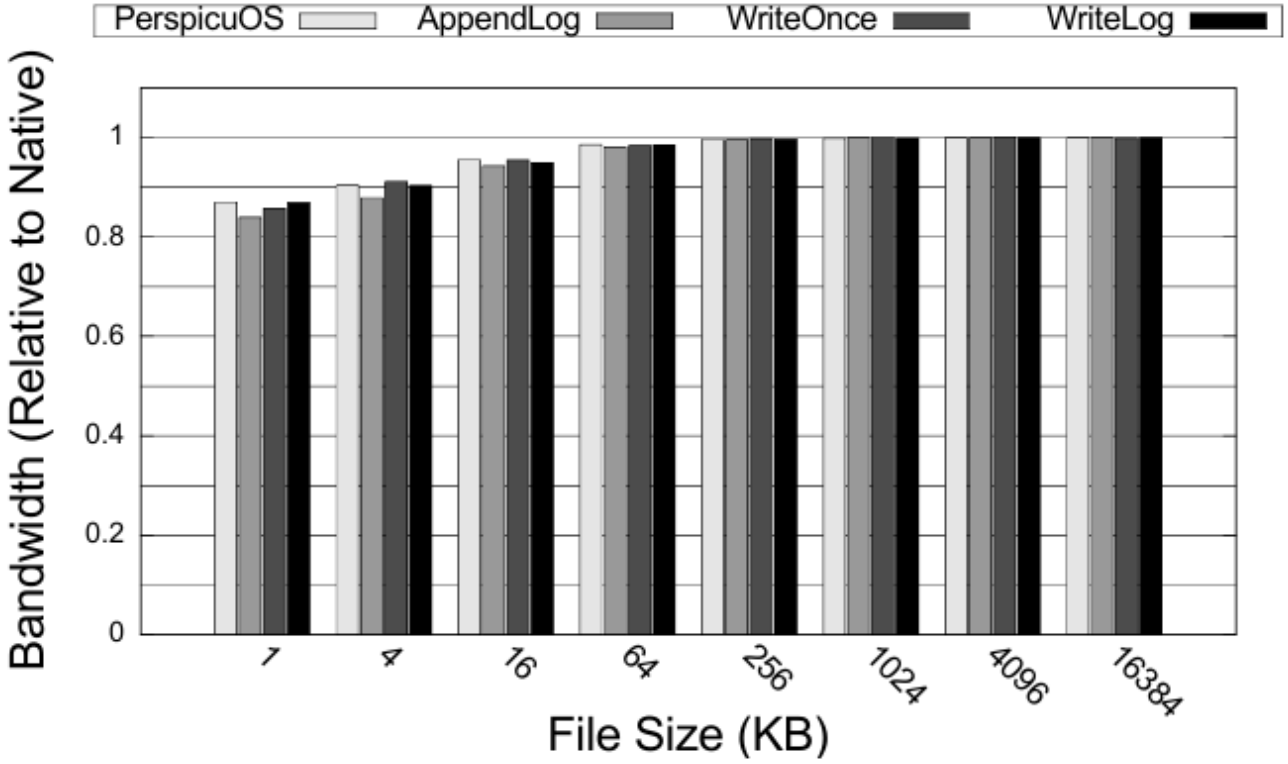


Figure 5. SSHD Average Bandwidth.

PerspicuOS – Evaluation (4)

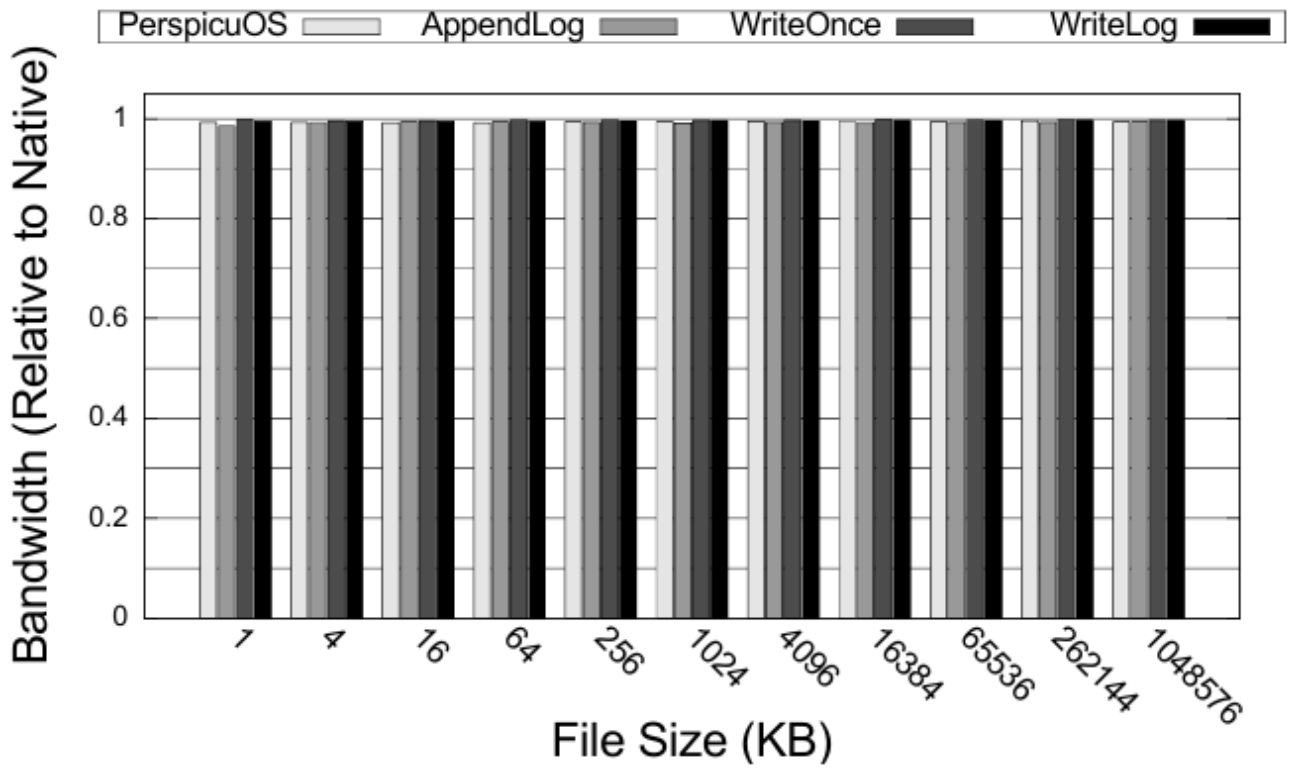


Figure 6. Apache average bandwidth.

PerspicuOS	AppendOnly	WriteOnce	WriteLog
2.6%	3.0%	2.6%	2.7%

Table 4. Kernel Build Overhead over Native.

Conclusion

- Nested kernel architecture
- Using the MMU to protect the MMU
- Write mediation policies for memory
- Protection domains without using VMM/Processes
  - Based on addresses

Discussion

Pro:
- Great if you want to log write accesses
- If the intention is rootkit detection this might be nice

Con:
- Not really suited for isolation of components
- Can't you still attack e.g. communication?
- The overhead evaluation … sucks