

# Recomposed Enforcement Paths: A Comparative Framework for Hardware-Security History

Federico Pacheco, Universidad Tecnológica Nacional, Argentina  
fpacheco@frba.utn.edu.ar | ORCID: 0000-0002-2803-1162

**Abstract**—Hardware security is often narrated as a succession of mechanisms or attacks. Comparing primary sources across memory protection, device assignment, boot measurement, execution metadata, physical roots, and capabilities, this article argues that protection was repeatedly recomposed around different objects, paths, and persistent states. Five provisional operations—naming, mediating, measuring, remembering, and erasing—organize the comparison; a guarantee-level test distinguishes local, composed, distributed, and unresolved paths. Two episodes examine the 80386 compatibility compromise and VFIO/IOMMU-group design through Linux v3.6 and v3.7. Local descriptor, capability, and CET return checks, measurement without a veto, and a composed VT-d invalidation path reject a monotonic-distribution account. The framework adds value where it exposes a consequential object, path, state, or lifecycle endpoint that a source-specific reading leaves implicit; it does not establish lineage, deployment prevalence, or causal priority among design constraints.

**Keywords**—hardware security; history of computing; computer architecture; memory protection; privilege; trusted computing; reference monitor; microarchitecture

## Introduction

Hardware security has no single origin. Histories beginning with physical protection, supervisory control, memory protection, cryptographic devices, or semiconductor security assign different purposes to hardware. Contemporary platforms combine secrecy, privilege, isolation, fabrication, and physical defense, yet these mechanisms are still commonly narrated in separate literatures.

The reference monitor is a useful point of departure. Anderson required a reference validation mechanism to be tamperproof, always invoked, and small enough to analyze [1]; Saltzer and Schroeder later articulated complete mediation [30]. Although often associated with an operating-system security kernel, modern enforcement spans processor privilege logic, MMUs, IOMMUs, interconnects, firmware, roots of trust, memory controllers, microarchitectural resources, security processors, and manufacturing states. Because no single block observes every relevant event in many platform claims, the reference monitor is better analyzed as a topology.

This article asks how hardware-enforced platform security can be compared as controls move among processor logic, translation structures, firmware, devices, roots of trust, memory controllers, and lifecycle state. It does not claim that these mechanisms share one lineage or that distribution necessarily increased over time. Platform security here means mechanisms below or alongside the operating-system abstraction that enforce or substantiate authority, isolation, integrity, or evidence. The scope excludes cryptographic accelerators as such, hardware Trojans, and supply-chain assurance unless they alter those platform boundaries.

Instead, we propose a comparative vocabulary of five provisional operations. Hardware names a subject, object, or state to which authority can attach. It mediates a path through which an operation must pass. It measures a state and produces evidence about it. It remembers history in tables, caches, counters, metadata, or physical enrollment data. It erases, invalidates, revokes, or transfers that state when authority ends. These operations are descriptive rather than claimed to be necessary, sufficient, recurrent in a representative population, or independently validated. Their purpose is to make dissimilar mechanisms comparable without collapsing their technical differences.

The comparison yields three contributions. First, it identifies a recurring historical problem—how to make a newly relevant object, path, or state governable—without treating similar solutions as a lineage. Second, it tests a provisional five-operation vocabulary and an inspectable guarantee-level topology rule against local, composed, proposed, and unresolved cases. Third, it reconstructs two bounded design episodes, the 80386 compatibility compromise and the VFIO/IOMMU-group proposals, to show how technical boundaries were shaped alongside performance, software compatibility, and interface concerns. These explanations can coexist; the selected records do not establish their relative causal weight.

## Historiographical Positioning

Historical scholarship has established that computer security emerged from time-sharing, military information policy, security-kernel research, and formal verification. Donald MacKenzie and Garrel Pottinger traced the security kernel, Bell-LaPadula, and the Orange Book [25]. Steven B. Lipner's history of the Orange Book and Paul A. Karger and Roger R. Schell's Multics retrospective further show that assurance regimes and adversarial evaluation cannot be inferred from architecture manuals alone [22], [17]. This article therefore does not claim the first history of computer security, the reference monitor, or trusted systems.

The cases range from memory protection, privilege, descriptors, and capabilities to DMA mediation, measured boot, confidential execution, execution integrity, microarchitectural leakage, entropy, unclonability, and tamper resistance. They are not descendants of one invention, but historically situated attempts to make particular entities, events, states, and transitions enforceable. Two recent systematizations compare hardware-assisted trust mechanisms and architectural building blocks for trusted execution environments; they are used here as comparative frameworks, not as evidence of historical adoption [43], [31].

Our contribution is methodological rather than a new universal theory of security or a causal history of adoption. Distributed reference monitor has an established use for systems coordinating local monitors and mandatory access-control policies across machines [26]. Distributed enforcement fabric denotes a narrower within-platform topology and is assigned only after the decision procedure below is satisfied. The five operations are analyst categories for comparison; recurrence alone is not evidence of influence or cause.

## Sources and Method

### Research design

This study is a structured historical comparison, not an experiment or a bibliometric census. Its unit of analysis is a concrete guarantee and its enforcement path within a mechanism family. Purposive selection contrasts architectures and periods rather than estimating frequencies of adoption.

### Case selection

Cases were selected from primary sources when they: (1) enforce or substantiate a platform property; (2) expose a mediated path or state; (3) permit historical comparison; and (4) pose a problem beyond the mere implementation of cryptography. Privilege, memory, DMA, boot, confidential execution, metadata, microarchitectural state, capabilities, and physical roots are dimensions of comparison, not exhaustive or mutually exclusive classes.

Selection targets theoretical contrast rather than statistical representativeness. Early local and composed checks, measurement without a veto, spatial capability checks, temporal revocation, device mediation, and microarchitectural observables were retained because they can produce different classifications under the same rules. The 80386 and VFIO episodes receive more process detail because the available design testimony and dated proposals expose competing constraints as well as mechanism semantics. Neither episode represents every architecture in its family. Requiring an exposed mediated path or state favors cases legible to the proposed coding framework; local and not-classifiable outcomes reduce but do not eliminate that selection dependence. The corpus is intentionally non-exhaustive; omission of intermediate capability machines, secure coprocessors, and other vendor architectures limits any claim about prevalence or a continuous trajectory.

### Source hierarchy

For claims about mechanism semantics, the preferred evidence is contemporaneous and primary: architecture manuals, original research papers, design notebooks, standards, patents, and certification profiles. Later surveys and historical studies are used to reconstruct context, adoption, terminology, and controversy. Current vendor documentation is admissible for mechanisms that remain evolving specifications, but a fixed edition or archived version is preferred where the claim depends on exact semantics.

This v5 working draft inherits the v4.4 claim register, case-operation and guarantee-decision matrices, boundary-sensitivity analysis, and reference audit without treating their results as a new audit. Separate v5 ledgers record the added VFIO documents, tagged code paths, and CET negative case with locators and excluded inferences. The inherited matrix reports the guarantee tuple and F1-F3 result for selected contrasts; G14 is a new bounded extension, and no end-to-end VFIO

assignment is classified. Earlier multi-AI reviews and their disagreement over G13 are disclosed as correlated consistency checks. No external human recoding or fresh multi-model review of v5 is claimed. The companion packet remains local and has no v5 archival DOI.

## Coding framework

TABLE 1. Coding fields used in the source-and-claim register accompanying this study.

| Field               | Question                                       | Example values                                                           |
|---------------------|------------------------------------------------|--------------------------------------------------------------------------|
| Protected state     | What must remain controlled or trustworthy?    | Instruction stream, mapping, key, measurement, cache history             |
| Enforcement path    | Where is the decision made?                    | Decode, address translation, memory controller, firmware, secure element |
| Trust boundary      | Which actors or components are inside/outside? | User/kernel, guest/hypervisor, enclave/host, device/factory              |
| Derived state       | What auxiliary history sustains the property?  | TLB, VMCS, PCR, tag, predictor, helper data                              |
| Adversary           | What access and capability are assumed?        | Process, guest, OS, device, remote verifier, physical attacker           |
| Residual limitation | What remains unmediated or ambiguous?          | DMA, rollback, side channel, debug, lifecycle transition                 |
| Operations          | What does the mechanism do?                    | Name, mediate, measure, remember, erase                                  |

## Comparison and inference

Each family is first reconstructed on its own terms: the problem described by contemporary designers, the mechanism they proposed, the surrounding system assumptions, and later evidence of limitations. A second pass applies the common coding framework. Similarities are analytically meaningful only when the mechanisms expose comparable relationships, not merely shared vocabulary. A page-table permission and a physical tamper sensor, for example, address different events, adversaries, and failures even though both can be called mediation. For the 80386 episode, a later design-team oral history is compared with the 1986 architecture manual; recollection is evidence of reported design reasoning, while the manual establishes available mechanisms. For VFIO, dated proposals and documentation establish stated problems, while v3.6 and v3.7 tags expose selected code paths. Neither sequence measures deployment or ranks competing motives.

Causal language is graded. Succession means only that one event followed another. Convergence means that independently documented mechanisms expose comparable relationships. Influence requires evidence of transmission, citation, personnel movement, or an acknowledged design source. Causation additionally requires a documented problem, a response mechanism, and evidence against plausible rivals. This article establishes succession and selected convergence; it does not claim cross-family influence or causal priority where those records are absent.

The corpus is selective and cannot estimate the prevalence of an operation or topology. Public documentation favors commercially significant systems and open standards, while proprietary security processors, military devices, manufacturing practice, and failed designs are underrepresented. Intermediate capability systems, secure coprocessors, and several non-Intel trusted-execution designs therefore remain comparison targets rather than evidence silently generalized by this article.

The study does not measure mechanism effectiveness through new experiments and does not infer deployment from architectural availability. Published attacks are evidence of a documented omitted path or assumption only for the cited platform and configuration. Institutional explanations in the rival-explanations section are research hypotheses unless accompanied by adoption, procurement, standards, or design-process records.

## A Focal Genealogy of Protection

The vocabulary of modern computer security encourages a compressed origin story. Early machines lacked protection; supervisor mode separated operating system from user; virtual memory isolated processes; rings refined privilege; protected-mode microprocessors brought these ideas to personal computers. Each statement contains some truth, but their sequence suggests a single problem and a cumulative solution. Contemporary documents instead show several protection problems developing in parallel: controlling sensitive instructions, allocating scarce storage, bounding references, sharing named objects, and invoking protected subsystems.

### **Atlas: protection within resource management**

The Atlas system at Manchester joined multiprogramming with an automatically managed memory hierarchy. Its Supervisor scheduled work, coordinated input/output, responded to interrupts, and maintained the directory underlying the one-level store [18, pp. 279–283], [10]. The 1961 paper specifies that hardware prevents an ordinary program using main control from accessing the subsidiary store and V-store [18, p. 279]. Programs could invoke extracodes, transferring control to predefined routines in fixed memory; the account on p. 283 distinguishes that controlled entry from direct access to the private stores [18, p. 283]. These mechanisms placed ordinary computation and supervisory services in different operating contexts, but their historical purpose cannot be reduced to defense against malicious code. They made an expensive shared machine usable while offering programmers a more manageable memory abstraction.

### **System/360: privilege and storage as separate dimensions**

IBM System/360 made a different distinction explicit. The Program Status Word recorded whether the processor was in problem or supervisor state; privileged operations were valid only in the latter. Storage protection used a separate key in the active PSW and keys associated with storage blocks. The manual specifies both checks in ‘System Structure,’ pp. 16–17; OS/360 documentation described partitions with different keys while resident control-program components operated with key zero [12], [11].

This arrangement matters because instruction privilege and memory authority were orthogonal. Entering supervisor state governed the ability to perform sensitive operations, whereas a key comparison governed particular storage references. Neither mechanism was a per-process paged virtual address space. Treating ‘kernel mode’ as the historical essence of protection would conceal the way System/360 represented jobs and storage partitions through a second hardware dimension.

### **Burroughs: typed descriptors rather than raw addresses**

The B5000 provides an early negative test rather than a necessarily incomplete precursor. Its 1963 manual documents that the processor checks an index against a descriptor size field before accessing a word [6, p. 4-1]; for data descriptors with nonzero size it also describes bounds checking [6, p. 5-2]. If the defined operation is that descriptor-mediated word access and failure prevents the access, the decision is local: the case refutes a universal reading of distributed enforcement. Descriptor creation, I/O paths, and special size semantics remain outside this narrow test. Two manual sections do not establish a system-wide guarantee or effective deployment. The later B6500 manual documents a different combination: an invalid descriptor index interrupts execution, while the master control program handles an absent data descriptor and moves data from disk [7, p. 3-4]. That is evidence for the 1969 architecture, not an additional check demonstrated for the 1963 B5000.

### **Multics: a hierarchy of protected subsystems**

Multics addressed sharing in a computer utility through segmented information and a hierarchy of protection rings. Michael D. Schroeder and Jerome H. Saltzer described rings of decreasing privilege, hardware validation of cross-ring references, and controlled calls through designated entry points [32]. The objective was not simply to create more supervisor modes. Intermediate rings could support protected subsystems whose users were less privileged, while the subsystem itself remained less privileged than inner services.

This design made domain crossing an architectural operation. A call to a protected subsystem was intended to resemble an ordinary procedure call while the processor validated the change in authority. Segments provided named units of sharing and protection; rings described the relation between executing procedures and those segments. The combination addressed a question that a binary mode does not: how can mutually suspicious or separately administered services coexist without moving every service into a monolithic supervisor?

## Protected-mode x86: recombination under compatibility

The 80286-to-80386 transition exposes a compatibility decision rather than merely a list of features. The manuals document real-address and protected execution, segmentation, and privilege checks [13], [14]. A 2008 panel of 386 designers recalled that a flat 32-bit model appealed to the workstation and Unix market, while binary compatibility mattered unevenly to customers and strongly to Intel field engineers selling existing software paths [8, transcript, pp. 4, 7–9]. Recompiling source for a different architecture was discussed as an alternative, not documented as the option adopted. These are retrospective accounts of the design debate, not a contemporary survey of customers.

The reported compromise retained segmentation and inserted a linear address between a program's logical address and physical memory. A large segment could present a nearly flat programming model, while software using older segments could retain that structure [8, transcript, pp. 8–9]. The 1986 manual independently documents segment translation into a linear address and optional paging into a physical address [14, ch. 5]. Virtual 8086 mode also gives a protected-mode monitor control over selected interfaces of an 8086 program; paging can remap its apparent address space, but the manual does not imply that every legacy instruction or device action receives an identical protection check [14, ch. 15, secs. 15.1–15.2].

The panel also recalled concern about the cost of two translation steps and a choice among proposals with different register and instruction-encoding implications [8, transcript, pp. 7–9]. The manual confirms the architectural path, not the relative weight of these considerations in the internal decision. Taken together, the sources show a documented form of recombination: legacy address semantics remained available, while a new intermediate address and monitor-controlled execution let other software use a different boundary. They do not establish how widely each path was used or make compatibility the cause of every 80386 security feature.

## What changed—and what did not

TABLE 2. Comparative interpretation of early protection architectures.

| System     | Security-relevant object made explicit  | Principal mechanism                            | Persistent limitation or dependency             |
|------------|-----------------------------------------|------------------------------------------------|-------------------------------------------------|
| Atlas      | Supervised job and managed storage      | Interruptions, extracodes, one-level store     | Correctness and authority of the Supervisor     |
| System/360 | Execution state and keyed storage block | Supervisor/problem state; storage keys         | Coarse regions and privileged key management    |
| Burroughs  | Bounded, typed program object           | Tagged descriptors and automatic bounds checks | Integrity and construction of descriptors       |
| Multics    | Segment and protected subsystem         | Segmentation, rings, gates                     | Software configuration and trusted inner rings  |
| 80286      | Segment, task, and privilege relation   | Descriptor tables and four privilege levels    | Complexity and compatibility with real mode     |
| 80386      | Page within a segmented linear space    | Segmentation, paging, IOPL and I/O bitmap      | Kernel control of mappings and transition state |

Comparative interpretation from contemporary manuals and original papers. The categories are not claims of functional equivalence.

The comparison distinguishes capability, design decision, and deployment. Atlas, System/360, and B5000 already divided responsibilities between supervisory programs and machine mechanisms, but some specific checks remained local. The 386 retained legacy paths while adding protected control [13], [14], [8]. This supports a history of recomposed enforcement paths rather than a necessary progression toward greater distribution.

Source note for the focal genealogy: see Kilburn, Payne, and Howarth [18]; Howarth, Jones, and Wyld [10]; IBM [12], [11]; and Burroughs Corporation [6], [7].

The early cases therefore support neither a processor-only origin nor a monotonic increase in distribution. They provide contrasting enforcement paths against which later device, boot, enclave, metadata, and lifecycle cases can be compared. The analysis reports recombination of responsibilities, not a universal transition.

No influence is inferred from similarity alone. Where a later mechanism resembles an earlier one, the article records convergence unless a primary source documents transmission or an explicit response. This rule prevents the comparative vocabulary from manufacturing a lineage after the fact.

## Expanding the Boundary

The focal genealogy made execution state, storage region, bounded object, protected subsystem, segment, and page visible to hardware. Later mechanism families did not replace those objects. They exposed additional actors, temporal states, metadata, observables, and physical processes that the processor-centered boundary had left implicit. The following cases are necessarily condensed; their purpose is to test whether the comparative model travels beyond modes and memory management.

### Devices become principals

Classical process isolation concentrated attention on memory references mediated by the processor. DMA devices exposed an additional path, while server consolidation encouraged direct assignment to untrusted virtual machines. In 2006, engineers from IBM, Intel, and AMD connected vendor IOMMU designs to Linux and Xen [4]. They distinguished older translation and reliability uses from isolation and reported Linux changes entering the mainline kernel. Under then-current Xen, a domain with direct hardware access had to be treated as trusted because the device could reach other guests; IOMMU isolation was intended to reduce that trusted set, not evidence that it had already done so. The paper provides technical context for VFIO, but this corpus does not establish a direct line of influence between its authors and the later patch series.

Tom Lyon's June 2010 VFIO patch proposed user-level PCI drivers for privileged and unprivileged processes [24]. Its revised form required an IOMMU domain before access to device registers, except for limited configuration-space reads, and proposed filtering writes to PCI configuration space. The proposed boundary therefore already involved software control over user access as well as DMA remapping. The document describes an individual-device interface; it does not prove that its authors regarded every device as independently isolatable or that the patch became a generally deployed system.

In an April 2012 RFC, Alex Williamson made the isolation granularity problem explicit [36]. Attaching individual devices to IOMMU domains could ignore restrictions imposed by IOMMU design, bus topology, or device behavior. The RFC proposed making an IOMMU group a more integral part of the driver model so VFIO could work with the set of devices the platform could isolate. This is a documented statement of design concern, not evidence that grouping alone solved every path.

The integrated series of 11 May 2012 proposed group tracking in `struct device`, support in Intel and AMD IOMMU drivers, PCI DMA-quirk and Access Control Services checks, a rewritten VFIO core, a modular IOMMU backend, and matching QEMU work [37]. On 29 May, Williamson separated IOMMU group support from the VFIO portion to advance the shared foundation through subsystem review [38]. The sequence shows that the group boundary required coordination across driver core, PCI, IOMMU, VFIO, and user software. It does not show which patches were accepted unchanged or how often the result was used.

The July pull request proposed VFIO for the Linux 3.6 merge window and credited Lyon's original work alongside the later overhaul [39]. Its claim that the new interface would be safer through groups and filtered resource access is contemporary design testimony, not an effectiveness measurement. A contemporary documentation patch explains why a multifunction device or non-ACS bridge can allow transactions to bypass the expected IOMMU boundary; it defines the group as VFIO's ownership unit while permitting several groups to share a mapping context in a container [40]. Thus device handle, assignable ownership group, and mapping context are distinct objects. The documented shift is in the public interface and its stated isolation assumptions, not the invention of IOMMU protection in 2012.

The tagged Linux v3.6 tree gives a narrower implementation check [44]. Its `Documentation/vfio.txt` calls the group the ownership unit and permits multiple groups in one container. The core's `group-unset` and `final-file-release` paths detach a group and dissolve an empty container; the `type1` backend calls `iommu_detach_group` and unmaps recorded DMA entries on release. The PCI device's final close calls a disable path that attempts `pci_reset_function`, but this path does not check the reset return value. These are identifiable software transitions, not proof that reset succeeded or that stale translations, peer traffic, and in-flight requests were exhausted on a real host. In v3.7 the `type1` backend and documentation are byte-identical to v3.6; the inspected core and PCI changes do not alter these release paths [45]. This

comparison establishes continuity of the selected paths across two tags, not invariance of VFIO as a whole.

The v3.6 documentation supplies a concrete but unexecuted topology: IOMMU group 26 contains bridge 0000:00:1e.0 and functions 0000:06:0d.0 and .1; behind the bridge, the IOMMU cannot distinguish transactions from the functions [44]. For an assumed Intel VT-d/type1 path, the tagged code checks group viability, attaches the group, records DMA mappings, and offers INTx or MSI teardown branches [44], [46]. This is a conditional source reconstruction, not a host trace. Its explicit VFIO\_IOMMU\_UNMAP\_DMA chain reaches `intel_iommu_unmap`, which clears page-table entries; the Intel helper says a TLB flush should follow, yet no invalidation call is visible before that inspected chain returns [46]. Group detach is different: the Intel path clears device context and requests context and IOTLB invalidation, with a queued wait path when enabled [46]. Final PCI close attempts a function reset without checking its return; the PCI helper can report that single-function reset is unsupported [44], [46]. The documented sample itself uses API names inconsistent with the same tag's header, so it is evidence for topology and intended sequence, not an executable experiment. Without the example host's capability, IRQ mode, reset result, peer traffic, and in-flight completion, G05 remains not classifiable. The group-26 trace in the companion packet records each transition and excluded inference.

VT-d revision 4.1 documents software-assigned DMA domains, which may be shared by devices, and separate interrupt remapping [16, secs. 2.5.2 and 3.2–3.4]. It specifies a configurable architectural capability, not universal deployment. The VFIO documents and tagged code show how software exposed an isolation unit compatible with such hardware and with platform topology. They do not supply a complete device-assignment, reset, interrupt, peer-traffic, invalidation, and release trace for a deployed host.

The device case makes a distinction the cross-family vocabulary might otherwise hide. A device or requester identifier can be an architectural principal for one translated DMA request, while the group is the proposed software ownership unit for safe user access. Naming both is necessary; treating them as equivalent would erase the topology problem documented in 2012. The guarantee must nevertheless be frozen before its topology is classified. A single DMA request that reaches one configured IOMMU is local (G04). Generic device reassignment remains not classifiable because the sources do not provide one complete device, interrupt, transaction, reset, and in-flight-request trace (G05). VT-d specifies a narrower completion mechanism: after software changes DMA-remapping structures and submits the applicable invalidations, a following Invalidation Wait Descriptor completes after the preceding descriptors and requested drains within their defined scope [16, secs. 6.5.2, 6.5.2.8, 6.5.2.11, 6.5.3.3, and 6.5.4]. This statement begins after a correct submission sequence; before invalidation, hardware may use an old or new translation. Its scope also depends on ATS and Device-TLB invalidation, PASID mode, every remapping unit sharing the structures, request type and range, peer traffic exclusions, and successful queue processing. For the frozen successful single-unit path, the specification supports F2 and F3, but it does not establish F1. G13 is therefore composed but not demonstrated to be distributed. Neither the richer VFIO record nor a multi-stage path turns G05 or G13 into an established distributed case.

### **Boot becomes a sequence that can be measured**

Sailer and colleagues used a TPM root in 2004 to protect measurements of dynamically loaded executable content in Linux as well as boot content [29]. Kernel agents collected the history and an external validator assessed its acceptability. Measurement records a state before execution; it is not an automatic local veto. In TPM 2.0, when invoked with a valid signing key, TPM2\_Quote signs an attestation containing a digest of selected PCRs and permits caller-supplied data; the PCR values must be obtained separately. With TPM\_RH\_NULL as signHandle, the command can instead return a null signature [34, part 3, sec. 18.4.2; part 1, p. 83]. The guarantee depends on the later operation a verifier conditions, measurement freshness, and applied policy [29], [34].

This is a negative test when the defined operation is local code loading: measuring and recording do not by themselves block undesired execution [29]. It is a conditional test when a relying party later denies a specific operation on the basis of the evidence. The two events must not be conflated, nor should a policy decided by another institution be attributed to the TPM root.

## Privileged software moves outside selected confidentiality boundaries

Trusted execution environments respond to an ownership inversion intensified by mobile and cloud computing: the operating system or hypervisor configures resources, but the processor may prevent it from directly accessing selected protected code and data. Intel's 2013 paper describes EPC pages and metadata that restrict those accesses even from privileged software; the OS nevertheless retains page mappings, resource allocation, and enclave scheduling [27, secs. 2–4]. This documents an architectural isolation model, not availability or general immunity from leakage. Costan and Devadas reconstruct SGX attestation and note that a malicious OS can still observe cache timings; their independent report identifies some implementation details as conjectural [9, secs. 5.8 and 6.6.6–6.6.7]. Local EPC isolation and third-party acceptance of an attestation are distinct operations.

## Execution metadata becomes a protected object

Memory- and control-flow-security mechanisms expose still finer-grained state. Shadow stacks protect return addresses through an independent control-flow record. Pointer authentication binds a cryptographic code to a selected pointer value. Memory tagging compares tags on pointers and memory regions. These mechanisms protect different properties: when enabled, CET compares returns with its shadow stack and tracks indirect branch targets with ENDBRANCH [15, secs. 2.3 and 3.3–3.4]; Armv8.3-A can create and validate a PAC in high address bits, and indirect use of an invalidated pointer can cause an exception [5, 'Pointer authentication']; MTE can detect certain spatial and temporal violations with four-bit tags, but does not guarantee detection of every violation in an execution [3, 'Memory safety with MTE'].

## Internal history becomes an observable

Caches, translation buffers, branch predictors, and execution buffers retain history to avoid repeating work. Timing attacks had already shown that a cryptographic operation's duration can depend on secrets [19]; FLUSH+RELOAD made the cache state of shared pages observable [42]. Spectre influenced branch prediction so that a victim transiently executed instructions that accessed data within its own authorized address space and encoded those data in cache state [20, secs. I.B and III]. Meltdown, on affected processors, exploited a user-mode load from a mapped but privileged kernel address; although the access raised or suppressed an exception, a transient instruction could leave a secret-dependent cache effect [23, secs. 3–5.1]. Both attacks separate rolled-back architectural state from residual observables, but differ in the operation and preconditions. Lampson's confinement note had conceptually described channels outside the authorized interface; it did not anticipate these particular mechanisms [21].

In an affected Meltdown case, access to a privileged page can be rejected architecturally while a measurable microarchitectural signal survives; in Spectre, the victim is authorized to access the data but the attacker exploits transient execution. Functional rollback is not informational rollback in either case [20], [23]. Historical interpretation must specify the processor, mappings, instruction sequence, and channel: the papers do not establish that every CPU or deployment shares the vulnerability. Mitigations such as partitioning, flushing, barriers, predictor restrictions, or constant-time programming trade cost against observability; their effectiveness depends on the threat and configuration.

## Manufacturing and physical variation enter the lifecycle

Physical mechanisms are boundary cases rather than a homogeneous genealogy. Entropy sources require characterization, entropy estimation, and health testing; conditioning is used when applicable rather than required universally [28]. Physical unclonable functions use manufacturing variation for responses or derived keys, adding enrollment and error reconciliation [33]. Secure elements restrict operations around stored secrets, while anti-tamper mechanisms detect, delay, evidence, or respond to physical access. Anderson and Kuhn documented attacks on devices regarded as tamper resistant: a physical barrier is not an absolute guarantee [2]. Their shared analytical value is limited but useful: each incorporates fabrication, provisioning, environment, or ownership into security state.

## Capability boundary case and CHERI

CHERI integrates tagged, bounded, permission-bearing capability references into a conventional ISA [35]. The bounds and permission check on each use can be local: for that spatial guarantee, CHERI is a negative case for a universal version of the thesis. Compilers and operating systems influence capability distribution, but that dependency does not automatically make every check a distributed decision.

The temporal guarantee supplies a different test. The CHERI report describes delaying address reuse until outstanding capabilities in registers and memory can be revoked, supported by tag metadata and tables [35, sec. 5.1.2]. CHERIvoke

proposes quarantine of freed objects, a shadow map, and periodic sweeps that invalidate dangling pointers [41, secs. 3.1–3.3]. Under that allocator model, revocation coordinates distinct actors and states; it is a conditional positive case, not evidence that every ChERI implementation has this property or deploys ChERlvoke.

**TABLE 3. Comparative results for named guarantees. G01–G13 outcomes are inherited from the v4.4 decision matrix; G14 is a new, source-bounded negative case. The VFIO history does not turn G05 into a classified deployed guarantee.**

| Named guarantee and event                     | Deciding path or missing link                                     | Result                        | Evidentiary boundary                                                                          |
|-----------------------------------------------|-------------------------------------------------------------------|-------------------------------|-----------------------------------------------------------------------------------------------|
| B5000 bounded word access (G01)               | Processor checks a descriptor on use                              | Local                         | Descriptor creation and I/O excluded                                                          |
| 80386 sensitive instruction in V86 mode (G03) | Processor privilege check                                         | Local                         | Architectural path, not prevalence                                                            |
| Measured load as a local veto (G06)           | Measurement exists; no automatic load refusal                     | Guarantee not established     | A later verifier can define a different goal                                                  |
| Safe device reassignment (G05)                | Reset, peer, interrupt, DMA, and release path not traced together | Not classifiable              | Tagged code exposes some transitions, not one deployed lifecycle                              |
| ChERI bounded capability use (G09)            | Processor checks bounds and permissions                           | Local                         | Temporal revocation is a separate goal                                                        |
| CET mismatched near return (G14)              | Enabled shadow stack compares the return addresses                | Local                         | Requires enabled, protected shadow-stack state; not whole-program control-flow integrity [15] |
| ChERlvoke post-sweep reuse (G10)              | Allocator, shadow map, sweep, and capability use coordinate       | Distributed in proposed model | Not every ChERI implementation deploys it                                                     |
| VT-d covered remapping update (G13)           | Software submission and one unit's invalidation completion        | Composed, not distributed     | F1 is false for the minimal specified path                                                    |

The table is a contrast, not a distribution of historical outcomes. A tighter object or more components need not yield a distributed guarantee: B5000, ChERI, and CET checks remain local for their named accesses, measurement alone does not veto a load, and a composed VT-d path fails F1. Intel's CET specification states that, when shadow-stack operation is enabled, a near RET compares the ordinary and shadow return addresses and raises a control-protection exception on mismatch [15, sec. 2.3]. G14 freezes that one instruction event and presumes protected shadow-stack state; it does not claim that all control-flow transfers or software configuration are secure. The 80386 and VFIO episodes concern design choices in different periods, while G03 and G05 isolate narrower guarantees within those episodes. Their comparable historical feature is the need to specify which object and path are controlled, not an identical topology or a line of influence.

These cases support the moving-boundary interpretation but also constrain it. The same movement may be motivated by different institutions: cloud providers require device assignment and confidential virtual machines; platform vendors require controlled boot and update; safety and payment industries require evaluated physical boundaries; performance competition creates shared microarchitectural state. A technical omission identifies what a mechanism must cover. It does not, by itself, identify who will pay for the coverage or which design will prevail.

## Five Provisional Operations

The selected guarantees in the decision matrix use the same record: (1) the named guarantee and protected state; (2) adversary capabilities and explicit exclusions; (3) trusted actors and components; (4) the deciding event and enforcement path; (5) derived state; (6) lifecycle completion; (7) evidence status; and (8) the result of the topology test. A field may be marked absent or not applicable in the historical actors' own framing rather than retrofitted. In the VFIO episode, this discipline separates a device handle, an isolatable ownership group, and a shareable mapping context. It also makes the end of an assignment an open evidentiary question. If a mechanism-specific reading already exposes both distinctions, the five questions add no demonstrated value for that case.

**Naming: making authority addressable**

A policy cannot be enforced until hardware distinguishes the entities and states to which it applies. System/360 names a protection context with a key; descriptors name bounded objects; rings and privilege levels name domains; page tables and IOMMUs bind addresses or requester identifiers; tags and capabilities name finer-grained authority. Analysis must identify the subject, object, state, granularity, and event that creates or changes the binding. Aliasing, substitution, or inappropriate granularity can make a precise check apply to the wrong thing.

Criterion N. For a protected claim, identify the subject, object, state, granularity, and event that creates or changes their binding. If these cannot be stated, the claimed boundary is underspecified.

**Mediating: covering every relevant path**

Mediation is the operation most closely associated with the reference monitor. A privileged-instruction check interposes at decode or execution; a descriptor or page-table walk interposes on an address; an IOMMU interposes on DMA.

Completeness is relative to a named operation and threat: CPU permissions do not mediate DMA, verified boot does not mediate later corruption, and enclave protection does not cover every side channel. Every agent capable of reaching or influencing protected state must map to a check, trusted bypass, or explicit exclusion.

Criterion M. Enumerate all agents capable of reaching or influencing the protected state, then map each path to a check, a trusted bypass, or an explicit exclusion. Unmapped paths are boundary hypotheses, not harmless implementation details.

**Measuring: producing evidence rather than authority**

Some mechanisms measure a condition and provide evidence to another actor rather than deciding locally. Boot measurements extend digests into PCRs; attestation authenticates a report; a remote verifier applies policy. The evidence must name the intended object, be fresh enough for the decision, resist unauthorized change, and reach an actor able to condition a defined operation. Measurement without such a decision path is observation, not enforcement.

Criterion E. State what is measured, when, by which root, with what freshness, and who converts the evidence into a decision. A measurement without an acceptance rule is description, not enforcement.

**Remembering: protecting the state that makes enforcement possible**

Hardware security depends on derived history. A TLB caches translations; an IOTLB caches device mappings; descriptor tables encode object metadata; predictors and caches retain execution history; counters and PCRs accumulate evidence. This state improves performance or continuity but can become stale, inconsistent, replayed, or observable. When a guarantee depends on it, the derived state belongs inside the protected state and must be included in the threat model.

Criterion R. List the security-relevant caches, tables, counters, tags, buffers, measurements, and enrollment artifacts on which the property depends. For each, specify integrity, confidentiality, synchronization, and lifetime requirements.

**Erasing: ending authority and paying the cost of history**

Authority eventually ends: a page is remapped, a device reassigned, a capability revoked, firmware updated, or an enclave torn down. Completion may require invalidating caches and translation structures, draining in-flight uses, erasing keys, updating epochs, or reconciling replicas. This is where performance and security most visibly conflict. The analysis must define the termination event, the invalidation completion condition, residual copies, and the actor trusted to declare completion.

Criterion D. Define the event that ends authority, the completion condition for invalidation, the residual copies or in-flight uses, and the party trusted to confirm completion.

**What the vocabulary adds—and where it does not**

The five operations are useful only if they change the questions asked of a source-defined guarantee. They do not create historical facts. In VFIO, the group and container are already named by the designers [36]–[40]; the v3.6 release paths identify detach, unmap, and an attempted PCI reset [44]. The vocabulary's incremental contribution is to ask whether ownership can be ended after peer traffic, stale translations, and in-flight requests have been accounted for. The selected record does not answer that broader question. For B5000 and the enabled CET near-return check, the primary documents already state the narrow deciding events [6], [15]. Relabeling them with five operations adds little. These negative results are part of the framework's test.

TABLE 4. Incremental diagnostic value of the five operations in selected source readings.

| Case                   | Source-specific reading                                                                                                                                         | Added question or distinction                                                                                            | Limit of the framework                                                                                                                                    |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| B5000 word access      | The manual directly states the descriptor size check [6].                                                                                                       | No material new question for that narrow access.                                                                         | The vocabulary mainly restates the source; keep the local negative case.                                                                                  |
| CET near return        | The specification directly states the comparison and #CP on mismatch [15, sec. 2.3].                                                                            | No new question for that enabled instruction event; enabling and shadow-stack integrity belong to a broader guarantee.   | Do not recast one local check as whole-program control-flow integrity.                                                                                    |
| 80386 address model    | Manual and design recollection separate architecture from reported motive [8], [14].                                                                            | Which address and mode state does each path preserve?                                                                    | The five operations cannot explain why one design won.                                                                                                    |
| TPM measured loading   | A measurement can inform a later verifier [29], [34].                                                                                                           | Separate evidence production from a specified vetoing decision.                                                          | The source already makes much of this distinction; no universal attestation guarantee follows.                                                            |
| VFIO device assignment | Proposals and v3.6 code name device, isolatable group, and mapping container; the documented group 26 permits a conditional source trace [36]–[40], [44], [46]. | Distinguish explicit DMA unmap from final group detach and ask which completion event ends the former owner's authority. | The inspected unmap chain lacks a visible invalidation before return; detach requests one, but neither path proves safe reassignment on the example host. |

## From a component to a topology

The five operations need not reside in one place, but a multi-stage path is not automatically distributed. Figure 1 reconstructs the bounded evidence for G13: software updates remapping structures and submits the applicable invalidations; one remapping unit orders its descriptors, performs requested drains within their defined scope, and completes a following wait descriptor [16, secs. 6.5.2.8, 6.5.2.11, 6.5.3.3, and 6.5.4]. The postcondition is limited to successful completion of that submitted sequence. ATS and Device-TLB invalidation, PASID state, other remapping units sharing the tables, peer traffic, and queue failures may require additional paths or fall outside the drain guarantee. This reconstruction supports composition and lifecycle reasoning, but it does not by itself satisfy F1 or establish a distributed guarantee.

We call a guarantee a distributed enforcement fabric only when all three conditions are documented: (F1) no single decision point observes every event required by the stated guarantee; (F2) an identifier, policy, measurement, or revocation state must retain its meaning across at least one component boundary; and (F3) correctness requires coordinated update, evidence interpretation, or invalidation across that boundary. A component is one independently specified decision or state-transition boundary, not an arbitrary grouping chosen to obtain a result. A guarantee is local when one such point decides the operation without cross-boundary coordination. It is composed but not distributed when the documented path spans components yet at least one of F1–F3 is false. It is not classifiable when evidence or boundary definition is insufficient. This topology classification differs from TCB membership and from distributed monitors across machines [26].

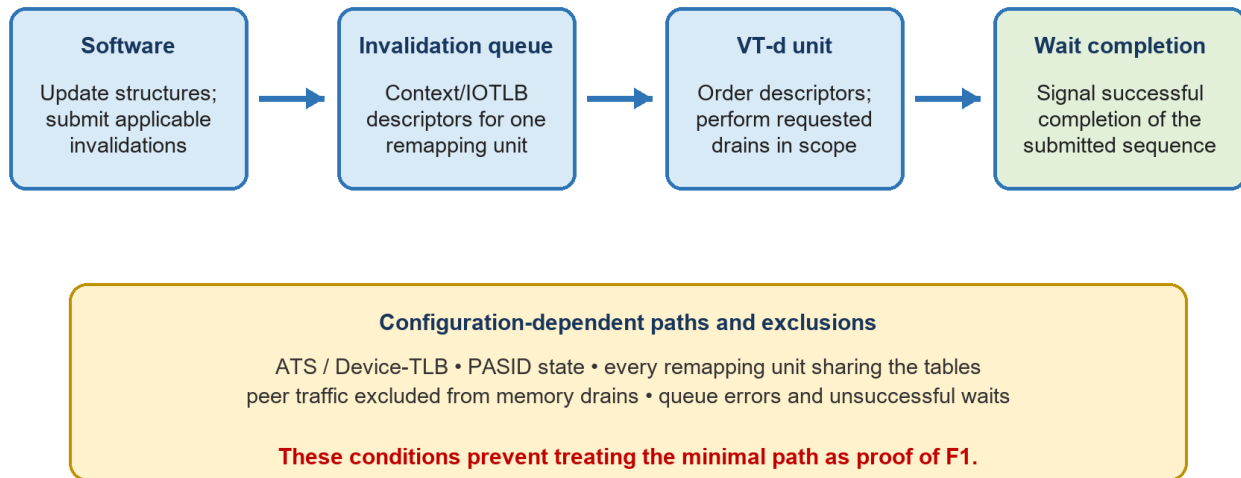
Operational test. First freeze a guarantee tuple  $G = (\text{protected state, security goal, adversary, trusted components, deciding event, lifecycle endpoint})$  and the component granularity justified by the source architecture. Then enumerate agents and paths and apply F1–F3. Each predicate is coded yes, no, unknown, or not applicable; conditional is not a permitted result. All three true yields distributed. A documented single deciding point without cross-boundary coordination yields local, even if data cross a component boundary. A multi-decision path with at least one false predicate yields composed but not distributed; missing path evidence or unresolved granularity yields not classifiable. If the named goal is demonstrably absent, the result is guarantee not established rather than not classifiable. Repeating the test after changing one element of  $G$  or one component boundary is a sensitivity analysis, not a rescue of the original classification. The five operations describe the path but do not determine the result. They are provisional analyst bins, not a validated measurement instrument. The supplement matrices expose each assignment and sensitivity case so readers can test alternative decompositions.

The sensitivity analysis changes one G13 input at a time. The minimal single-unit path remains composed but not distributed because F1 is false. Adding ATS/Device-TLB state, PASID state, another remapping unit, peer traffic, or a queue-failure path without a complete trace makes that broader guarantee not classifiable rather than distributed. These

perturbations demonstrate boundary dependence; they do not validate the categories independently.

Three bounded analytical expectations guide comparison: derived state should be included in protected state when G depends on it; completeness should be evaluated relative to the enumerated agents and paths; and a lifecycle claim should state a completion condition for ending authority. These are diagnostic expectations, not historical findings or completeness theorems. If a case is fully explained without them, the vocabulary adds no demonstrated value.

## VT-d remapping update: specified path and scope limits (G13)



Analytical reconstruction of Intel VT-d rev. 4.1 — composed path, not deployment evidence

Figure 1. Spec-defined VT-d completion path and configuration-dependent scope limits. The path is composed but does not by itself establish F1, deployment, or a distributed guarantee.

## Rival Explanations and Limits

A framework built after the events can confuse classification with explanation. The five operations classify work performed by mechanisms; recurrence does not explain adoption. Performance, compatibility, language structure, institutional demand, and boundary omission generate distinct evidentiary predictions. This article uses them to bound interpretation and does not assign causal priority without process evidence.

### Performance and resource utilization

Several mechanisms that later became security boundaries originated partly in the management of scarce resources. Atlas virtual memory coordinated core and drum; caches and predictors retained history to improve throughput; virtualization consolidated machines; IOMMUs supported device assignment as well as isolation. The VFIO documents also describe the cost of making each consumer assemble groups and the possibility of sharing mapping contexts across groups [37], [40]. A purely security-driven account would therefore mistake dual-use infrastructure for a sequence of defensive inventions. The moving-boundary model remains useful only if it distinguishes the property later enforced from the economic or performance conditions that made the mechanism viable. The selected documents support both isolation and interface-economy concerns; they do not rank their relative causal force.

### Compatibility and installed software

Protected-mode x86 illustrates how legacy semantics shape security. The availability of four privilege levels did not compel operating systems to use four domains, and real mode survived because software and boot processes depended on compatibility. Similar constraints recur in firmware interfaces, page-table formats, device protocols, and cloud migration. A mechanism may offer a finer boundary while the ecosystem selects a simpler subset. Hardware capability is thus evidence of design possibility, not evidence of historical practice.

## Languages and system structure

Burroughs descriptors and Multics rings show that the object visible to hardware depends on assumptions about programs and services. A language-directed architecture can enforce bounds and types that a page-oriented architecture leaves to compilers or runtime systems. Conversely, a general-purpose processor designed for multiple operating systems may expose lower-level primitives. The five operations should not erase these commitments. Naming a segment, an array, a page, and a pointer tag creates different policies even when all four can be described as mediation.

## Markets, institutions, and standards

Organizations may define acceptable states and pay the costs of enforcement, but this manuscript does not treat that general proposition as evidence for a particular adoption. Certification, procurement, standards, ownership, and liability remain candidate explanations that require dated institutional records before they can be assigned causal weight in any case.

**TABLE 5.** *Rival explanations applied to the two design episodes. Each row reports a documented concern, not its measured causal weight.*

| Episode and possible explanation  | Evidence in the selected record                                                                                                                              | What the record supports                                             | What remains unresolved                                                        |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|--------------------------------------------------------------------------------|
| 80386: binary compatibility       | Designers recalled field-engineer pressure to run existing software and considered source recompilation [8, pp. 7–9].                                        | Compatibility was a reported design constraint.                      | Contemporary decision records and software-use measures are absent.            |
| 80386: flat model and performance | Panel members described Unix/workstation goals and concern over two address translations; the manual specifies the resulting path [8, pp. 7–9], [14, ch. 5]. | Address-model and performance concerns coexisted with compatibility. | The testimony does not rank their influence or measure the adopted path's use. |
| VFIO: isolation granularity       | The April RFC and documentation identify IOMMU, bridge, and device-topology restrictions [36], [40].                                                         | Grouping was an explicit proposed response to an isolation problem.  | Release behavior and security across real host topologies are unmeasured here. |
| VFIO: interface economy           | The May series moved group tracking into shared infrastructure; documentation allows groups to share a container [37], [40].                                 | Common interface and mapping-context concerns are also documented.   | No controlled comparison assigns them a causal share relative to security.     |

The episodes therefore support coexistence of constraints, not a single-cause story. The 80386 manual can confirm the resulting address path but cannot establish which recollected consideration decided the design. The VFIO proposal sequence shows stated isolation and interface problems but does not measure adoption or comparative safety. Institutional demand and language structure remain plausible broader explanations; in these two episodes, the selected corpus is too narrow to adjudicate them.

The study has four principal limitations. Cases were selected for theoretical contrast, not prevalence. Public sources reveal specified capability and proposed interfaces more reliably than deployment or intent; no deployed historical case in this corpus independently validates a distributed classification, and neither the VT-d specification nor the VFIO record establishes operational prevalence or correct configuration. The group-26 reconstruction is a documented example joined to conditional v3.6 code paths, not a trace of an assigned and released host device. It cannot establish PCI reset success, IRQ and peer-path exhaustion, actual Intel IOMMU selection, or completion of in-flight DMA. Analyst categories can create retrospective similarity, so the register separates contemporary actor language from later coding and permits not-applicable values. The earlier frozen multi-AI recoding tested v4.4 consistency only; it is correlated, non-human evidence and neither establishes inter-coder reproducibility nor validates this v5 revision. No external human coders were used for v5.

Published attacks provide uneven evidence. A successful attack can reveal an omitted path or assumption, but the absence of a published attack does not establish completeness. Public documents also reveal architectural capability more reliably than actual deployment or institutional intent. The conclusions are therefore bounded to the cited revisions, platforms, and operations.

## Implications for Historical and Technical Analysis

For historical analysis, the first task is to identify the object and operation as the sources defined them. The 80386's linear address and legacy execution path, and VFIO's isolatable ownership group, were responses to different documented constraints. Their resemblance as boundary changes is an analytical comparison, not evidence of transmission or a shared cause. The comparison becomes useful when it specifies what a source did not settle: deployed use for the 80386 paths and a completed device-release guarantee for VFIO.

For architects and evaluators, the same reading yields compact questions. What has been named, at what granularity, and by whom? Which paths reach the protected state? Which measurements produce evidence rather than prevention, and who interprets them? What derived history sustains the guarantee? What must be invalidated or forgotten when authority ends? These questions do not replace threat modeling or formal verification. Table 4 shows their uneven value: they expose a missing lifecycle endpoint in the VFIO corpus but add little to the already explicit B5000 word-access check.

## Conclusion

The selected history does not trace a single march from local checks to distributed enforcement. B5000 bounds, CHERI spatial checks, and enabled CET near-return comparisons are local for their named accesses; measurement without a veto fails a different goal; CHERIvoke is distributed only in a proposed model; and the bounded VT-d completion path is composed without satisfying F1. The classification changes when the protected state, adversary, path, or lifecycle endpoint changes.

The two design episodes give that result historical texture. The 80386 retained legacy addressing while introducing an intermediate linear address and protected control over selected old-software paths. The VFIO proposal sequence made the isolatable group a unit of ownership distinct from a device handle or mapping container; v3.6 and v3.7 code expose selected release transitions without proving a safe deployment. In both cases the source record shows multiple constraints; neither supports a claim of causal priority, universal deployment, or influence between episodes.

The five operations should be retained only where they expose an otherwise missed actor, path, state dependency, evidence decision, or termination condition. Here their clearest incremental question concerns the end of a VFIO assignment; for B5000 and CET's narrow return check they mostly restate the primary source. A future end-to-end device trace or a fuller contemporary 80386 design archive could revise these readings. Until then, recomposition names a disciplined comparison of bounded guarantees, not a law of hardware-security history.

## ACKNOWLEDGMENTS AND AI DISCLOSURE

OpenAI Codex assisted at a generative level with the outline; comparative synthesis; research design; five-operation framework; sensitivity analysis; rival explanations; limitations; conclusion; Figure 1; bibliography normalization; and source-claim register. It drafted the v5 revision, the release-tagged VFIO comparison, the group-26 source reconstruction, and the CET negative-case extension. Multiple AI models performed a disclosed non-human recoding of the earlier v4.4 packet; no new independent model audit of v5 and no external human recoding are claimed. Before submission, the author must check every retained factual claim and citation against the cited sources, select or revise the interpretations, and accept responsibility for accuracy, originality, and conclusions. No external peer review is claimed. The author declares no conflict of interest or external funding.

## BIBLIOGRAPHY

- [1] J. P. Anderson, Computer Security Technology Planning Study, ESD-TR-73-51, vol. I, Electronic Systems Division, U.S. Air Force, Oct. 1972.
- [2] R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," in 2nd USENIX Workshop on Electronic Commerce, pp. 1–11, 1996.
- [3] Arm Limited, Armv8.5-A Memory Tagging Extension, white paper, 2019.
- [4] M. Ben-Yehuda, J. Mason, O. Krieger, J. Xenidis, L. van Doorn, A. Mallick, J. Nakajima, and E. Wahlig, "Utilizing IOMMUs for Virtualization in Linux and Xen," in Ottawa Linux Symposium, 2006, pp. 71–86.
- [5] D. Brash, "Armv8-A Architecture: 2016 Additions," Arm Community, Oct. 26, 2016.
- [6] Burroughs Corporation, Operational Characteristics of the Processors for the Burroughs B5000, 5000-21005, 1963.
- [7] Burroughs Corporation, B6500 Information Processing Systems Reference Manual, 1043676, Sep. 1969.

- [8] Computer History Museum, "Intel 386 Microprocessor Design and Development Oral History Panel," transcript, 2008, pp. 4, 6–9. [Online]. Available: [https://archive.computerhistory.org/resources/text/Oral\\_History/Intel\\_386\\_Design\\_and\\_Dev/102702019.05.01.acc.pdf](https://archive.computerhistory.org/resources/text/Oral_History/Intel_386_Design_and_Dev/102702019.05.01.acc.pdf)
- [9] V. Costan and S. Devadas, "Intel SGX Explained," IACR Cryptology ePrint Archive, report 2016/086, 2016.
- [10] D. J. Howarth, P. D. Jones, and M. T. Wyld, "The Atlas Scheduling System," in Proceedings of the AFIPS 1963 Spring Joint Computer Conference, vol. 23, pp. 59–67, 1963, doi: 10.1145/1461551.1461560.
- [11] IBM, IBM System/360 Operating System: Control Program with MFT, Y27-7128-3, 4th ed., May 1968.
- [12] IBM, IBM System/360 Principles of Operation, GA22-6821-8, 9th ed., Nov. 1970.
- [13] Intel Corporation, 80286 and 80287 Programmer's Reference Manual, 210498-005, 1987.
- [14] Intel Corporation, 80386 Programmer's Reference Manual, order no. 230985-001, 1986.
- [15] Intel Corporation, Control-flow Enforcement Technology Specification, rev. 3.0, May 2019.
- [16] Intel Corporation, Intel Virtualization Technology for Directed I/O Architecture Specification, rev. 4.1, order no. D51397-016, Mar. 2023.
- [17] P. A. Karger and R. R. Schell, "Thirty Years Later: Lessons from the Multics Security Evaluation," in Proc. 18th Annual Computer Security Applications Conference, 2002, pp. 119–126, doi: 10.1109/CSAC.2002.1176285.
- [18] T. Kilburn, R. B. Payne, and D. J. Howarth, "The Atlas Supervisor," in Proceedings of the AFIPS 1961 Eastern Joint Computer Conference, vol. 20, pp. 279–294, 1961, doi: 10.1145/1460764.1460786.
- [19] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in Advances in Cryptology—CRYPTO '96, pp. 104–113, 1996, doi: 10.1007/3-540-68697-5\_9.
- [20] P. Kocher et al., "Spectre Attacks: Exploiting Speculative Execution," in 2019 IEEE Symposium on Security and Privacy, pp. 1–19, 2019, doi: 10.1109/SP.2019.00002.
- [21] B. W. Lampson, "A Note on the Confinement Problem," Communications of the ACM, vol. 16, no. 10, pp. 613–615, Oct. 1973, doi: 10.1145/362375.362389.
- [22] S. B. Lipner, "The Birth and Death of the Orange Book," IEEE Annals of the History of Computing, vol. 37, no. 2, pp. 19–31, Apr.–June 2015, doi: 10.1109/MAHC.2015.27.
- [23] M. Lipp et al., "Meltdown: Reading Kernel Memory from User Space," in 27th USENIX Security Symposium, pp. 973–990, 2018.
- [24] T. Lyon, "[PATCH V2] VFIO driver: Non-privileged user level PCI drivers," Linux Kernel Mailing List, June 8, 2010. [Online]. Available: <https://lwn.net/Articles/391459/>
- [25] D. MacKenzie and G. Pottinger, "Mathematics, Technology, and Trust: Formal Verification, Computer Security, and the U.S. Military," IEEE Annals of the History of Computing, vol. 19, no. 3, pp. 41–59, 1997, doi: 10.1109/85.601735.
- [26] J. M. McCune, T. Jaeger, S. Berger, R. Cáceres, and R. Sailer, "Shamon: A System for Distributed Mandatory Access Control," in Proceedings of the 22nd Annual Computer Security Applications Conference, 2006, pp. 23–32, doi: 10.1109/ACSAC.2006.47.
- [27] F. McKeen et al., "Innovative Instructions and Software Model for Isolated Execution," in 2nd Workshop on Hardware and Architectural Support for Security and Privacy, 2013, doi: 10.1145/2487726.2488368.
- [28] NIST, Recommendation for the Entropy Sources Used for Random Bit Generation, Special Publication 800-90B, Jan. 2018, doi: 10.6028/NIST.SP.800-90B.
- [29] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and Implementation of a TCG-Based Integrity Measurement Architecture," in 13th USENIX Security Symposium, 2004, pp. 223–238.
- [30] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," Proceedings of the IEEE, vol. 63, no. 9, pp. 1278–1308, Sep. 1975, doi: 10.1109/PROC.1975.9939.
- [31] M. Schneider, R. J. Masti, S. Shinde, S. Capkun, and R. Perez, "SoK: Hardware-supported Trusted Execution Environments," 2022, arXiv:2205.12742.
- [32] M. D. Schroeder and J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," Communications of the ACM, vol. 15, no. 3, pp. 157–170, Mar. 1972, doi: 10.1145/361268.361275.
- [33] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in 44th ACM/IEEE Design Automation Conference, pp. 9–14, 2007, doi: 10.1145/1278480.1278484.
- [34] Trusted Computing Group, Trusted Platform Module Library, Family 2.0, Level 00, Revision 01.59, Nov. 2019.
- [35] R. N. M. Watson, S. W. Moore, P. Sewell, and P. G. Neumann, An Introduction to CHERI, University of Cambridge Computer Laboratory, Technical Report UCAM-CL-TR-941, Sep. 2019, doi: 10.48456/tr-941.
- [36] A. Williamson, "[RFC PATCH 0/3] IOMMU groups," Linux Kernel Mailing List, Apr. 2, 2012. [Online]. Available: <https://lwn.net/Articles/490284/>
- [37] A. Williamson, "IOMMU Groups + VFIO," Linux Kernel Mailing List patch series, May 11, 2012. [Online]. Available: <https://lwn.net/Articles/497009/>

- [38] A. Williamson, “[PATCH 0/7] IOMMU: Groups support,” Linux Kernel Mailing List, May 29, 2012 (May 30 UTC). [Online]. Available: <https://lwn.net/Articles/499239/>
- [39] A. Williamson, “[GIT PULL (PATCH 0/4)] VFIO driver for v3.6,” Linux Kernel Mailing List, July 25, 2012. [Online]. Available: <https://lwn.net/Articles/508088/>
- [40] A. Williamson, “Documentation/vfio.txt,” Linux Kernel Mailing List documentation patch, 2012. [Online]. Available: <https://lwn.net/Articles/509153/>
- [41] H. Xia et al., “CHERlvoke: Characterising Pointer Revocation Using CHERI Capabilities for Temporal Memory Safety,” in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), 2019, pp. 545–557, doi: 10.1145/3352460.3358288.
- [42] Y. Yarom and K. Falkner, “FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack,” in 23rd USENIX Security Symposium, pp. 719–732, 2014.
- [43] L. Zhao, H. Shuang, S. Xu, W. Huang, R. Cui, P. Bettadpur, and D. Lie, “SoK: Hardware Security Support for Trustworthy Execution,” 2019, arXiv:1910.04957.
- [44] Linux kernel, v3.6 tagged source tree, Documentation/vfio.txt, drivers/vfio/vfio.c, drivers/vfio/vfio\_iommu\_type1.c, and drivers/vfio/pci/vfio\_pci.c, 2012. [Online]. Available: <https://github.com/torvalds/linux/tree/v3.6/drivers/vfio>
- [45] Linux kernel, v3.7 tagged source tree, selected VFIO release paths and documentation, 2012. [Online]. Available: <https://github.com/torvalds/linux/tree/v3.7/drivers/vfio>
- [46] Linux kernel, v3.6 tagged source, Intel IOMMU implementation and linked IOMMU and PCI helpers, 2012. [Online]. Available: <https://github.com/torvalds/linux/blob/v3.6/drivers/iommu/intel-iommu.c>

Federico Pacheco is an electronic engineer and cybersecurity professional affiliated with Universidad Tecnológica Nacional in Buenos Aires, Argentina. His research interests include hardware-security history, computer architecture, and trusted systems. Contact him at [fpacheco@frba.utn.edu.ar](mailto:fpacheco@frba.utn.edu.ar).