Exploratory Investigation of Hardware-Software for Highly Assured Multilevel Security (MLS): From 1965 to 2035

SRI IRAD Project

Peter G. Neumann

SRI Computer Science Laboratory
Menlo Park CA 94025

December 24, 2024

Distribution Unlimited. This work was funded by an internal SRI IRAD project. It intentionally avoids all sensitive areas. To the best of my abilities, knowledge of everything is based on unclassified materials, all carefully cited. The views, opinions, and/or findings contained in this report are those of the author and his noted advisors, and should not be interpreted as representing the official views or policies, either expressed or implied, of the Department of Defense or the U.S. Government.

Contents

Chapter 1

Multilevel Security Revisited

1.1 Preface for This Conceptual Report

After 9 months of nattering over some high-level concepts, this final still seems like mostly potatoes rather than meat. This is not a surprise to me. I am attempting to establish a basis from which broad conclusions might be postulated as to what might be worth refining in the future some conceptual highly trustworthy multilevel-secure subsystems that could be developed and seriously evaluated. The assumptions under which they might be used need to be precisely specified and analyzed to prove soundness under the stated assumptions.

The security community has spent many years on the MLS R&D noted here. It would be a shame to throw it all away if there are gold suggests still lurking for serious pursuit. On the other hand, it would be a waste if the future could readily be deemed hopeless *a priori*. At the moment, I still remain a conservative optimist. Otherwise, I would not have undertaken the present study and the resulting document. I hope this effort makes it possible to draw realistic conclusions about the long-term future – even if it helps only to extend the trustworthiness of conventional systems and networks.

A fundamental question remains: Is this all a pipe-dream, or is there a realistic future for something modest, especially if developed on hardware such as CHERI that would be significantly more trustworthy than previously existing hardware with correspondingly more trustworthy software. Or, might that desire actually be irrelevant, and the notion of clean-slate Multilevel Security/Integrity hardware-software systems and networks is actually a colossal losing battle, which is the typical common wisdom of Butler Lampson ("Capabilities are the way of the future, and always will be.") and many others. (For for further examples, see Appendix Section C, including Steve Bellovin's outspoken pessimism and Steve Lipner's recollections.)

I hope we can converge on realistic answers before I run out of this year's support. I use the plural "we", because this draft has already listened thoughtfully to others with relevant experience, and hopes to have some further critical review (either openly or anonymously). PGN

1.2 Abstract: Goals of This Report

The primary goals of this report for an SRI IRAD study are (1) to document the major pitfalls associated with multilevel security; (2) establish incisive principles, guidelines, and methodology for MLS; and (3) encourage where feasible some future efforts at implementing and evaluating MLS to be orders of magnitude more realistic and more trustworthy than previous efforts – based on requirements that are more complete and more precise, a holistic architectural system perspective, development practices that are more comprehensive, assurance methodologies that are more easily applied, and whatever else might be necessary.

In particular, we suggest basing future work on the further development of the principle-based CHERI hardware-software systems and formal analyses in development by SRI and Cambridge UK. In short, this document tracks an analysis of the past into the what might have to happen in the future, with a quest for an over-arching clean-slate approach to total-system multilevel security for reasonable applications that are deemed feasible. For example, certain highly constrained operational environments might enhance feasibility, along with systems, subsystems, and networks that can be inherently more trustworthy than what is available today.

From the outset of my conceiving and organizing this report, several comments are worth noting here:

- This is a rather personal view, initially recollecting several ground-breaking clean-slate architectures with which I have been deeply involved over the past 60 years, along with an analysis of some of the past oversights and mistakes that arose in these and other would-be MLS system developments. It also extensively includes information provided by my informal advisors, especially where some of their personal comments help illustrate the inherent complexities that must be addressed.

- Surprisingly, many past constructive innovations have been widely ignored in practice, forgotten, or misunderstood. The most egregious example involves buffer overflows, for which Multics beginning in 1965 supplied effective hardware and software tools that could prevent stack buffer overflows – with almost no acceptance or even recognition by the computer-system industry or government procurers of trustworthy systems who can help set stricter requirements. Multics system programmers could use the facilities of Multics and languages such as PL/1 (actually, using the compiler Bob Morris and Doug McIlroy created for their Early PL/1 EPL stark subset that eliminated certain dangerous PL/1 constructs such as pointers), to develop Multics operating-system code that would consistently block out-of-bounds references. (Sloppy or malicious code could still access storage it should not; sloppy code could also clobber a pointer with garbage: the call that added a segment to the virtual-memory set-memory-access controls so the only thing in the process's memory image was data that the process to which the owner had rights. Furthermore, hcs$initiate returned a pointer only after checking process identity versus access-control list. The pointer was untyped, which

gave access to an entire segment; similarly, access control was on segment granularity.[1]

- One of the main Multics innovations was the programming discipline. The Triumvirate approach required approval by by the late Fernando Corbató, the late Charlie Clingen, and PGN (until 1969 when Bell Labs bailed out altogether) – based on reasonably careful documentation of all module inputs, outputs, functionality, and exceptions or intended error messages – before any EPL code could be written.[2] Unfortunately, the imposed discipline did not continue past the informal specification document into detailed early code review. (David Parnas subsequently made enormous contributions by formalizing the stages of his disciplined development approach (e.g., [65, 64, 66]). It is unfortunate that his insights did not find practical applications for system architecture until the next decade – for example, in PSOS and KSOS. Rigorous attention to his work could have significantly improved Multics.)

- Jon Callas reminded me of a meme that was popular after Multics was abandoned by Bell Labs: the Orange Book took over on how to design computer kernels, and Unix and C became dominant in the research communities: "Unix and the C programming language set the state of the art of security research back 50 years."[3] Jon added, "These days, I follow that meme up with saying that fortunately, we're now over forty years into the fifty. It's far more true with C than with Unix itself these days. Yet, I remember when I'd argue with Unix people, they did come around. Also, FreeBSD has helped significantly." Fortunately, CHERI has created a rebirth of research on memory safety, which is seriously lacking in C, and emerging in newer languages such as Rust (which unfortunately is not entirely memory safe, e.g., given its escapes to native code similar to Java).

- Despite its considerations of numerous previous secure-system developments, this document is intended as a prospective *view of the future*, with primary attention on how to identify and overcome past limitations and shortcomings. This is now possible with enhanced system/network trustworthiness based on hardware such as CHERI.

- A fundamental methodological realization is required: We cannot create a secure system, even if the entire team of developers working on the hardware and software is

[1] Tom Van Vleck (who runs the rather comprehensive historical Multicians.org website) has worked for people who really want secure computing on an insecure platform, for at least 45 years. "Telling them they can't have it is not an option. Telling them *why* they can't have it didn't work: they don't want a lot of details. Maybe we need to appeal to a higher-level concept such as "the gods do not permit it; Claude Shannon; or entropy". Or a movie plot." THVV

[2] The dynamic linker from symbolic file names to hardware-identified descriptors for each active segment seems to have been the only exception initially – it was written in assembly code language – until the programmer left after several iterations, when it was reprogrammed in EPL by someone else directly from the multiply-updated documentation. Surprisingly, the new code ran three times faster than the hand-optimized assembly code.

[3] This was also said of Windows undercutting all research in operating systems.

committed to *Zero Defects*; that extreme-programming approach is very rare, but almost completely precludes tradeoffs between correctness and speed, schedule, cost, convenience, etc.[4]

- Ultimately, the meaning of *trustworthiness* must be formals specified in terms of all the requirements that must be satisfied. The notion of trustworthiness is meaningless without such well-defined expectations. Whenever we refer to trustworthiness is mentioned here, that is assumed.

Tom Van Vleck suggested that the following questions be addressed here:

- What can you get from MLS? The ability to deal with information at multiple security levels, and programs at multiple integrity levels, without violating the fundamental rules – information never leakds to a lower level of security, and programs never depend on anything less trustworthy).

- What are the costs? The costs to do this in the open (e.g., across the Internet – which includes the Dark Web not playing by your rules) are insurmountable. The costs of something housed entirely in a controlled environment with appropriate shielding and compartmentalization could be manageable compared with the costs of losing what is being protected.

- What you can't get (e.g,, problems MLS cannot not fix)? such as physical attacks on hardware such as Rowhammer and other noise injection (e.g., [8]) and physical additions of analog hardware (e.g.,[100]), or even kidnapping key personnel for ransom.).

[4]Tom De Marco has written beautifully about the important influence of intending to produce zero-defect code. "The alternative to defect removal is defect abstinence." (Controlling Software Projects p. 207) Chapter 22 of Controlling Software Projects is on Zero Defects. De Marco says programmers program but are not allowed to use a compiler; programmers pass the source code to the test team who compile it; any compilation error means the program is defective and rejected; programming organization must assign somebody else to write the program' if it compiles, the program must then pass tests that the test org creates; programmers are typically not shown the tests their program has to pass.
Tom Van Vleck amplifies this point: Multics designers, managers, and developers did not have a complete commitment to Zero Defects. However, they did evolve a process that was better than most others, but not perfect. Bugs were included in shipped upgrades, but then fixed as soon as identified. Perhaps NASA comes closer to zero defects, but at the cost only about 3 lines of debugged code per day per programmer. Tom's earlier thinking on this topic is at https://multicians.org/thvv/nasty.html. However, he thinks he should have gone further in that essay. What would it be like if we started a project with Zero Defects as our plan? How could we get it right the first time, every time? PGN notes this was akin to the rather over-hyped IBM Clean Room approach advocated by Harlan Mills [52], where nothing would be accepted unless it had been developed under stringent conditions and strongly determined to be *clean*. However, Mills's approach and numerous other so-called informal methodologies were all incomplete, and most of them made no use of formal specifications and formal analyses suggested by Parnas, Edsger Dijkstra, Tony Hoare, Nicolas Wirth, et al.

1.3 Introduction

Some forms of Multilevel Secure Systems and Networks have been sought in one context or another since the late nineteen-forties, originally as a veneer on top of batch-processing computer hardware. The initial versions of RACF, ACF2, and TopSecret enforced some discretionary access, but no real sense of MLS. RACF later gave users the impression that successive jobs could be run securely at system high with complete purging the entire system memory between jobs. Each run was intended to be a clean stand-alone process, supposedly completely independent of all previous batch jobs. That knee-jerk approach was a very primitive approach, before the notion of time-sharing came along in MIT's CTSS (Compatible Time Sharing System), and other shared systems at Dartmouth, Berkeley, and elsewhere). Ever Since the early days of primitive time-sharing (e.g., users from one another became important, although the separation was often inadequate.

Multics was the first of three clean-slate hardware-software architectures to which I was a major contributor. The GE 636/645 hardware (as a major augmentation of the GE 635) had been designed (by Ted Glaser at MIT and John Coleur at General Electric and Honeywell with extraordinary foresight on the part of Ted, to support rigorous separation of users and their data via true virtual-memory segmentation and paging, dynamic linking of symbolically named entities (files/segments and then Ken Thompson's redesign of the I/O system with Multics input-output streams that anticipated his UNIX pipes), as well as integrity of the operating system via layered levels in the operating system. All of that OS-inspired hardware thinking led to at least 3.5 decades of Multics evolution, based on the Multics principles first established in May 1965.[5] Multics was originally documented in five papers at the Fall Joint Computer Conference in Las Vegas in November 1965 (e.g., [12, 13]). All five papers are on https://multicians.org.

In the following decade, the Trusted Computer Security Evaluation Criteria (TCSEC) Orange Book [54] attempted to seek different levels of security, with formally assured multilevel security being the ultimate goal. The Orange Book C levels pushed IBM (in moving batch operating systems from C1 to C2 into handling complete deletion as opposed to marking data items as deleted (from which undelete [undo] was still possible). This was a problem not just for batch processing, but also for the earliest IBM time-sharing systems. The B levels required MLS software security kernels with ¡explicit labeling of classified information. The A and B TCSEC levels required an isolated operating system kernel with more structure and minimization of trusted code and more formal assurance measures as one went from level to level above B1. The A2 level introduced formal requirements and formal specifications, and A1 required formal proofs, respectively. However, the Orange Book attempted to provide the security primarily through the *software* security kernel and trusted computing base, presuming that the hardware could not be compromised.[6] The dependence primarily

[5] Even though the Multics clock ran well past 1 January 2000, the system as a whole was not intended to be strictly Y2K-compliant, as noted by Steve Lipner.

[6] Hardware security was required or expected during the Orange Book days in the sense that the hardware provided controls that they worked as expected. See, for example, Olin Sibert's IEEE Symposium on Security and Privacy (Oakland) paper [83], which found x86 flaws.

on software kernels (and trusted computing bases that might provide overrides) is one of the primary assumptions this report seeks to overcome, as it is clear today that if the hardware is untrustworthy, the operating system and mission software cannot be trustworthy.[7][8]

Other deficiencies of the Orange Book and the Rainbow series of successive interpretations are also addressed in my 1990 paper, Rainbows and Arrows: How the Security Criteria Address Computer Misuse [56], and in an independent paper by Willis Ware [89] five years later.

A historical retrospective by Steven B. Lipner is also very worthy: See reference 46 in The Birth and Death of the Orange Book: https://www.stevelipner.org/links/resources/, http://www.bitsavers.org/pdf/sdc/adept-50/Lipner_-_The_Birth_and_Death_of_the_Orange_Book_2015.pdf

> Over the past 50 years, U.S. government computer security strategy has shifted focus from government-funded research and system development to evaluation of commercial products. By tracing the history of the Trusted Computer System Evaluation Criteria (TCSEC) or Orange Book during this period, this article covers the role of government agencies, vendors, and policymakers in determining IT system security requirements and development.

Roger Schell pioneered the Rainbow Series when he was at NCSC, and also spent many years working on the Gemini GEMSOS MLS security kernel. (More recently, he had been trying to resurrect it.) GEMSOS had 8 Multics-like rings.

Other approaches to security kernels were developed at the University of Newcastle (Secure Distributed System [74, 75]), Honeywell (Scomp [21] and LoCK/Secure Ada Target [78]), Hydra (the MLS kernel of a multiprocessor system developed at Carnegie-Mellon [97, 98]), SDC [25] (KVM/360, with 4 MLS levels and 62 compartments), to cite just a few other earlier software kernels with some degree of MLS.

Another problem is that many of the advances pioneered by Multics were more or less lost, including its integrated hardware-software security (including defense against stack buffer overflows, as noted above– which was effectively understood by Multicians in 1965 –

[7] The TCSEC did implicitly reference hardware state separation, partly because of Multics penetration efforts (e.g., by Roger Schell, Dan Craigen, Marv Schaefer and others). There was also an implicit reference to incomplete hardware mediation that necessarily had to be corrected by the software prior to control being shifted to the hardware for execution: e.g., in the SDC/IBM security study of VM/370 (1980-1981), it was found that the virtual address in a Channel Control I/O program could potentially (e.g., explicitly) transfer the actual address rather than virtual address to the numerous I/O device specific controllers to access all physical addresses. In retrospect, the semantic insights required for sanitizing I/O would forevermore expand the size and complexity of the VM kernel!
Marv Schaefer noted that Peter omits the implicit reference to a state separation in the hardware between the reference monitor and the untrusted user populace. That state separation was relied upon even in OS/MVS/MVT to isolate the access-control mechanisms from user state.

[8] Steve Lipner was very familiar with the early-1970s Multics penetration test that preceded the development of AIM and the AFDSC deployment. There are references later in the paper to testing as part of the B2 evaluation in the 1980s, and that was completely separate. Multics was already operational at AFDSC. Paul Karger should be named here. Note that the Multics penetration test did discover a hardware vulnerability.

and a very strong hardware special-purpose co-processor (GIOC) to manage the input-output security problem of having access to absolute addresses rather than paged virtual memory.[9]

Fortunately, these problems are now being confronted by our SRI-UofCambridge CHERI security model and matching hardware-software [94], all of which provide a relevant base platform for some of the approaches to MLS outlined in this report.

A variant security policy is the *high watermark policy*, whereby each object becomes contaminated by the highest (and possibly compartmented) information the running process acquires, which then effects everything that is written.[10]

One other wrinkle: any advanced multilevel security system should be general enough and trustworthy enough that it can handle multiple policies (e.g., the U.S. Department of Energy classification system, and variants in other countries). Thus, the focus on the U.S. intelligence and national security classification system should be considered illustrative of what might reasonably be accomplished with adequate assurance.

We are all confronted with the fundamental problem that our critical systems (hardware as well as software) are not sufficiently trustworthy. Incremental improvements have demonstrated themselves to be seriously inadequate. Flawed hardware upon which flawed software is built does not allow a long-term strategy for success, especially when deployed in systems with ultra-critical trustworthiness requirements (e.g., for total-system/network security, human safety, survivability, real-time performance, interoperability, usability, and other essential attributes).

1.4 An Intuitive Formulation of MLS

Beginning with the Orange Book, the so-called Rainbow Series [54] implicitly assumed the dependence on the originally convoluted Bell and LaPadula model of MLS [4] and later updated to its much more accessible subsequent Multics interpretation [5]. Rather than work our way through those formulations, I prefer to give my own somewhat oversimplified version to give a relatively understandable intuitive view of the MLS and other principles involved.

Intuitively, MLS seeks to enable information to safely and assuredly coexist at different security levels and compartments without in any way leaking any information to lower security levels or collateral compartments: Basically, **information should never leak to or in any way influence anything at a less trusted security level** (e.g., with levels such as Top Secret, Secret, Confidential, and Unclassified) and collateral compartment.[11]

[9] This is a problem that has resurfaced in CHERI's approach to control dozens of hidden undocumented and proprietary microcontrollers, discussed in Section 1.7.4.

[10] There is a granularity issue: if my process had a SECRET segment open, but never fetched a bit from the page, is might nevertheless still be tainted.

[11] MLS has been viewed by B&LP as having two properties – **no read-up from a higher level** (or incomparable compartment and also **no write-down to a lower level** (or incomparable compartment). However, that seems to conflate the former *confidentiality* property with the latter *integrity* property, and I vastly prefer my single simplified leakage property, which leaves more room for associated integrity properties (considered below).

The mathematical simplicity of the linearized (or actually lattice-based) security levels is often complicated by restrictive *caveats* (such as NOFORN or NUCLEAR, or even the rather unwieldy CONTROLLED UNCLASSIFIED INFORMATION restrictions), as well as classified arbitrarily assigned special names for the strictly compartmentalized *categories* at secret and top-secret levels. (These caveats either break the natural simplicity of the basic lattice model, or make it unwieldy – e.g., multidimensional in the presence of myriad categories and caveats that are no longer a strict-sense lattice model.)

Although not explicitly visible in the TCSEC, Ken Biba came up with a sort of orthogonal dual model for Multilevel Integrity (MLI) [7]. In its intuitive sense, multilevel integrity attempted to enforce hierarchical levels of integrity with the property that any derived computational entity should never depend on any less trusted entities (i.e., with respect to the integrity levels (e.g., the simplest being merely high vs low integrity), suitably defined.

One of the problems that arose in trying to implement MLS and MLI at the same time was that certain conflicting requirements arose that had to be resolved. Basically, if pursued rigorously with integrity levels and confidentiality levels, such an approach tends to break most existing apps, and requires redesign. Consequently, MLI retained an amorphous role. However, the Multics system ring-structure integrity levels – in which by design each ring depended only on inner rings but could not compromise them – predated the Biba report by 10 years, and later became a strong contributor to the trustworthiness (security plus integrity) of MLS Multics. The Multics ring-structured integrity design layers actually mirror Biba levels of integrity compatibly.[12]

Also relevant is a paper by Roger Schell and Dorothy Denning [81] that suggests the Biba model is really an upside-down Bell and LaPadula. That paper also mentions Multics rings, the low-water-mark policy, the problem with having both integrity and security policy at the same time, and covert channels (see Section 1.9.13).

For readers not familiar with the history of MLS, Tom VanVleck suggested that we consider what MLS and system security imply to a person well-versed in the state of the art:

- Rigid security rules can be automatically enforced.
- Classification of Information can be achieved by marking the container.
- A clearance process determines to which levels users may be authorized.
- Structured levels and categories can be defined.
- Mandatory Access Control (e.g., MLS) can be built into computer systems.
- Additional Discretionary Access Controls (e.g., Multics access control lists (e.g., [13] or UniX-like RWX codes) are also possible.

[12] Note the confusion of the Multics ring structure, which is actually a one-dimensional ordering, rather than the apparent two-dimensional ring. However, it is graphically appealing, because compromising an inner ring is forbidden.

- Users with different clearance levels may coexist within the same computer system or network environment.
- Security officers require efficient automatic enforcement of the rules, rather than ad-hoc controls.
- Sensitive information often results from the accumulation of less-secret information, which necessitates its own rules. But the fewer rules, the better from a management viewpoint.[13]

1.5 Software Kernels on Conventional Hardware

The absence of trustworthy hardware (i.e., relying only on existing hardware) and the inherent incompleteness of the requirements for overall system security and multilevel security have seriously impeded the development of trustworthiness in any realistic sense of multilevel security. In particular, the absence of assured satisfaction of the MLS properties has seriously hindered attempts in the past to develop multilevel-secure systems. Furthermore, almost all of the work based in the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC, also encompassing the entire Rainbow Series) implicitly assumed that the hardware could not be compromised, and made very few references to vulnerabilities resulting from hardware problems.[14]

1.5.1 Software Security Kernels in the 1970s and 1980s

A National Academies Air Force Study Group MLS Data Base Management effort in 1982-1983 led by Marv Schaefer [79] included a chapter by Dorothy Denning and PGN [80] that sprung from a our two-person research subgroup that postulated the would-be development of an untrusted DBMS resting on top of an MLS kernel actually being multilevel secure – despite the untrustworthy nature of the DBMS.

Roger Schell of course wanted to pursue an approach relying on a trusted security kernel and untrusted everything else, which became the model for SRI's SeaView. Marv Schaefer also worked on a trusted DBMS project for AF Rome Labs that pursued the same idea back in the mid-1970s. Intriguingly, the Denning-PGN research subgroup belief led directly to

[13] This is both true and difficult, or perhaps impossible to contend with in a structured or automated way. Steve Lipner

[14] MLS is a wacky idea anyway. MLS is a very elaborate Rube Goldberg construction with many complex parts. It is a fine example of the First Law of Military Procurement: The Customer Is Always Right. That is, The Customer is not required to explain their entire reasons for what they asked for. They may have many layers of reasoning that they need not explicate. The design organization has to say, "No matter what the customer asks for, invent something that does that."
MLS says "an authorized person who reads a classified file that contains a secret cannot write an unclassified file, except that they can memorize the secret, log out, log in as unclassified, and write whatever they remember." This looked like a big hole in MLS to me. But of course, I don't know how they evaluate risk and how they ensure that clearances are awarded only to people who would never do that naughty thing. THVV

SRI's SeaView [47, 46, 44] and formal verification [96, 69] and security model [45]: with an MLS kernel (originally going to be Roger Schell's A1 candidate GEMINI kernel) underneath an Oracle off-the-shelf DBMS, although it was eventually replaced with a compartmented-mode workstation (Section 1.5.3) when GEMSOS was not ready in time.

Oracle's early DBMS for SeaView shared a single buffer space among all users, which immediately had to be changed to user-specific working buffers, to avoid the monster inter-user leakage channels.[15]

The KSOS [6, 50, 67] MLS kernel underwent analysis from SRI's MLS analyzer – initially Richard J. Feiertag and then Neumann did the formal analysis of the KSOS SPECial specifications using the Feiertag flow analyzer and the Boyer-Moore Theorem Prover to analyze whether the specs did or did not satisfy the Bell and LaPadula model. At first, something like 16 of the original 34 kernel-command specifications had problems. Several covert information channels (both storage channels and timing channels) were discovered, along with MLS flaws in the specs. Some of these were easily fixed with minor changes to the specifications. Others required rethinking the kernel design. Most of these flaws were eventually fixed.

The closest approach to real MLS resulted from the effort Steve Lipner led, the VAX SVS (Digital Equipment VMM Security Kernel), which Paul Karger and others worked on in the 1980s. It got most of the way through A1 certification before Steve canceled the project for lack of a sufficient market. (The documentation was all proprietary to DEC, and now in the archives of Tandem. We have been unable to locate its whereabouts, hoping to get it released for public use.) See Steve Lipner's remarks in Appendix C.

UCLA [88] was also involved, although less oriented toward MLS, if I recall correctly. Also, a few special-purpose kernels such as the RAP [68] (Restricted Access Processor) Guard and the MITRE guard [85], and special-purpose sanitizers and downgraders were developed. As it turns out, formally proven A1 kernels were relatively rare at the time, and a few came along a decade later (albeit with rather limited sets of requirements).

1.5.2 More-Recent Software Security Kernels

CertiKOS [27]is the closest kernel to the hierarchically layered hardware-software PSOS methodology![20, 72, 73, 84], because its formal proofs use a methodology akin to the SRI HDM hierarchical proof structure – proving one layer at a time from the bottom up – although applied only to the software kernel. HDN

The Australian efforts are quite remarkable relating to an evolution with extensive proofs for L4 [30] and seL4 [37] software kernels. Their recent work is also exploring multilevel security as well.

See appendix Section A.2 for an ongoing adaptation of seL4 ported to CHERI hardware to increase the security coverage of seL4. It could eventually also include the recent

[15] Steve Lipner correctly recalled that the main body of the effort had a lot of push for an approach based on encrypting or maybe signing data stored in an untrusted DBMS and then using a trusted filter that would authorize access if the signed labels matched the user's authorization. Unfortunately, the approach was subject to various covert channels in addition to the shared-buffer problem.

MLS work in Australia. This work is being done by Cambridge UK PhD candidate Hesham Almatary under the aegis of Robert Watson (wearing his hat as head of Capabilities Ltd). (Hesham was previously part of the L4/seL4 team.)

1.5.3 Compartmented-Mode Workstations

As sort of a half-way measure, given the complexities of MLS, Compartmented-Mode Workstations were a compromise. A document by Gary Huber (MITRE), CMW Introduction https://dl.acm.org/doi/pdf/10.1145/190748.190750 evaluated five products at an intermediate informal TCSEC level with B1+ requirements (somewhat stronger than B1, but still weaker than B2): DEC Ultirx MLS+, Sun Trusted Solaris, SecureWare MLS+ (A/UX base), IBM (trusted Xenix, AIX RS6000), and Harris Addamax ACMW (System V). This level had its own Rainbow series set of evaluation criteria document.

1.6 Principles and Requirements

One of the most important aspects of would-be trustworthy systems involves the definition of – and adherence to – underlying principles whose observance tends to starkly diminish vulnerabilities and risks relating to security, reliability, human safety, predictable system behavior, real-time guarantees, and so on. Extensive use of formal requirements, formal architectural specifications, formal foundations for programming languages, and formal analyses of all of these throughout system design and development also can provide much higher evidence-based assurance, especially wherever most effective. Indeed, a formal theory is needed for each required system property, such as *system security/network, system/network integrity, crash-safety, bounded performance, ...* However, formal methods by themselves are never a panacea.

The sequential development of system principles is outlined further in Section 1.7.1 as they emerged from the Multics effort, in Section 1.7.3 extended for PSOS including significant dependence on the contemporary work of David Parnas, and then in Section 1.7.4 as they were adapted for CHERI.

These principles were also invoked for my DARPA project on predictable high-assurance composability [57] in the early 2000s, which later was refined to become the basis for my 2017 ad-hoc analysis of the inherent principle-based trustworthiness underlying the entire development of CHERI [58] since 2010, noted in Section 1.7.4.

Revisiting all of these trustworthiness principles, an up-to-date summary is included in Table D.1 in Appendix C.

Indeed, principle-based development has to be an iterative process – pervasively followed through out design, development, implementation, bug-fixing, and upgrades. This is also a huge management challenge, defining the principles, refining and extending their statements and interpretation, and observing their application. A major challenge of future will be applying all this to MLS implementations, which themselves must be defined precisely. Are

these definitions formalizable and consistent? Are there cases where the official legal requirements are unclear or lead to major inefficiency or undefined behavior?

Underlying the existing CHERI security model and hardware ISA specifications are multiple principles [58], most notably the Principle of Least Privilege and the Principle of Intentionality (allocating the most appropriate set of privileges), along with principles relating to separation (such as separation of user information, separation of state and encapsulated abstraction, separation of duties, and functional isolation), and pervasively overarching – the Einstein Principle that **Everything should be made as simple as possible, *but no simpler*.** All of these principles are highly relevant to the design, development, and operation of MLS systems and networks – in observance throughout.

Also fundamental to MLS hardware architectures and software developments are specific requirements that must be met. This is of course necessary for multilevel security, where superficial adherence to the Bell and LaPadula security model is not nearly enough. Hardware specifications are critical to analyses of ISAs, and software specifications are critical to implementation and analyses of the systems. Sound software engineering is of course also expected to minimize software errors during development.

A weak alternative to principled construction of hardware and software appears to be what is euphemistically called *best practices*, which are inherently incomplete and nowhere near good enough.

Another approach is the MITRE-NIST collection of Common Vulnerabilities and their enumeration (as CVEs). To illustrate some of the futility of tracking vulnerabilities, there were 130,000 CVEs in April 2000; four years later, there are over 200,000 CVEs. This is not converging. Patch and Pray is not a viable approach.

1.6.1 Comments on Government Intervention

The Orange Book and the Rainbow Series represented a remarkable intervention. Getting from C1 to C2 was a basic step forward for IBM, with their ability to mark as deleted, but also to undelete. However, emphasis on multilevel security was perhaps premature – especially the notion of having all of the security in the software kernel. However, in some eome sense it also raised exercise in better system software that was in some ways more secure. However, Steve Lipner's cancelation of a million-dollar MLS project for which there was apparently no marketplace followed the aftermath of the Multics Dockmaster system. See Steve Lipner's comments in the Appendix.

1.7 Principled and Formally Based Clean-Slate Systems

Several clean-slate architectures are particularly relevant here, as examples that began with strict adherence to established principles and/or the use of formal methods. As noted above, each of them is rather close to my own professional experience, and deeply influenced by my November extended-breakfast session with Alber Einstein in one way or another relating to complexity.

1.7.1 Multics

The Multics development approach created a highly structured environment, with attention to principles (which were enforced from the very beginning; they were stated in 1965 (by Ted Glaser and PGN) in the first chapter of the Multics Programmer's Guide, discussed more broadly by PGN in the second ACM conference on Principles of Operating Systems (The Role of Motherhood in the Pop Art of System Programming) in 1969 [55], and then later codified from their Multics experience in 1975 by Saltzer and Schroeder [77] – and updated further in 1989 [76]) as general system-development principles that evolved from teaching computer science and operating systems at MIT. These principles were also fundamental to the CHERI hardware-software security model and implementations, which appears to be the most principle-based system yet developed.

2 The Multics hardware designed by Glaser and Coleur (noted above) was prescient in that it anticipated needs of the operating system. In the first five years beginning 2 January 1965 (when I was heavily involved), no software was permitted to be written without a detailed informal spec regarding all of the inputs, outputs, hidden state information, and exception conditions. The design was highly modular, with strict encapsulation of internal state. This was a precursor to later work of David Parnas for formal specifications of modular decompositions. (See [57] for an excellent elaboration of that effort. Rather than listing all of Parnas's important contributions here, see the appendix contributed by Virgil Gligor to my 2004 DARPA report [57] on compositional assurance.) The relevance of those principles to the onging CHERI architecture and system development is noted in the following Section 1.7.4 on CHERI.

1.7.2 MLS Multics

The retrofit of MLS into Multics began in the early 1970s, with a little pushing from NSA, and leading to the Access Isolation Mechanism (AIM). The encapsulated modularity resulting from the Multics ring structure was a fundamental factor in the ability to retrofit MLS [82] into the original 1960s Multics hardware-software design and implementation [63], prior to the MLS retrofit and the 6180 hardware replacing the 645. The retrofit added credibility for the ring-layered system integrity, where ring $n+1$ could not compromise ring n, because very few changes to the original ring layers were required for the MLS retrofit [82]. The MLS implementation was entirely in Ring 0, supporting 8 arbitrarily defined linear MLS levels and 18 categories. MLS Multics (AIM) was also subjected to a detailed security evaluation [34, 35]. Although no formal methods were used in the development, the formal bases for the Bell and LaPadula model were relevant to the evaluation. The performance hit over the baseline Multics was surprisingly small.[16]

[16] One of the curious quirks is noted in https://www.multicians.org, involving the `extend_high` primitive, which in my mind is a terrible compromise. According to THVV, The solution to this problem contained two elements. First, the forum subsystem was modified to allow read access from levels dominating the sensitivity label of any forum (if permitted by the ACL, of course). Next, an extended category called `extend_high` was created that was a superset of all categories. By using the `extend_high` category at the proprietary level

Tom Van Vleck had this commentary relating to covert information channels:

> In the 1970s, Multics had some timing channels: https://www.multicians.org/timing-chn.html At that time, we had nothing from NSA about any of this.
>
> In the 1980s, the Orange Book B2 spec mandated covert channel analysis, and Keith Loepere did one for Multics (published in SIG-OPS?).
>
> Multics MLS added level+category info to files and file system directories, and every file-system operation checked against user privileges.
>
> Login accepted level and category arguments, validated against registration data. The file system implemented checks to prevent storage channels, e.g., quota/usage. there is a manual on this, will send link.
>
> From https://multicians.org/b2.html:
>
> The Trusted Facilities Manual required for B2 certification is contained in Part VI *Assuring System Security*‘ and Appendix B *Audit Tables and Include Files* of AK50-03 (Renamed the *Multics System Administration Procedures Manual*, May 19 85). Part VI consists of Chapters 18 through 26 of the manual, and provides guide lines for the system administrator on how to manage Multics as a secure system. [info from Ed Ranzenbach] (21 MB pdf, 408 pages)
>
> See pages 18-25 ff for info on the Access Isolation Mechanism (i.e., MLS): http://www.bitsavers.org/pdf/honeywell/large_systems/multics/AK50A-03B_sysAdmin_Dec87.pdf

1.7.3 PSOS with MLS

The NSA Research directorate funded the PSOS project from 1973 to 1983 (under R Director Doug Hogan and Hilda Faust). The first seven years were devoted to the design of *Provably Secure* hardware and software, formally specified, and intended to have proofs upward from the 7 hardware layers and through the software layers. The iconoclastic nature of PSOS

in conjunction with forum's new capability of being readable from dominant AIM classification levels, an evaluator or an evaluation manager could check all meetings with a single interactive session. The supercategory also provided a way for the product evaluation community to provide separation from all vendor-specific data for its internal discussions of issues pertaining to all product evaluations.

So, a manager could have `extend_high` and look at all their company's products in evaluation. Also managers could have meetings to discuss general evaluation issues. Managers logged in to this clearance would be able to write things only at `extend_high`. They could read data with any extended category, subject to ACLs.

Introducing the `extend_high` category was a clever way to identify some trusted individuals who could see, but not write, status info, and to add a small feature to forum to make their lives easier. This seems to be cop-out, because you should not be able to see anything at a higher level that you can make use of one way or another. As we say sarcastically in the ACM RISKS Forum, What could possibly go wrong?
Added note: `extend_high` was a local change made only at the Pentagon's Dockmaster. It was not discussed with the system designers or the B2 Orange Book designers and implementers, because Honeywell Bull laid them all off, probably 10 years before the NSA guys started hacking on the forum software. THVV

(with respect to the Orange Book) was that one of the upper software layers was that the multilevel-security layer, specified in terms of a strongly typed object-capability layer in which the capability mechanism and the rest of the hardware supported strongly typed MLS objects, somewhat different from the realm of the Orange Book, but totally within the notion which I recently outlined in terms of *total-system trustworthiness*, with the desire for proofs from the bottom up all the way to user and mission requirements [59]. The 1980 final PSOS report is online [60], and was previewed in an NCSC paper [18] and revisited in an ACSAC Classic Papers article [61].

PSOS was a highly principled paper study, based heavily on the principles of David Parnas. Any work on formal proofs of the hardware and software was decades ahead of the field. [17]

A fundamental theoretical basis for PSOS involved the SRI Hierarchical Development Methodology (HDM [72, 73, 84], which encouraged a system to be specified in layers in a non-executable formal language (SPECial), each layer of which had encapsulated modules, and each layer included an abstract implementation in terms of the visible primitives of lower layers. Thus, the MLI property of of Multics was explicitly part of the design. The 1980 Final PSOS report contained the formal specifications and abstract implementations of each layer; layers 0 to 7 would have been in hardware, the rest in operating systems and software. The bottom layer zero contained only two hardware instructions – one to create a new capability with specified privileges for accessing a given virtual-addressed Multics-like segment, and the other to create a restricted version of an existing capability. The rest of the PSOS hardware and software was specified in higher layers building upward from those two instructions. MLS was introduced at a software layer in which conceptually formal proofs could be constructed from the bottom layer up, one layer at a time.

A skeletal structural PSOS abstraction hierarchy is shown in Table D.2. Although this might look inefficient, a user command executed as a single instruction, invoking capability instructions as the conceptual layer zero. once the dynamic linking had been completed, the segment paging had taken place (similar to the Multics dynamic linking of symbolically addressed segments). The bottom seven layers in the figure were intended to constitute the PSOS hardware ISA, at the presumed hardware-software boundary for any subsequent hardware. In addition, some of the properties that might have been formally proved about the design specifications are summarized in Table D.3.

1.7.4 CHERI

The multiple CHERI projects since 2010 (DARPA I2O and MTO, and additional recent UK funding) have led to hardware that is much more trustworthy than any predecessors. CHERI can overcome many of the historical shortcomings noted above. CHERI appears to be the most principle-based system ever conceived, respecting a more extensive detailed set

[17] The last three years of the project resulted in the two Goguen-Meseguer papers [23, 24] on a newly formulated multilevel property, *noninterference*, as well as the first rudiments of what is now SRI's PVS formal proof system. These two papers were largely inspired as a formalized outgrowth of the earlier work by Feiertag, Levitt, and Robinson [20].

of principles [58], adherence to which has greatly reduced the vulnerabilities so common in other hardware/software systems. Although several of those principles were fundamental to the CHERI hardware-software architecture from the beginning (e.g., least privilege and intentionality), other principles were actually satisfied without their being explicitly required – because of the disciplined and principled development.

The CHERI website https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/ includes a variety of released reports. Most important is Version 9 of the CHERI Instruction-set Architectures; it includes specifications for CHERI-Arm-Morello [92, 1, 3, 2] and various CHERI-RISC-V specs, plus a CHERI-x86 user-layer ISA Specification [94]. The website also has numerous published papers, typically considering solutions to problems such as the buffer-overflow problem originally addressed in Multics, and once again the result of the total-system architecture in CHERI:

Two aspects of CHERI are particularly relevant to its eventual use for implementations of multilevel security: CHERI's trustworthy memory safety and compartmentalization, in hardware and software.

- **CHERI memory safety:** We make the usual distinction between spatial memory safety and temporal memory safety. Violations of spacial memory safety involve dereferencing out-of-bounds memory addresses, as typified by buffer and bounds overflows, made possible particularly by programming languages such as PL/1 and C not treating pointers improperly. Violations of temporal memory safety involve improper deallocation of pointers, as well as issues such as use-after-free residues that have not been completely deleted or disconnected from potential access. Both types of memory-safety vulnerabilities are anathema to MLS implementations, and can lead to serious violations of MLS properties. See [1] and the multi-authored paper, CHERI: Hardware-Enabled C/C++ Memory Protection at Scale, an invited paper for the IEEE Security and Privacy Journal special issue on memory management, July-August 2024 [93].

  As Brett Gutstein notes in his PhD thesis [29], the currently existing CHERI still needs two major advances: (1) support for modern supposedly memory-safe language interpreters (e.g., for Java and Rust) – see Appendix Section A.8, and (2) providing temporal safety for user heaps, not just for the stack and the security kernel.

  A new paper by Brett is currently in progress on these topics (although he is currently involved in a major commitment for his startup, and is also working with Capabilities Limited in Cambridge UK on the DEC large-scale compartmentalization project).

- **CHERI compartmentalization:** A very useful introduction to this is given by Dapeng Gao and Robert Watson, Library-based Compartmefalisation on CHERI in a video on the CHERI website:// https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/gao2023librarycomp Numerous papers have been written on CHERI's approach to trustworthy compartmentalization [28, 95, 10].

  For the reader's convenience, Appendix Chapter B of this report includes a summary of our CHERI team's history with respect to compartmentalization.

- **Thunderclap:** This paper illustrates the vulnerabilities resulting from inadequate protection against malicious UCB-C sticks [48], and is a predecessor of the ongoing work on providing CHERI's approach to protection against microcontrollers with direct memory access (DMA). This problem was recognized and controlled all the way back in the Multics GIOC, with almost nothing comparable between the 1960s and now. A paper in flight on the proposed CHERI solution – a capability-based interposer that deals with committed absolute addresses – is nearing completion. Its generalization to protect DMA from otherwise uncontrolled embedded microcontrollers will guide our approach to permissioning is the as-yet undocumented capability-based interposer to replace the IO-MMUs still being used in other hardware. The interposer uses selected absolute addresses under highly controlled and eventually verified permissions on read-write of main memory from all of the otherwise DMA-permitted microcontrollers. See Section 3.11.4 of the Version 9 ISA specification document for the most recent discussion on CHERI DMA, although more recent discussion of some of the engineering implications is in progress. Furthermore, a new paper has just been submitted to ASPLOS in July 2024: Enabling CHERI Hardware Security for Heterogeneous Hardware Accelerators, with Theo Markettos as the primary author.

- **DMA Lessons from Multics Input-Output**

  When Multics transitioned from the 645 to the 6180, the new system did not use the GIOC. Instead, it used a standard product I/O controller, the IOM. The IOM had a hardware feature that supported base and bounds registers for each desired I/O channel. (Later iterations of the IOM added *paged-mode* support and other features.)[18][19]

  Multics re-implemented the inner levels of the I/O system to use the IOM, and provided security features that allowed secure I/O usage and user-ring device control. This software was called the GIM. It had several drawbacks and inefficiencies, and was not able to support executing Test and Diagnostic programs on the IOM and attached peripherals and peripheral controllers. Noel Morris proposed an improved design, the I/O Interfacer (IOI), which interacted with Multics memory management to handle buffer management, wiring and unwiring, process isolation, etc. This enabled Multics to move device drivers out of the hardcore into the user ring, with no performance or security degradation.

  – Ideas for a GIM replacement (31 Dec 1973)
  https://multicians.org/mtbs/MTB-028.pdf

[18] The Paged-mode IOM was used by Multics: https://web.mit.edu/multics-history/source/Multics/mdds/mdd012.compout describes the IOM and IMU successors to the GIOC, and the way IOM page tables differ from CPU page tables. It also mentions security related features of peripheral controllers (MPCs).

[19] I'm not sure that the requirement to choose a level before creating input is user-acceptable although it's what we in effect did in SVS (and Multics for that matter). A user-accessible and audited facility for downgrading after creation is a practical necessity.
I believe that CMW model allowed for multilevel documents with individual more or less trusted levels per paragraph? Better usability although, as I said, the CMW model was not actually secure. Steve Lipner

  - I/O Interfacer Design Specifications (27 Mar 1974)
    https://multicians.org/mtbs/MTB-056.pdf
  - I/O Interfacer Access to Disk (4 Feb 1975)
    https://multicians.org/mtbs/MTB-158.pdf
  - I/O Interfacer Specification Changes (25 Mar 1975)
    https://multicians.org/mtbs/MTB-178.pdf

  Later versions of Multics hardware replaced the IOM with an Integrated Multiplexer Unit (IMU), which contained a Maintenance Channel Adapter (MCA) and Integrated Peripheral Controllers (IPCs). IOI was updated to use them.

- **C**HERI Speculative Execution Contracts, attempting to provide explicit agreements on what can or cannot be speculatively executed, even if exploitation requires side channels to acquire the speculated information or to invoke an executable. (Note that no collaboration is generally required in a side channel – it just merely exists and is effectively transmitting or otherwise available. On the other hand, covert channels typically may require willful collaboration between the sender and the receiver.) Speculative execution in an MLS environment is considered further in Section 1.9.12.

- **Formal proofs** of the CHERI-Arm-Morello instruction-set architecture [62], showing that the basic capability-based security properties (least privilege, nonbypassability, and monotonicity relating to non-increasing virtual memory size and permissions, as in the PSOS capabilities).

1.8 Extended Requirements for MLS

Elements of all of the hardware-software contributions noted into the previous subsections will be critical to any future efforts regarding multilevel security and integrity, where the requirements for total-system security must be considerably more stringent than implied by the software security kernels of the Orange Book. This must include all sorts of information leakage channels (including collusion-required covert storage channels and covert timing channels, and side channels (which require no collusion), plus review of all shared resources – for which "every case is a potential channel"), speculative execution (Section 1.9.12), DMA information extraction channels, and of course the primary MLS property of no leaking from higher to lower security levels (whether covert or not). Also in scope are covert integrity channels such as malicious or adverse actions via direct memory access (DMA), Rowhammer, and inserting Trojan horses in hardware as the result of stray analog circuitry (e.g., [100]), noted above in Section 1.7.4. The requirements for multilevel security thus considerably transcend the Orange Book A1 requirements, and must address both hardware and software. Leakage channels are a particularly annoying risk, and considered in Section 1.9.13.

All of these concerns are potentially critical for MLS in general, although certain vulnerabilities and information channels may be less of a threat for systems that exist wholly within a tightly controlled physical enclave. This is an issue that is considered more thoroughly in Sections 1.15 and 1.16.

1.9 Capability Architecture Issues for MLS

The CHERI team's experiences with the constructive benefits of the various principle-based CHERI capability architectures (Morello and RISC-V variants, and even the CHERI-x86 preliminary user instruction-set architecture specification [94]), strongly suggest that our CHERI capability-based hardware would be a very strong natural foundation on which to develop MLS systems and networks, from bottom to top. (This supposition is also amplified after considering the advantages and limitations of some of the earlier capability systems in general.) This belief in the value of a highly principled capability-based hardware/software such as CHERI is explored in the following subsections of this section, discussing some advantages and some missing pieces that would still need to be developed.

1.9.1 Tagged Capabilities

For illustrative purposes, we consider several alternatives that are explicitly relevant to adapting the existing CHERI architecture to MLS requirements. This set of possibilities also provides some potential measure of backward compatibility with the existing CHERI capability representations, adapted for the potential MLS entities.

Note that the existing CHERI capability single-bit integrity tag bit (see the Version 9 specification document, Section 2.2.) ensures hardware capability unforgeability: it protects against all accidental or intentionally misguided capability modifications, which result in zeroing the integrity tag bit and thus disabling the capability (including in the following cases for handling MLS):

1. To the existing one-bit integrity tag, add a two-bit MLS level (e.g., 00 for Unclassified up to 11 for TopSecret) to represent the four possible security levels (with a relatively small but observable performance hit), again with the capability integrity rule that any attempt to modify a capability must reset its integrity tag bit to zero, disabling the capability. (Resetting tag and MLS bits to zero also could prevent using the altered capability as a signaling covert channel, although that might seem to be overkill.)

2. With item 1, rely on the strength of the CHERI compartmentalization mechanism (with additional formal proofs of its trustworthiness) to manage MLS-associated categories. (We already have future plans to formally verify the security of the compartmentalization mechanism. Without such formal analysis, MLS using this item would be unwise.)

3. Instead of item 2, alternatively extend the existing CHERI object-capability mechanism to recognize MLS objects as a full-fledged strongly typed capability object class (as was done in the PSOS design).

4. Perhaps better yet, items 2 and 3 could be done in combination, as a presumed redundancy mechanism, or if the formal analysis shows some unsuspected vulnerabilities with either item. This is is an example of composabilty vulnernability that can arise as an emergent property discussed in my 2004 DARPA project on trustworthy composition [57].

5. For completeness, another possibility would be to add many more bits to the first item to represent a large number of orthogonal categories associated with explicitly Secret and Top Secret). This is most likely very impractical, and therefore a potentially bad idea. However, it would make the capability self-defining with respect to the category. (This item is mentioned here for completeness, but also to discourage it – unless it can be made effective.)

6. However, perhaps trustworthy MLS could be achieved with no changes whatsoever to the existing CHERI capabilities, relying on the CHERI hardware spec with software MLS kernels based on the existing CHERI fine-grained least-privilege access controls and CHERI compartmentalization, if that could be enough for certain application environments (but with stronger requirements to compensate for things like speculative execution, microcontroller DMA, and unwanted channels). This alternative is perhaps fatuous and unrealistic, but it at least deserves to be considered.

Each of these alternatives has its own issues with backward compatibility relating to the existing hardware capability mechanism.//

Hardware attacks such as tampering and Rowhammer (and other noise injections such as in Dan Boneh's breaking an RSA implementation [8], being able to do a mathematical difference between the correct RSA encoding and the faulted one to reduce cryptology in finding the key to a linear search) would otherwise have required techniques beyond the realm of formal analysis. It must be assumed for the moment that the hardware is physically isolated from such attacks, as the simplest way to combat these hardware attacks.

1.9.2 Capability Revocation

Addressing temporal memory safety includes the ability to delete information but also to prevent all forms of use-after-free, and require total revocation of access. Numerous approaches to capability revocation have been tried in past capability architectures, with varying success. These earlier architectures are mostly quite different from CHERI, and each has its own quirks relating to capability revocation.[20]

[20]For example, PSOS stored the master capability for segments and MLS objects in file directories similar to Multics.

One of the earliest structural approaches for revocation of capabilities was documented in detail in David Redell's doctoral thesis [71]). His scheme involved an indirection through a master capability whose removal would automatically and instantly invalidate all subtended capabilities derived from the master. (There's an old adage – many problems can be solved by adding a level of indirection.)

Revocation in any CHERI-based MLS implementation may be nontrivial, because it is somewhat tricky already for CHERI today (e.g., [99]). However, although the Redell scheme does have performance implications, it might just work only for capabilities that have a natural probability of needing fast revocation – for example, single-shot input-output as in allegedly USB-C charger-only sticks or other capabilities that have an intentionally short life, as opposed to capabilities for certain dedicated absolute addresses for long-term DMA that are not likely to ever be need revocation. The existing CHERI scheme (for lazy garbage collection) could still work as well for the rest of the less easily revocable capabilities, depending on which of the tagging schemes in the previous section are adopted.

1.9.3 Capability Types, Sealed Caps, and Sentries

Capabilities are used in CHERI for stack pointers and function pointers, with LLVM compiler options for converting some or all C/C++ pointers into capability calls. They also help protect against return-oriented programming and jump-oriented programming (ROP and JOP, respectively) [94], Section 2.4.2.

Section 2.4.3 of that document introduces other types for definable object classes. Presumably, one class could be defined for MLS segments and another for complex MLS objects,[21] although the definition would have to be very comprehensive and formally verified for its correctness (but not completeness).

Section 2.4.4 of the CHERI ISA document suggests using a type explicitly for protecting absolute (non-virtual, physical) addresses in memory, which is particularly useful for dedicated input-output memory locations – and tied to controlling DMA from embedded microcontrollers. MLS input-output is likely to be a rather complex problem, as it will escalate the DMA problems along with the exclusive use of certain dedicated absolute addresses, and have to be thoroughly integrated with attempts at MLS networking. All of that will need thorough formal analysis.

CHERI'S linear physical capabilities could be of considerable help here (guaranteed to be unique references to physical memory), as they might support MLS enclaves. However, this could incur potential problems with page tables, as it seems likely that the memory in question would not show up in page tables if the user memory access is not virtual. (Again, see Section 2.4.4 of the Version 9 CHERI ISA spec document for further background that will be highly relevant to MLS I/O.)

Much more needed here (e.g., on sentries), particularly as it would emerge as peculiar to MLS.

[21] PSOS also had user-defined types for object classes, one of which was explicitly intended for MLS segments and complex objects.

1.9.4 CHERI Isolation, Compartmentalization, and Sharing

Section 2.4.5 of the CHERI Spec Document [94] considers isolation applied to communication and structural compartmentalization, respectively:

- "Efficient controlled communication can persist across domain transitions through the appropriate delegation of capabilities to shared memory, as well as the delegation of sealed capabilities allowing selected domain switching."
- "Software compartmentalization is one of the few known techniques able to mitigate future unknown classes of software vulnerability and exploitation, as its protective properties do not depend on the specific vulnerability or exploit class being used by an attacker." A brief history of the evolution of CHERI compartmentalization since 2010 is included in the appendix, Chapter B.

1.9.5 CHERI Compartmentalization's Role in MLS

To a first approximation, it would appear that CHERI's baseline (single-level) trustworthy strong compartmentalization (formal proofs are currently anticipated) should play a major role in MLS enclaves and multilevel separation of security levels and MLS compartments. The same is true of CHERI's trustworthy internal communications, although considerably more attention to covert and side channels will be required in both cases.

1.9.6 Stack Discipline

Single-level stacks require no cross-connections at that structural layer. Absence of buffer overflows can be guaranteed by provable hardware/software and side-channel evaluation, which would be a major step forward there.

1.9.7 Handling Interrupts and Hardware Exceptions

Safe interrupt and exception handling would be fundamental, as would atomic transactions to prevent intermediate states and incomplete transactions, as well as race-conditions. Integrity is needed despite any nested/recursive calls.

Trampolines (sometimes referred to as indirect jump vectors) are memory locations holding addresses pointing to interrupt service routines, I/O routines, etc. Execution jumps into the trampoline and then immediately jumps out, or bounces, hence the term 'trampoline'. CHERI trampolines would have to respect the MLS properties, in particular, overall monotonicity across jumps and returns – despite whatever privileges the jump acquires.

1.9.8 MLS File Systems and Operating Systems

One of the rudiments of an MLS operating system would be an MLS File System. This presents an opportunity to create a clean-slate file system, transcending all the past short-

comings and taking the best of all with respect to security, reliability, recovery from losses and file corruptions, and most important, adhering to the MLS requirements. Using tagged capabilities with embedded level/compartment information to access each file, perhaps with dynamic allocation of file names (like Multics). This could look externally like one virtual file system per level/collateral compartment, irrespective of how it was implemented, with various alternatives. Other options need to be explored as well.[22]

1.9.9 MLS System Input-Output and Networks

Due largely to Ken Thompson's arrival at Bell Labs around 1967, the Multics input-output system eventually evolved along the model of the Multics file system [13], with dynamic linking of symbolic stream names mimicking the symbolic file names. For CHERI-based MLS, a trustworthy MLS linker would be needed, with clear labeling of each I/O entity. MLS input would need to have a defined level at entry. MLS output would always be at the level at which it was created or combined with other-level information permitted only according to the MLS information-flow rules. Multilevel documents as required with clear markings of every paragraph of course would have to be treated as contaminated at the highest taint level.

Although many MLS systems would have no MLS networks and would be treated as system-high single-level networks, the appeal of truly MLS systems in certain highly constrained environments might necessitate the use of truly multilevel networks. However, that will remain a pipe dream until we have truly MLS systems that are sufficiently trustworthy for their maximally evaluated environments.. Nevertheless, pure MLS networks might have considerable interest for intercommunications. Today's hodgepodge of incompatible classified telephone networks is a real mess, and could be effectively replaced by integrated trustworthy networks for carefully constrained multilevel communications. At the moment, most of this wishlist remains a pipedream.

The simplest operating system might be a stripped-down real-time system (with or without a file system and minimal networking). See Appendix Section A.4.

1.9.10 Recovering Properly from Hardware Failures

This may be tricky regarding any residues from partially complete speculative executions, but in any event needs to return to a stable recent hardware-software system state.

Jim Gray's Tandem nonstop system [26] was one use of redundancy for reliability and survivability? SRI's SIFT (Software-Implemented Fault Tolerance) real-time avionics hardware-software system with seven-fold redundancy and two-out-of-three voting on all tasks was an extreme example of self-diagnosing self-reconfiguring fault-tolerant system [51] and its formal analysis [53, 9]. (This was the project that investigated Byzantine Agreement [40, 41] and led to the original theorem that 3k+1 clocks were required to manage up to k clocks

[22] I notice Red Hat has an option for an MLS file system; it would be worth exploring. PGN

that were arbitrarily untrustworthy, and proofs – the first proof was incorrect, but the later Lincoln-Rushby prooof was correct.

Recovering MLS networks after hardware/software outages must also be considered, although encrypting classified traffic will be essential to minimizing leakage.

1.9.11 Complex MLS Objects and Transactions

The MLS capability types for MLS segments and MLS complex objects, respectively, should provide a provably sound basis for use in MLS systems with transactional integrity.

The early computer-science literature is filled with old but relevant possible building blocks: Jim Gray's work on the Tandem Non-Stop system [87], Les Lamport's theories of multiplexing, Dijkstra's THE system hierarchical approach to locking mechanisms to avoid deadlocks [15], co-operating process [14] would all be relevant. Parnas's approach for completely encapsulated abstractions, strong typing, atomic transactions that can withstand interrupts, transactional undo, compartmentalization, and the SeaView approach to MLS database management system without requiring an MLS operating system or an MLS DBMS noted in Section 1.5.

1.9.12 Speculative Execution

The surge of speculative execution began with Meltdown [43] and Spectre [38], both in 2018. It has blossomed extensively since then. The background section of our paper on speculative execution hardware-software specification contracts, Architectural Contracts for Safe Speculation, Franz Fuchs and Jonathan Woodruff et al., was presented at the 41st IEEE International Conference on Computer Design (ICCD), 6-8 November 2023, Washington DC, USA. That paper enumerates several classes and examples of each from the literature. It develops contracts that establish criteria that must be satisfied by safe speculation for the variety of speculative-execution classes:

> We propose architectural contracts that specify the allowable limits of speculative execution to enable both software safety guarantees and hardware verification. Transient-execution attacks have presented a major threat in recent years, driving deployment of software mitigations and research into hardware solutions. Recent work on hardware/software contracts for secure speculation recognizes the need for cooperation between hardware guarantees and software analysis, and demonstrates that speculative execution models can enable formal analysis of programs with respect to transient-execution vulnerabilities. Therefore, we have extended these limited models into comprehensive architecture-level contracts that can be verified at a microarchitecture level. We define a set of speculation contracts for translation (TSC) and branching (BSC), and for memory ordering (MOSC). We also develop a set of directed-random test routines that reproduce all known contract violations in a prototype out-of-order processor, most of which represent known transient-execution vulnerabilities. We

> also extend the RiscyOO processor to enforce each contract and evaluate performance, demonstrating the practicality of the chosen contracts with an overhead between -1.2% and +1.8% for this prototype. These general-purpose contracts set the stage for specification of speculative execution for complete instruction-set architectures, and particularly for new security-focused ISA extensions.

Together with formal analysis, these architectural contracts go a long way toward identifying leakage channels in speculative execution in any MLS system or network. (A further paper is currently in submission to ASPLOS 2025.)

1.9.13 Covert Information and Integrity Channels

We make a distinction here between information channels (e.g., *covert leakage channels and side channels* on one hand, and *covert integrity channels* on the other hand – the latter of which involve surreptitiously altering system behavior by means other than conventional hardware instructions or operating system commands. Examples of covert integrity channels include Rowhammer attacks [36] on physical memory, DMA attacks causing state changes from microcontrollers or external sources such as USB-C sticks [48], , and analog circuitry inserted into processors [100]). Integrity channels are considered further in Section 1.8. If not specified, the term *channel* generally refers here to information leakage channels, while *integrity channels* are always denoted as such. Note that DMA misuse and speculative execution may be exploited as either information channels or integrity channels.

As noted in Section 1.5.1, formal analysis was used to detect covert channels and side channels in supposedly MLS kernels (e.g., in the KSOS MLS kernel). That approach needs to be dramatically extended for any future MLS hardware and software kernel designs and their implementations.

See also Proctor and Neumann, Architectural Implications of Covert Channels [70].

More publications on detecting and preventing DMA attacks (Markettos et al.) are in development, considerably beyond what is in Section 3.11.4 of the Version 9 CHERI ISA document. The basic approach involves a trustworthy capability-based interposer between the more-or-less invisible assorted microcontrollers and the main memory, providing necessary strict controls on the DMA.

Detecting and possibly recovering from Rowhammer attacks and other noise injection is likely to remain out of scope for the time being – but nevertheless highly relevant unless the physical facilities can be adequately controlled.

1.9.14 Structural Uses of Cryptography

The use of cryptography to achieve process and MLS level system and network separation/isolation was considered occasionally in the 1970s and 1980s (and explored seriously for a while in PSOS, but discarded at that time – the cryptography was not ready then, and the would-be solutions seemed like overkill in the clean-slate hierarchical architecture). However, we will reconsider it here as many things have changed technologically. Cryptography

is likely to be useful in hardware fixed-time protection of keys and data protection, but perhaps less effective in ensuring MLS *per se* in hardware and software. Nevertheless, it is worth investigating here.

Hardware security modules (HSMs) are hardened tamper-resistant hardware devices that secure cryptographic processes by generating, protecting, and managing keys used for encrypting and decrypting data and creating digital signatures and certificates. An approach called Cryptographic Communities of Interest created MLS networks in which each separate level or compartment hand its own cryptographic keys. This could be relevant for computing environments as well as MLS networks, with keys associated with different security levels and compartments.

Cryptographic Communities of Interest are typically defined by certificates sharing a common trust relationship, perhaps associated with a particular set of keys. In one particular implementation, a part of this relationship is the ability for the public and private keys within that group to negotiate unique session keys. Various other approaches are also possible, such as allocating fixed keys to different security levels and compartments, e.g., per user when not shared, or per object when shared. This needs to be explored in considerable detail, and evaluated carefully.

> THVV wondered whether there would be some way to support MLS with crypto. SECRET things would be encrypted with a SECRET key that only authorized people could have. SECRET NOFORN would be double-encrypted, have to have both. Could do the high-water mark rule: a process's output is encrypted by all its active keys.
>
> One problem with this is the process machinery, the ring 0 stack and so on. It would be a challenge to divide processes into data that the supervisor could understand and operate on, and encrypted data that the supervisor can move around but not decrypt. Probably some process machinery would have to be split into two parts: the MLS data and the process management data; and then there would be side channel issues.[23]
>
> Another problem with this crypto scheme is that encryption decays over time. That is, data that was encrypted 10 years ago might be breakable now, or in 10 more years. I discussed this with Whit Diffie years ago: how about a different kind of encryption that gets stronger with age: harder to break as a function of time? The longer a message has been encrypted, the more work it takes to decrypt. We would have to re-encrypt messages every so often or they would just appear to be random bits.
>
> Whit didn't see a way to do it, and he is way smarter than me – but he didn't immediately see a way to show it was impossible.

[23] Note that a lot of work is going into *Confidential Computing* that enables processors to operate on data that's encrypted in storage (both primary and secondary) and on network, and only decrypted in an enclave. This feature is used for attestation (I can tell where data came from) and confidentiality. Not sure about the assurance of the mechanisms or how it would be modified to accommodate MLS. Probably very complicated, but worth exploring. Steve Lipner

> I could see a lot of uses for encryption like this. Secrets of the Nixon administration. Treasure hunts. Might help with side channels.
>
> Maybe adiabatic computing can show if this kind of crypto is impossible. I could ask Charlie Bennett. End-THVV

PGN: This will be a very interesting exploration, and my guess is it could be effective. It is clearly worth revisiting.

1.9.15 Backup and System Recovery

Protecting backup files and system images of sensitive MLS information and code strongly suggests using encryption with limited admin access to cryptographic keys. Formal analysis could determine whether successful recovery cannot be compromised (integrity and security of the process, resistance to denials of service, etc.)[24]

1.9.16 Other Formal Analysis Approaches

The KSOS Feiertag flow analyzer noted above as the front-end for the Boyer-Moore theorem prover was a very primitive approach at the time. It could be dramatically improved today. However, the KSOS analysis to determine whether the kernel had any adverse information flows is still a relevant concept applied to MLS HW/SW specifications. Theorem provers. model checking, and SMF solvers modulo theories, and proof checkers would all be in scope.

The CHERI proofs that the specifications we delivered to ARM for the construction of several hundred experimental Morello boards is a fine example of the ability to prove properties about hardware specifications. At a deeper level, the University of Texas at Austin developments of the ACL theorem prover (an outgrowth of the Boyer-Moore prover) has been used to model Intel and other x86 hardware. It has recently been able to boot Linux within the prover.

1.10 Preliminary MLS Architectural Thoughts

Intuitively, it appears that the sexisting CHERI hardware-software architecture, capability-based least-privilege memory protection, compartmentalization, segmentation, paging, and stack discipline can all contribute significantly to the isolation and security required for the hardware underlying MLS systems and subsystems. Fundamentally, CHERI's principled system architecture, formal specifications, and formal analysis all add to the necessary high assurance. However, there are many potential issues to be resolved, carefully documented, and evaluated. The devil is not just in the details, it is also in the soundness and completeness of the documentation and evaluation of those details. Thus, the notion of ubiquitous

[24] Steve Lipner's comment about Confidential Computing is also applicable here.

MLS systems and networks remains unlikely – although under certain constrained environments, it could be realistic and useful. Much detailed work would remain to analyze such implementations.

This report focuses primarily on what a strong sense of past history plus understanding of CHERI's capability-based architecture can contribute to high-assurance MLS systems and networks. We believe that conventional hardware could partially satisfy some but nowhere near as many of the stated requirements. We also believe that software MLS kernels of the past are all vulnerable to many types of hardware attacks, although they be adequate when completely within trustworthy enclaves. However, the broad coverage seemingly provided by CHERI-based hardware seems superior to other conceivable solutions.

[Robert, I don't want this report to be totally CHERI centric, especially if there are ideas from other developments that might be worth considering. For that reason, Section 1.14 mentions Trusted Xenix in partitioning admin privileges (for example). Is there other seminal work in compartmentalization worth mentioning here? What else? NRL's work on MLS? Draper? Greg Morrisett's Safe Dialect of C [31] or his PittSFIeld Software Fault Isolation [49] and Josh Kroll and Drew Dean's BakerSFIeld during one summer at SRI [39]? I just noticed the Asbestos OS that Robert Morris and Frans Kaashoek were involved in at MIT in 2005 [17]. SFI may be of less interest when compared with CHERI, but still may have some merit. PGN]

1.11 Total-System Trustworthiness for MLS

The notion of total-system trustworthiness [59] is in some sense an outgrowth of the PSOS Hierarchical Design Methodology (HDM), where the intent was to be able to specify hardware and software, from the most rudimentary hardware instructions, upward to the operating system and software building blocks and potentially even higher in certain application uses. Given a set of trustworthiness requirements for a total system such as an avionics control system or a military mission, that goal would need to be extended to all of the emergent properties of the entire system complex, networks, and any thing else on which the desired properties might depend. This could make particular sense with real-time systems with requirements for guaranteed performance. That approach may seem unrealistic to most people – even if they are well-versed in formal specification and formal analysis. However it is an open-ended goal that could certainly find at least a vast array of vulnerabilities in the lower ends, and even inconsistencies or conflicts in the mission requirements and subtended system requirements. Thus, even partial analyses could be very valuable, and the effort would be ongoing and open-ended to the extent that the payoff is deemed feasible and effective. If nothing else, it would find flaws or weaknesses in the design and potentially in the implementation from the bottom up. It would also provide a forcing function for total-system mission requirements and specifications.

CHERI's ability to analyze control flow (see Section 2.3.10 of the CHERI spec Version 9) would be particularly important in analyzing MLS control flow and information flow, to

ensure there are no violations of the basic MLS requirements.

1.12 MLS Design, Implementation, and Methodology

Formally based methodologies seem to have significant advantages when it comes to high-assurance systems. This seems even more relevant in the presence of formal requirements, formal specifications of hardware/software/security models, formal analyses (e.g., proving that requirements are demonstrably self-consistent and sound, proving that the specifications satisfy the requirements, and better yet, proving that implementations are consistent with the specifications).

The CHERI development has been steeped in formal methods – from models of security, specifications of all hardware, and formal analysis of the CHERI Morello specifications – including as a basis for something like formally based performance analysis. (See Chapter 12.7 of the CHERI ISA Version 9 spec document.) That kind of highly principled R&D would have to be pervasively continued in any MLS design and implementation.

1.12.1 Formal MLS Models

Formal representations of the Bell and LaPadula formulation of MLS and the Goguen-Meseguer non-interference properties are thus highly desirable. Goguen-Meseguer *non-interference* is even stronger than Bell and LaPadula with respect to covert integrity channels, and dictates that any operations at a particular security level must not be contaminated with or in any way depend on higher-level information (or perhaps less trusted functionality(?) depending on the formulation – picking up a piece of the Biba model). Using both B&LP and G-M in the presence of particularly critical security requirements might be desirable.

However, formal methods still have some risks, such as incomplete or incorrect formulations of requirements and functional specifications, flaws and errors in the analytic tools, and misinterpretation of the results of the analyses. Also, the expertise required to use them wisely can be daunting. In general, they should not be viewed as a panacea.

The KSOS experience is a good example of determining whether a security kernel is consistent with a model such as Bell and LaPadula or Goguen-Meseguer. The KSOS analysis also detected some covert storage channels and side channels. Such an analysis could in principle be applied to object code and possibly to MLS hardware flow analysis. Proof checkers can also add assurance that the proofs are sound. However, some of the other issues (such microcontroller DMA, speculative execution, and analog Trojan horses) may require more refined techniques.

1.12.2 Identification and Analysis of Undesirable Channels

Formal methods can be used effectively to smoke out covert storage channels and side channels, but may be less well suited to identifying timing channels. Here are three items relating to covert-channel analysis in Multics (courtesy of THVV):

- Keith Loepere, MTB-696 Covert Channel Analysis, 7 Dec 1984: https://multicians.org/mtbs/mtb696.html

- Keith Loepere, Resolving covert channels within a B2 class secure system (i.e., unverified MLS), ACM SIGOPS Operating Systems Review, Volume 19 Issue 3, July 1985 https://portal.acm.org/citation.cfm?doid=850776.850778

  > For a secure computer system in the B2, B3 and A1 classes (as defined by the DoD Trusted Computer System Evaluation Criteria), the problem of confining a process such that it may not transmit information in violation of the *-property is an analyzable and solvable problem. This paper examines the problem of covert channels and attempts to analyze and resolve them relative to satisfying the B2 security requirements. A novel solution developed for the Multics computer system for a class of covert channels is presented.

- Keith Loepere, The Covert Channel Limiter Revisited, ACM SIGOPS Operating Systems Review, Volume 23 Issue 2, April 1989 https://doi.org/10.1145/858344.858347

  > In a previous article, [Keith] introduced the idea of a mechanism (the covert channel limiter) that would watch for the potential uses of covert channels and affect the responsible process (or process group) only when such potential uses exceeded the allowable bandwidth for covert channels. Recent work involving the design of the Opus operating system (target class B3) has refined and extended this idea. This paper extends the informal basis for the covert channel limiter and extends its possible utility.

Warning: Loepere's MTB analysis report said that a channel with capacity less than 1 bit/sec could be ignored. This sounds arbitrary, and perhaps wrong-headed: situations clearly exist in which one-bit per second could be devastating, and indeed one bit *only once* seems sufficient in some terminating cases such as triggering an array of cell-phone explosions.[25]

1.13 The User Perspective

This section picks up some thoughts relevant to the use of MLS environments.

Security is usually somewhat of an inconvenience, unless it can be made largely invisible – e.g., biometric access controls (when they work seamlessly, single sign-on (which generally has too many associated risks), self-defining objects, intuitively helpful user interfaces, ...

Multilevel security is inherently more inconvenient than single-level security, unless it can be made largely invisible – e.g., multilevel secure network communications that transmit

[25] One of the Steves noted, "If my memory is correct, this was the Orange Book threshold. I believe that more powerful computers bring greater covert-channel bandwidths.

and receive only at the level at which you are currently logged in, at the same time multiplexing with other users at the same and other levels. Similarly, file security can exhibit the same useful behavior. MLS tends to become a burden if you are constantly switching levels, unless you are already logged in separately at each working level but working only at one level at a time – and if you are used to doing that regularly.[26]

The properties of no-read up to a higher level and no write-down to a lower level are intuitively clear. However, there seem to be cases in which MLS is inconvenient – unless the user is used to it. Nevertheless, one exception involves handling authorized downgrading requires special permissions to violate MLS, applicable only under appropriate circumstances.

1.14 The Admin Perspective

With respect to administrators having too many privileges, Trusted Xenix [22] is particularly worth mentioning here. It separated all-powerful admininstrator privileges into separate categories to address the excessive privileges typically accorded system and network admins, which reduced the number of insider-misuse problems. Trusted It is particularly relevant to MLS systems and networks. Xenix is a harbinger of the added complexity involved in the management and operation of MLS systems and networks, a topic that Jose Bricio-Neto (Joe Bricio) inherited in an extensive DARPA's Advanced Capability Org (ACO): how to organize and simplify management of MLS – even if there were no reasonable MLS implementations to which the lessons of that project could be to applied.[27] The simple solution is that all of the system and network admins would be cleared to system-high. However, the lessons of defending against insider misuse suggest that is not a realistic solution, with far too many opportunities for rogue behavior; consequently, the techniques of Trusted Xenix seem particularly applicable to MLS.

Tom Van Vleck reflected on the Multics experience.

Can we break the idea of *Admin* down into functional atoms, and describe the logical operations they must perform? Different sites may then aggregate the atoms into roles.

Anticipating the Xenix strategy noted above, Multics sites implemented several roles, such as:

```
- SysAdmin
- Registration and accounting officer
- Hardware field engineer
- Site Security Officer
```

[26] VAX SVS anticipated a VM-per-access class (level-category combination that was needed on the system), with users able to switch between active sessions on their VMs with a small number of keystrokes. We dogfooded SVS in our development environment for about a year and a half (I used it some) and the experience seemed acceptable. The big UX problem arose in cases where users were constantly required to downgrade information and confirm their actions as a side effect of the conservatism of the Bell-LaPadula model. You mention that in your last paragraph. SBL

[27] This IRAD report is something I had hoped Jose Bricio-Neto was interested in but was unable to support.

```
- Network and communications admin
- System Operator, and maybe *assistant* and *junior* variants.
```

(Note: The practice of *clear all admins to system_high* may be a way to (over-)simplify certain functions, but is not required by the architecture.)

A small single-level secure site might have one person that did all the roles. The definition and responsibilities of each role are derived from the site mission statement and requirements. This might be broken down into:

```
Model     Who does what, and what the process is
Policy     Constraints and goals to make the model work
Procedure  Step-by-step execution of the policies
Tool     Automation of particular steps
```

So, the model for a site at the Pentagon would acknowledge data classification and personnel authorization, and require that the SSO approve certain kinds of steps.

It would be most excellent to be able to use CHERI compartmentalization to assure that Policies are enforced; the Multics ”rings of protection” facility was like a heavy and unwieldy sledgehammer.

Tom Van Vleck also noted a relevant publication, Leslie Gotch and Shawn Rovansek, Implementation and Usage of Mandatory Access Controls in an Operational Environment, *Proceedings of the 13th National Security Systems Conference* describes actually using MLS Multics (AIM) on The Pentagon's Dockmaster configuration. [We will explore that paper further here. PGN]

The entire proceedings that year have several relevant items (including one by PGN) https://csrc.nist.gov/files/pubs/conference/1990/10/01/proceedings-13th-national-computer-security-confer/final/docs/1990-13th-ncsc-proceedings-vol-2.pdf, as do the other years as well) have numerous papers related to MLS. In addition, the list of authors/referees/etc. of this series of conferences is an illustrative catalog of the security community at the time.//

Marv Schaefer commented on his recollections in this context.

> My memory of Multics hardware models has now blurred completely. In 1972 when I was briefly employed by CII (Compagnie Internationale pour lInformatique) in les-Clayes-sous-Bois, CII had two processors I was aware of, both serviced and maintained by Honeywell staff. At that time, CII was part of Honeywell-BULL, though it was still manufacturing its own line of computers, especially the MITRA-15. I did not regularly communicate with the Honeywell employees, but I did run a few programs in both implementations of Multics during that year from a model-33 teletype. The Honeywell folks were somewhat surprised that a U.S. [expert/expat?] knew about or had used Multics, since it was their impression that PL/I users in the US only touched it on S/360s. The Multics users I knew best were researchers from the neighboring IRIA centers in Louveciennes and Grenoble.

> When I later returned to SDC, I noted that the only Honeywell computers we had could not run Multics!

Tom Van Vleck responded:

> I guess you were using the Paris 645. See https://multicians.org/bellec.html and https://multicians.org/drv-bull.html
>
> If you look on https://multicians.org/site-timeline.html⟨https://multicians.org/drv-bull.html, you will see it is the 7th Multics installed. (It also says in https://multicians.org/features.html#tag22 that one CPU destined for Paris was dropped in shipment at Logan airport and had to be replaced.

Marv responded:

> At least one of them could have been the 645. I remember there having been two separate installations at CII les-Clayes-sous-Bois, in different buildings. The Honeywell reps were all Americans (based on their accents) but both machines were installed and isolated in separate machine rooms. Half a century of my erstwhile sharp memory has dulled terribly, and most of my period at CII was pen-and-pencil rather than online. I had done some compiler analysis on their IRIS80 computers, only to discover that their *optimizing* FORTRAN-H compiler from a vendor was not legitimate.

1.15 Potential MLS Subsystems and Systems

Appendix A gives some snapshopts of a variety of possible MLS systems, subsystems, and network components that could be considered for design, specifying, implementing, formally modeling and analyzing.

It would seem to be premature to flesh out each of these possible approaches before the national security community and the development community have agreed that significant commitment needs has both (1) a sufficient need for realistic MLS, and (2) that any selected choice for future development would have sufficient feasibility of attaining the desired trustworthiness – particularly considering the fundamental doubts within the historical MLS community expressed in Appendix C and elsewhere. Whereas some of our friends in that community have also considered that there is a lack of marketplace to support the development of any MLS-based industry components, others suggest simply that adequate trustworthiness is generally impossible to achieve. As a result, i am left with a feeling that any genuine efforts at carrying out future developments and analysis are not likely in the near future. However, I also suspect that the need for a few of the choices in Appendix A may become great enough within the national security community for particular application spaces somewhen in the next 10 years that government support may outweigh the overall trustworthiness issues when constrained *within the physical constraints of those spaces.*

1.16 Feasibility Assessment and Residual Risks

compartmentalization would be of considerable help in assuring MLS memory management. Direct memory access from both embedded microcontrollers and USB ports is likely to be controllable. (A paper on that subject is still in flight.) However, residual risks are likely to remain from the only partial treatment of speculative execution. Rowhammer (as just one example of noise injection) and other physical attacks on hardware are going to need special attention unless physical access can be effectively controlled in any particular working environment.

A few special cases will need to be considered.

- Some uncovered speculative-execution cases are likely to remain, as are undiagnosed DMA accesses from undocumented microcontrollers. See the subsections of Section 11.5 of the Version 9 CHERI ISA document.
- An unusual corner case involves a potential vulnerability outlined in Section 10.2 of the Version 9 ISA spec document: creating a capability for a non-accessible virtual memory location for which an absolute address can sometimes be reached after altering the TLB (Transaction Look-aside Buffer).

See Chapter 10 (CHERI in High-Assurance Systems) in the Version 9 CHERI ISA Spec document for more relevant details, most of which are highly relevant to any application attempting to implement multilevel security separation.

1.17 Conclusions

This document attempts to bring the lessons from the previous second half-entury into realistic directions for the future, The intent is to make a real dent in addressing the deeper challenges of MLS on reflection from the past, and to present some constructive system/subsystem approaches for the future. However, recognizing that perfection is impossible, together with the fallibility of people (users, admins, and would-be attackers), potenitial implementations need to be confined to applications where adequate control of all risks will be essential, and any potential violations will be completely understood and accceptable because of the limited nature of the applications.

True multilevel security for computer systems and networks is clearly among the most complicated of all total-system trustworthiness requirements, along with human safety and guaranteed real-time behavior in life-critical systems, all of which require analysis of total-system trustworthiness. To an initial approximation, full-bodied MLS systems and networks remain a pipe dream, and cannot be used in unconstrained environments.

I have enumerated a large set of obstacles and illustrated some past problems. The need for some sort of high-assurance mandatory access controls with trustworthy compartmentalization seems fairly obvious, particularly in systems with national security sensitivity. However, it has been seriously hampered by the lack of trustworthy conventional systems that

could be used as a place from which to build trustworthy MLS. Without meaningfully trustworthy hardware, there cannot be much in the way of genuinely MLS-like systems, even if totally constrained to operate only in physically protected enclaves with no electromagnetic, acoustic, or other emanations. Several thoughts come to mind, which need to be further explored:

- Recognizing that the Orange Book requirements are inherently incomplete. Adding requirements and constraints to the existing MLS models. Creating more-realistic alternative models to MLS. Resolving confusions about security (e.g., MLS) and integrity (e.g., Biba [7], and Clark-Wilson [11]). Hybrid holistic models tailored to special-purpose systems such as those illustrated in Appendix A are also needed.

- Developing total-system/network architectures with alternative security-integrity models applicable to starkly constrained operational environments, including guards, downgraders, and other requisite functional components – perhaps based on CHERI-like hardware-software specifically to increase hardware assurance and ultimately system assurance. See my CACM Inside Risks column on the challenges of total-system trustworthiness [59].

- Significantly refining the understanding of what trustworthy hardware such as CHERI might or might not be able to provide, e.g., including with respect to memory management and compartmentalization, side channels, and physical hardware attacks such as Rowhammer.

- Reflecting on marketing considerations resulting from lack of general-purpose interest in MLS. The needs for MLS are primarily for intelligence and national security – nominally a much smaller niche marketplace. For example, what might be done with some sort of mandatory access controls for healthcare. as a completely different target application area? (See Steve Lipner's 1982 paper on potential civilian uses of MLS: Non-Discretionary Controls for Commercial Applications [42], which suggested some alternative applications.) See also Steve Lipner's contributions to this report in Appendix Section C.2.

Chapter 2

References

A wide variety of released reports and published papers on CHERI can be found on the Cambridge CHERI website:
https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/ Other urls are included in some of the cited bibliography items, as appropriate. In addition, a full set of the NCSC conference proceedings is available online, although for many years it was available only to registrants. (I have a relatively full set in my office.)

I am very grateful to members of my informal advisory group of friends and colleagues. Tom Van Vleck and Marvin Schaefer contributed from the outset. Jon Callas has a history as one of the few people left who can channel the incisiveness of Paul Karger (e.g., [33]), as well as add his own background. Also highly relevant is a review by Steve Lipner, who was close to Paul Karger [32]. Steve has added considerable information about the DEC development efforta that he lead. Steve Bellovin did a late review for me, and his comments are also included in Appendix C.

My sometime co-author Brett Gutstein read the draft as of 22 May 2024, and had some pithy comments relating to the critical importance of CHERI memory safety and compartmentalization, subjects of his 2022 Cambridge PhD thesis. I expect more feedback from him and to him on a paper he is writing. As you can see, they have all contributed to this document. Their added insights have been very important in helping illuminate many of the complexities that MLS introduces additional to the otherwise ever-present issues of engineering complex systems with stringent requirements, and thus I am delighted to include their pithy comments.

I had hopes of getting comments from Jerry Saltzer and Rich Feiertag. Jerry declined after having totally detached from that era in retirement. We unable to contact Rich, which I deeply regret. Together with Michael Schroeder, Steve Lipner, Paul Karger, Roger Schell, and Tom Van Vleck, they were all involved not only in the MLS Multics retrofit (e.g., see [34]), but also the early days of the Orange Book. Marv later became the Chief Scientist of NSA's NCSC when the Pentagon was running the Multi0cs MLS Dockmaster, following after Bob Morris. In addition, I had a cameo role toward the end of the Multics MLS retrofit process in my early years at SRI.

In addition, Robert Watson is the person who has the outstanding breadth and depth

of all of the total-system issues throughout the CheRI development since we (SRI and the University of Cambridge) started working together in 2010. Robert contributed to this IRAD study early in 2024, largely by getting me to tone down my real-world expectations for MLS, although part of his reasoning is beyond the scope of this document – in that any efforts with DARPA or NSA relating to MLS would most likely take away some of our joint efforts to fund engineering work that is still needed for bringing CHERI more extensively into the real world.

Bibliography

[1] G. Barnes, R. Grisenthwaite, R. N. M. Watson, S. W. Moore, P. Sewell, and J. Woodruff. The Arm Morello Evaluation Platform – Validating CHERI-Based Security in a High-Performance System. *IEEE Micro*, 43(3):50–57, May-June 2023.

[2] Thomas Bauereiss, Brian Campbell, Thomas Sewell, Alasdair Armstrong, Lawrence Esswood, Ian Stark, Graeme Barnes, Robert N. M. Watson, and Peter Sewell. Verified security for the Morello capability-enhanced prototype Arm architecture. Technical Report UCAM-CL-TR-959, University of Cambridge, Computer Laboratory, September 2021.

[3] Thomas Bauereiss, Brian Campbell, Thomas Sewell, Alasdair Armstrong, Lawrence Esswood, Ian Stark, Graeme Barnes, Robert N. M. Watson, and Peter Sewell. Verified Security for the Morello Capability-enhanced Prototype Arm Architecture. In *31st European Symposium on Programming (ESOP 2022)*, May 2022.

[4] D.E. Bell and L.J. La Padula. Secure computer systems : Volume I – mathematical foundations; volume II – a mathematical model; volume III – a refinement of the mathematical model. Technical Report MTR-2547 (three volumes), The Mitre Corporation, Bedford, Massachusetts, March–December 1973.

[5] D.E. Bell and L.J. La Padula. Secure computer system: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, The Mitre Corporation, Bedford, Massachusetts, March 1976.

[6] T.A. Berson and G.L. Barksdale Jr. KSOS: Development methodology for a secure operating system. In *National Computer Conference*, pages 365–371. AFIPS Conference Proceedings, 1979. Vol. 48.

[7] K.J. Biba. Integrity considerations for secure computer systems. Technical Report MTR 3153, The Mitre Corporation, Bedford, Massachusetts, June 1975. Also available from USAF Electronic Systems Division, Bedford, Massachusetts, as ESD-TR-76-372, April 1977.

[8] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. *Journal of Cryptology*, 14(2):101–119, 1997.

[9] R.W. Butler, D.L. Palumbo, and S.C. Johnson. Application of a clock synchronization validation methodology to the SIFT computer system. In *Digest of Papers, FTCS 15*, pages 194–199, Ann Arbor, Michigan, June 1985. IEEE Computer Society.

[10] David Chisnall, Brooks Davis, Khilan Gudka, David Brazdil, Alexandre Joannou, Jonathan Woodruff, A. Theodore Markettos, J. Edward Maste, Robert Norton, Stacey Son, Michael Roe, Simon W. Moore, Peter G. Neumann, Ben Laurie, and Robert N. M. Watson. CHERI-JNI: Sinking the Java Security Model into the C. In *Proceedings of the 22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2017)*, April 2017.

[11] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 Symposium on Security and Privacy*, pages 184–194, Oakland, California, April 1987. IEEE Computer Society.

[12] F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the Multics system. In *AFIPS '65 (Fall, part I): Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, part I*, pages 185–196, New York, NY, USA, 1965. ACM.

[13] R. C. Daley and P. G. Neumann. A general-purpose file system for secondary storage. In *AFIPS Conference Proceedings, Fall Joint Computer Conference*, pages 213–229. Spartan Books, November 1965.

[14] E.W. Dijkstra. Co-operating sequential processes. In *Programming Languages, F. Genuys (editor)*, pages 43–112. Academic Press, 1968.

[15] E.W. Dijkstra. The structure of the THE multiprogramming system. *Communications of the ACM*, 11(5), May 1968.

[16] A.L. Donaldson. A multi-level secure local area network. In *Proceedings of the Seventh DoD/NBS Computer Security Initiative Conference*, pages 341–350, Gaithersburg, Maryland, September 1984.

[17] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. *SIGOPS Oper. Syst. Rev.*, 39:17–30, October 2005.

[18] Richard J. Feiertag and Peter G. Neumann. The Foundations of a Provably Secure Operating System (PSOS). In *Proceedings of the National Computer Conference*, pages 329–334. AFIPS Press, 1979. http://www.csl.sri.com/users/neumann/psos.pdf.

[19] R.J. Feiertag. A model of security policy for the integrated computer network. Technical Report SYTEK-TR-81001, Sytek Inc., Mountain View, California, April 1981.

[20] R.J. Feiertag, K.N. Levitt, and L. Robinson. Proving multilevel security of a system design. In *Proceedings of the Sixth ACM Symposium on Operating System Principles*, pages 57–65, November 1977. https://dl.acm.org/doi/pdf/10.1145/1067625.806547.

[21] L.J. Fraim. SCOMP: A solution to the multilevel security problem. *Computer*, 16(7):26–34, July 1983.

[22] V. D. Gligor et al. Design and implementation of Secure Xenix[TM]. In *Proceedings of the 1986 Symposium on Security and Privacy*, Oakland, California, April 1986. IEEE Computer Society. also in *IEEE Transactions on Software Engineering*, vol. SE-13, 2, February 1987, 208–221.

[23] Joseph A. Goguen and Jose Meseguer. Security policies and security models. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20, Oakland, California, April 1982. IEEE Computer Society.

[24] Joseph A. Goguen and Jose Meseguer. Unwinding and inference control. In *Proceedings of the 1984 Symposium on Security and Privacy*, pages 75–86, Oakland, California, April 1984. IEEE Computer Society.

[25] B.D. Gold, R.R. Linde, and P.F. Cudney. KVM/370 in retrospect. In *Proceedings of the 1984 Symposium on Security and Privacy*, pages 13–23, Oakland, California, April 1984. IEEE Computer Society.

[26] J. Gray. Why do computers stop, and what can be done about it? Technical report, TR85.7, Tandem Computers, Inc., Cupertino, California, 1985.

[27] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, pages 653–669. USENIX Association, November 2016.

[28] Khilan Gudka, Robert N.M. Watson, Jonathan Anderson, David Chisnall, Brooks Davis, Ben Laurie, Ilias Marinos, Steven J. Murdoch, Peter G. Neumann, and Alex Richardson. Clean application compartmentalization with SOAAP (extended version). Technical Report UCAM-CL-TR-873, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, December 2015.

[29] Brett Gutstein. *Memory Safety with CHERI Capabilities: Security Analysis, Language Interpreters, and Heap Temporal Safety*. PhD thesis, University of Cambridge UK, 2022. https://www.repository.cam.ac.uk/handle/1810/342801.

[30] Gernot Heiser and Kevin Elphinstone. L4 microkernels: The lessons from 20 years of research and deployment. *ACM Transactions on Computer Systems*, 34(1):1:29, April 2016.

[31] Trevor Jim, J. Greg Morrisett, Dan Grossman, Michael W. Hicks, James Cheney, and Yanling Wang. Cyclone: A safe dialect of C. In *ATEC '02: Proceedings of the USENIX Annual Technical Conference*, pages 275–288, Berkeley, CA, USA, 2002. USENIX Association.

[32] P.A. Karger. Limiting the damage potential of discretionary Trojan horses. In *Proceedings of the 1987 Symposium on Security and Privacy*, pages 32–37, Oakland, California, April 1987. IEEE Computer Society.

[33] P.A. Karger. *Improving Security and Performance for Capability Systems*. PhD thesis, Computer Laboratory, University of Cambridge, Cambridge, England, October 1988. Technical Report No. 149.

[34] P.A. Karger and R.R. Schell. Multics security evaluation: Vulnerability analysis. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC), Classic Papers section*, Las Vegas, Nevada, December 2002. Originally available as U.S. Air Force report ESD-TR-74-193, Vol. II, Hanscomb Air Force Base, Massachusetts.

[35] P.A. Karger and R.R. Schell. Thirty years later: Lessons from the Multics security evaluation. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC), Classic Papers section*, Las Vegas, Nevada, December 2002. http://www.acsac.org/ .

[36] Yoongu Kim, R. Daly, J. Kim, C. Fallin, Ji Hye Lee, Donghyuk Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 361–372, June 2014.

[37] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: formal verification of an operating-system kernel. *Communications of the ACM*, 53:107–115, June 2009.

[38] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *ArXiv e-prints*, January 2018. https://spectreattack.com/spectre.pdf.

[39] J. Kroll and D. Dean. Bakersfield: Bringing software fault isolation to x64. Technical report, Princeton and SRI, February 2010. http://www.cs.princeton.edu/~kroll/papers/bakersfield-sfi.pdf.

[40] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[41] P.D. Lincoln and J.M. Rushby. Formally verified algorithms for diagnosis of manifest, symmetric, link, and Byzantine faults. Technical Report SRI-CSL-95-14, Computer Science Laboratory, SRI International, Menlo Park, California, October 1995.

[42] S.B. Lipner. Non-discretionary controls for commercial applications. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 2–10. IEEE, 1982. Oakland, California, 26–28 April 1982.

[43] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *ArXiv e-prints*, January 2018.

[44] T.F. Lunt. Final report vol. 4, secure distributed data views: Identification of deficiencies and directions for future research. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.

[45] T.F. Lunt, D.E. Denning, R.R. Schell, M. Heckman, and W.R. Shockley. The SeaView security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, June 1990.

[46] T.F. Lunt and R.A. Whitehurst. Final report vol. 3a: The SeaView formal top level specifications. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.

[47] T.F. Lunt and R.A. Whitehurst. The SeaView formal top level specifications and proofs. Final report, Computer Science Laboratory, SRI International, Menlo Park, California, January/February 1989. Volumes 3A and 3B of "Secure Distributed Data Views," SRI Project 1143.

[48] A. Theodore Markettos, Colin Rothwell, Brett F. Guttstein, Allison Pearce, Peter G. Neumann, Simon W. Moore, and Robert N. M. Watson. Thunderclap: Exploiting operating-system IOMMU bypass vulnerabilities with DMA from malicious peripherals. In *Proceedings of the Network and Distributed Systems Symposium*, February 2019. https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201711-iccd2017-efficient-tags.pdf, =http://www.thunderclap.io.

[49] Stephen McCamant and Greg Morrisett. Evaluating SFI for a CISC Archiecutre. *Proceedings of the 15th USENIX Security Symposium*, 2006.

[50] E.J. McCauley and P.J. Drongowski. KSOS: The design of a secure operating system. In *National Computer Conference*, pages 345–353. AFIPS Conference Proceedings, 1979. Vol. 48.

[51] P.M. Melliar-Smith and R.L. Schwartz. Formal specification and verification of SIFT: A fault-tolerant flight control system. *IEEE Transactions on Computers*, C-31(7):616–630, July 1982.

[52] Harlan D. Mills. *Principles of Information Systems Analysis and Design*. Academic Press, New York, 1986.

[53] L. Moser, P.M. Melliar-Smith, and R. Schwartz. Design verification of SIFT. Contractor Report 4097, NASA Langley Research Center, Hampton, Virginia, September 1987.

[54] NCSC. *Department of Defense Trusted Computer System Evaluation Criteria (TCSEC)*. National Computer Security Center, December 1985. DOD-5200.28-STD, Orange Book.

[55] Peter G. Neumann. The role of motherhood in the pop art of system programming. In *Proceedings of the ACM Second Symposium on Operating Systems Principles, Princeton, New Jersey*, pages 13–18. ACM, October 1969. http://www.multicians.org/pgn-motherhood.html.

[56] Peter G. Neumann. Rainbows and Arrows: How the Security Criteria Address Computer Misuse. In *Proceedings of the Thirteenth National Computer Security Conference*, pages 414–422, Washington, D.C., 1–4 October 1990. NIST/NCSC. https://csl.sri.com/users/neumann/ncs90.txt.

[57] Peter G. Neumann. Principled Assuredly Trustworthy Composable Architectures. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, December 2004. DARPA CHATS program final report, http://www.csl.sri.com/users/neumann/chats4.pdf, .html, and .ps.

[58] Peter G. Neumann. Fundamental Trustworthiness Principles in CHERI. In A. Shrobe, D. Shrier, and A. Pentland, editors, *New Solutions for Cybersecurity*, chapter 6. MIT Press/Connection Science, 28 January 2018.

[59] Peter G. Neumann. Toward Total-System Trustworthiness. *Communications of the ACM*, 65(6):30–33, June 2022. http://www.csl.sri.com/users/neumann/cacm252.pdf.

[60] Peter G. Neumann, Robert S. Boyer, Richard J. Feiertag, Karl N. Levitt, and Larry Robinson. A Provably Secure Operating System: The system, its applications, and proofs. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, May 1980. 2nd edition, Report CSL-116; see http://www.csl.sri.com/users/neumann/psos/psos80.pdf.

[61] Peter G. Neumann and Richard J. Feiertag. PSOS revisited. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003), Classic Papers*

section, pages 208–216, Las Vegas, Nevada, December 2003. IEEE Computer Society. http://www.acsac.org/ and http://www.csl.sri.com/users/neumann/psos03.pdf.

[62] Kyndylan Nienhuis, Alexandre Joannou, Thomas Bauereiss, Anthony Fox, Michael Roe, Brian Campbell, Matthew Naylor, Robert M. Norton, Simon W. Moore, Peter G. Neumann, Ian Stark, Robert N. M. Watson, and Peter Sewell. Rigorous engineering for hardware security: Formal modelling and proof in the cheri design and implementation process. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1003–1020, 2020.

[63] E.I. Organick. *The Multics System: An Examination of Its Structure*. MIT Press, Cambridge, Massachusetts, 1972.

[64] D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), December 1972.

[65] D.L. Parnas. A technique for software module specification with examples. *Communications of the ACM*, 15(5), May 1972.

[66] D.L. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, SE-5(2):128–138, March 1979.

[67] T. Perrine, J. Codd, and B. Hardy. An overview of the Kernelized Secure Operating System (KSOS). In *Proceedings of the Seventh DoD/NBS Computer Security Initiative Conference*, pages 146–160, Gaithersburg, Maryland, September 1984.

[68] N.E. Proctor. The restricted access processor: An example of formal verification. In *Proceedings of the 1985 Symposium on Security and Privacy*, pages 49–55, Oakland, California, April 1985. IEEE Computer Society.

[69] N.E. Proctor. SeaView formal specifications. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, April 1991.

[70] N.E. Proctor and P.G. Neumann. Architectural implications of covert channels. In *Proceedings of the Fifteenth National Computer Security Conference*, pages 28–43, Baltimore, Maryland, 13–16 October 1992.

[71] D.D. Redell and R.S. Fabry. Selective revocation of capabilities. In *Proceedings of the International Workshop on Protection in Operating Systems*, pages 197–209, August 1974.

[72] L. Robinson, K.N. Levitt, P.G. Neumann, and A.R. Saxena. A formal methodology for the design of operating system software. In *R. Yeh (editors), Current Trends in Programming Methodology I, Prentice-Hall, 61–110*, 1977.

[73] Larry Robinson, Karl N. Levitt, and Brad A. Silverberg. *The HDM Handbook*. Computer Science Laboratory, SRI International, Menlo Park, California, June 1979. Three Volumes.

[74] J.M. Rushby and B. Randell. A distributed secure system. Technical Report 182, Computing Laboratory, University of Newcastle upon Tyne, May 1983.

[75] J.M. Rushby and B. Randell. A distributed secure system. *IEEE Computer*, 16(7):55–67, July 1983.

[76] J.H. Saltzer and F. Kaashoek. *Principles of Computer System Design*. Morgan Kaufmann, 2009. Chapters 1-6 only. Chapters 7-11 are online: http://ocw.mit.edu/Saltzer-Kaashoek.

[77] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975. http://www.multicians.org.

[78] O.S. Saydjari, J.M. Beckman, and J.R. Leaman. LOCKing computers securely. In *10th National Computer Security Conference, Baltimore, Maryland*, pages 129–141, 21-24 September 1987. Reprinted in Rein Turn, editor, *Advances in Computer System Security*, Vol. 3, Artech House, Dedham, Massachusetts, 1988.

[79] M. Schaefer (editor). *Multilevel Data Management Security*. National Academies Press, Air Force Studies Board, National Research Council, Washington, D.C., 1983. Report of the 1982 Summer Study, National Academy of Sciences, Air Force Studies Board, Marvin Schaefer, Chairman, For Official Use Only).

[80] M. Schaefer (editor). *Multilevel Data Management Security*. National Academies Press, Air Force Studies Board, National Research Council, Washington, D.C., 1983. Report of the 1982 Summer Study, National Academy of Sciences, Air Force Studies Board, Marvin Schaefer, Chairman; published in 1983). In particular, see Chapter 3 – General Security Policy, D.E. Denning and P.G. Neumann (eds).

[81] Roger Schell and Dorothy Denning. Integrity in trusted database systems. In *Proc. 9th National Computer Security Conference*, Baltimore, 1989. NCSC/NIST.

[82] Michael D. Schroeder. Engineering a security kernel for Multics. In *SOSP '75: Proceedings of the Fifth ACM Symposium on Operating Systems Principles*, pages 25–32, New York, NY, USA, 1975. ACM.

[83] O. Sibert, P.A. Porras, and R. Lindell. An analysis of the Intel 80x86 security architecture and implementations. *IEEE Transactions on Software Engineering*, SE-22(4), May 1996.

[84] SRI-CSL. *HDM Verification Environment Enhancements, Interim Report on Language Definition*. Computer Science Laboratory, SRI International, Menlo Park, California, 1983. SRI Project No. 5727, Contract No. MDA904-83-C-0461.

[85] S. Stahl. LSI Guard security specification. Technical Report MTR 8451, Mitre Corporation, Bedford, Massachusetts, September 1981.

[86] Gary R. Stoneburner and Dean A. Snow. The Boeing MLS LAN: headed towards an INFOSEC security solution. In *Proceedings of the Twelfth National Computer Security Conference*, pages 254–266, Baltimore, Maryland, 10–13 October 1989. NIST/NCSC.

[87] I.L. Traiger, J. Gray, C.A. Galtieri, and B.G. Lindsay. Transactions and consistency in distributed database systems. *ACM TODS*, 7(3):323–342, September 1982.

[88] Bruce J. Walker, Richard A. Kemmerer, and Gerald J. Popek. Specification and verification of the UCLA Unix security kernel. *Communications of the ACM*, 23(2):118–131, 1980.

[89] W.H. Ware. A retrospective of the criteria movement. In *Proceedings of the Eighteenth National Information Systems Security Conference*, pages 582–588, Baltimore, Maryland, 10–13 October 1995. NIST/NCSC.

[90] Robert N. M. Watson. New Approaches to Operating System Security Extensibility. Technical report, Ph.D. Thesis, University of Cambridge, Cambridge, UK, October 2010. https://www.cl.cam.ac.uk/research/security/capsicum/papers/2010usenix-security-capsicum-website.pdf.

[91] Robert N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway. Capsicum: Practical capabilities for Unix. In *Proceedings of the 19th USENIX Security Symposium*. USENIX, August 2010.

[92] Robert N. M. Watson, Graeme Barnes, Jessica Clarke, Richard Grisenthwaite, Peter Sewell, Simon W. Moore, and Jonathan Woodruff. Arm Morello Programme: Architectural security goals and known limitations. Technical Report UCAM-CL-TR-982, University of Cambridge, Computer Laboratory, July 2023.

[93] Robert N. M. Watson, David Chisnall, Jessica Clarke, Brooks Davis, Nathaniel W. Filardo, Ben Laurie, Simon W. Moore, Peter G. Neumann, Alex Richardson, Peter Sewell, Konrad Witaszczyk, and Jonathan Woodruff. CHERI: Hardware-enabled C/C++ memory protection at scale. *IEEE Security & Privacy*, 22(04):50–61, July 2024. https://doi.ieeecomputersociety.org/10.1109/MSEC.2024.3396701.

[94] Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, Graeme Barnes, David Chisnall, Jessica Clarke, Brooks Davis, Lee Eisen, Nathaniel Wesley Filardo, Richard Grisenthwaite,

Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alex Richardson, Peter Rugg, Peter Sewell, Stacey Son, and Hongyan Xia. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 9). Technical Report UCAM-CL-TR-987, University of Cambridge, Department of Computer Science and Technology, September 2023.

[95] Robert N. M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brook s Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, May 2015.

[96] R. Alan Whitehurst and T.F. Lunt. Final report vol. 3b: The SeaView formal verification: Proofs. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.

[97] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. HYDRA: the kernel of a multiprocessor operating system. *Communications of the ACM*, 17(6):337–345, 1974.

[98] W.A. Wulf, R. Levin, and S.P. Harbison. *Hydra/C.mmp: An Experimental Computer System*. McGraw-Hill, New York, 1981.

[99] Hongyan Xia, Jonathan Woodruff, Sam Ainsworth, Nathaniel W. Filardo, Michael Roe, Alexander Richardson, Peter Rugg, Peter G. Neumann, Simon W. Moore, Robert N. M. Watson, and Timothy M. Jones. Cherivoke: Characterising pointer revocation using cheri capabilities for temporal memory safety. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 545557, New York, NY, USA, 2019. Association for Computing Machinery.

[100] Kaiyuan Yang et al. A2: Analog malicious hardware. In *SSP '16: Proceedings of the 2016 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2016. IEEE Computer Society. https://www.ieee-security.org/TC/SP2016/papers/0824a018.pdf.

Appendix A

Illustrative Architectures for Exploration

A.1 Formal Analysis of CHERI-RISC-V

Robert Watson wrote early this year: One option for a hardware-software baseline would be CHERIoT, for which there is already a serious hardware formal verification effort afoot at Oxford (out of Tom Melham's group). To date, their focus has been on verifying CHERI-RISC-V extensions to the baseline Ibex core - and they've found some quite interesting bugs, which have been resolved by Microsoft. I believe they plan to turn their attention to the RISC-V ISA next (although I haven't confirmed that). The CHERIoT OS also has a very small domain-switch routine (perhaps 100-200 instructions) that should be reasonably easily subjected to formal verification. RNMW

That approach has some significant simplicities as a microkernel in which the architecture may be less amenable to adverse channels, especially in that formal analysis is already underway.

Indeed, CHERIoT would make a very nice baseline for an open-source MLS hardware-software system or subsystem, or network component, especially in that some hardware alterations to CHERI are likely to be needed. PGN

A.2 MLS CHERI-seL4 Separation Kernel

Robert Watson writes: We could look at the ongoing seL4 adaptation to Morello (Hashem Almatary, CapLtd), which is gradually growing support for added CHERI C memory protection, and being performed by CapLtd. While we don't intend to engage with formal verification at this point, the design is rigorously engineered and verification methodology is clearly accessible for it. Another limitation is that the current contract supporting that work doesn't include extending it for CHERI compartmentalization, but that would make for a great R&D project. A further limitation is that the Morello hardware, while designed to implement CHERI, is probably subject to quite a few microarchitectural side channels, given both its high-performance design and specific vintage baseline core (the Neoverse N1). Given existing use of seL4 in high-assurance environments (and presumably in network guards, etc),

the transition narrative here is quite strong. RNMWatson

Considering the years and manpower spent developing L4 and seL4 and the formal proofs, this would seem to be a huge effort. However, the availability of the fully worked development and the analytic results suggest that a new clean-slate development could be orders of magnitude less expensive and time-consuming.

PGN Note: I keynoted the very first seL4 Summit in November 2018, managed by Jason Li for DARPA. I discussed the possibility of putting the formally proven seL4 security kernel on top of CHERI hardware, to provide a much more trustworthy basis for the software kernel. Hashem Almatary is now doing that under a U.S. contract to U.of Cambridge and/or Capabilities Limited. Hesham was present at that Summit meeting, and I did get a chance to discuss the feasibility and potential benefits of an seL4-on-CHERI with him. I am very pleased that his current effort is ongoing. PGN

A.3 A Clean-Slate MLS Hypervisor Separation Kernel

CHERI hardware (with or without the capability format changes noted in

Section 1.9.1) could be an ideal hardware platform for an MLS a new separation kernel to ensure no violations of Bell and LaPadula, or some less stringent security policy and some sort of integrity policy such as the carefully designed hierarchical prevention of compromising inner layers. It would require extensive formal analysis to demonstrate that the policy was sound, and consistent with that policy correctly specified in hardware and software. This would have a significant advantage of avoiding the long history of problems associated with seL4 and the strong desire of the seL4 team not to have to redo any of the proofs (all of which evolved from the earlier L4 work, with considerable effort over numerous years). The memory management and compartmentalization would be taken care of by the CHERI hardware rather than the software hypervisor. Also, redoing the proofs would not be a from-scratch effort, as many of the required low-level kernel properties would already be covered by the existing Morello proofs. However, all of the difficulties of past efforts would have to be overcome.

A.4 MLS CheriFreeRTOS and CompartOS

Both of these real-time operating systems run on one of our Bluespec-System Verilog CHERI-RISC-V cores (although they may eventually also wind up on an emerging CHERI-RISC-V hardware base from Codasip). The CHERI-RISC-V cores have seen less validation, and there are no in-progress proof efforts, but the cores could well be more amenable to hardware modification / extension if that were required. The CheriFreeRTOS supervisor is considerably bigger – unlike the CHERIoT effort, no one has attempted to do a significant TCB reduction. So, it is a less honed platform, but a more exercised one, and easier (I suspect) to extend in both hardware and software. (Adapted from RNMW notes.) They could also be more amenable to proofs that their BSV specs satisfy the CHERI hardware ISA properties

than the more complicated CHERI-Arm-Morello spec [62].

A.5 A Highly Trustworthy MLS Telephone Network Bridge

Several incompatible secure networks are currently in use from SCIFs, particularly to communicate with different government agencies. Most of these appear currently to be incompatible with other classified telephone networks, which seriously hinders multi-agency communications. A high-level architectiral examination of tightly bound requirements for highly constrained trustworthy MLS realistic systems suggests that a trustworthy hardware-software bridge that is able to accommodate networks with comparable clearance levels could be an enormous gift to the intelligence and national-security communities. The hardware-software CHERI-Arm Morello prototype boards (with proofs that their hardware specifications provably satisfy critical security properties) could be a suitable hardware base.

A.6 Clean-Slate MLS Firewalls and MLS Local Nets

A modern MLS firewall would open up the possibility of some measure of assurability for suitably constrained efforts at MLS local networking independent of, and necessarily not connected to, the Internet. The Sytek local-area network [19] that a cryptographic option to give it some semblance of MLS, as well as the Verdix MLS local-area network [16], and the Boeing MLS Local Area Network [86]. These and others should be re-examined, and a new modern architecture considered if that were desirable.

A.7 An MLS Internet

The mulitilevel-secure network suggested at the end of Section 1.9.9 is worth pursuing along with MLS systems that can work together in highly controlled networking, with extensive use of cryptography, oversight, monitoring, misuse detection, and thorough attention to covert and side channels.

This would be a huge step forward, but suggests starting anew to avoid all the accumulated cruft of the existing Internet – especially with a clean-slate architecture that integrates interconnections among multilevel-secure systems and makes use of new protocols and sound post-quantum cryptography, overcoming existing vulnerabilities, also taking advantage of extensive formal analysis.

However, experience with the existing Internet, with rampant security flaws and ongoing exploits, suggests that this is a terrible idea. Combining Secret sites with Top Secret sites is already too risky. On the other hand, an internet of only Top Secret collateral sites might seem to be potentially reasonable – with extensive vetting of would-be users and comparable monitoring of activities. The requirements would have to be very carefully understood,

and the threat model and risks carefully assessed. Adding compartmental isolation is also probably a very bad idea.[1]

A.8 Some Guidance on Using Rust for MLS Software?

Our proposed (but never picked up) DARPA I2O CHERI-Rust seedling effort implementing a Rust compiler on CHERI for the Rust language (aided bu a formal model for Rust) us worth mentioning here, especially if we can find a funding source for our white paper outlining the effort. The main idea would be to (1) develop a model for Rust that includes all of the non-memory-safe corner cases (e.g,, library calls, machine-language, and input-output native code), and (2) then mechanically compartmentalize the pieces of the Rust language/compilation that are not memory safe. (Earlier, David Chisnal showed how that could be done for Java [10].) With a formal model of Rust, formal analysis could demonstrate the soundness of such an approach. Would Rust be any more useful for writing code for MLS applications? Probably not. But it is mentioned here as something that might help us and others to understand the implications relating to minimizing security risks in programming, compiling, dealing with multilevel-secure libraries, data integrity, and so on.

[1] My intuition tells me that a new start on a replacement Internet is a non-starter. Achieving MLS over IP using cryptographic protection and encrypting MLS gateways is the way to go. There will be residual covert channels, but that's a price to pay, and auditing or noise generation may reduce the risk. SBL

Appendix B

A Brief History of CHERI Compartmentalization

It is worth noting that CHERI's approach to compartmentalization was inspired by Capsicum [91, 90] (included in FreeBSD since its version 9x, developed by Robert Watson, Ben Laurie and Jon Anderson). It was a precursor of CHERI that already made some major conceptual leaps forward regarding compartmentalization. Capsicum extend, rather than replaced, UNIX APIs, providing new kernel primitives (sandboxed capability mode and capabilities) and a userspace sandbox API. These tools support compartmentalization of monolithic UNIX applications into logical applications, an increasingly common goal supported poorly by both discretionary and mandatory access controls.

Early work on CHERI compartmentalization began under the DARPA I2O CRASH program CTSRD project, which ran from 2010 to 2019. The history began with our Deimos work - a clean-slate (and pretty minimalist) demonstration microkernel that we built in the first couple of years of CRASH, and demo'ed on a very early CHERI ISA on the t-Pad tablet. This was entirely supported by CRASH.

Next, our 2015 IEEE Oakland paper [95] demonstrated compartmentalization running over CheriBSD, including a compartmentalized tcpdump isolating various layers of packet processing. This was supported by both CRASH and MRC.

The first real examples of the use of CHERI Compartmentalization were for mitigating the Heartbleed attack and in a paper on making Java more robust/secure [10], in which the Java Native Interface (JNI) was partitioned and protected using CHERI compartmentalization. This compartmentalization protected the Java Virtual Machine's (JVM) memory safe execution from the unsafe C code accessed via the JNI.

Under CTSRD from 2015 to 2019, we developed the CheriABI pure-capability process environment. This spatially safe ABI is the technical foundation for our library and coprocess compartmentalization efforts. The scope of software we have ported (over 35 million lines of code) provides proof of the deployability of CHERI, and provides the foundation for large-scale compartmentalization in CPM.

Compartmentalization was nominally a task in our slightly overlapping I2O SSITH program ECATS project (2017–2021), although some of the proposed work could not be accom-

plished as proposed – because of MTO funding cuts, and their management reprogramming the project to satisfy somewhat inappropriate metrics. CompartOS (a compartmented real-time system on CHERI-RISC-V) was initially developed under ECATS.

During DARPA SSITH, we also developed the co-process compartmentalization model, which accelerates inter-process IPC, with early results on FPGA in that programme suggesting an order-of-magnitude performance win. This is now running on Morello, and weve actually had a few third parties (to our surprise) using it successfully in TAP.

Under the SSITH ECATS project, we also developed a compartmentalized CheriFreeRTOS. Alongside that, supported by EPSRC and Arm, Lawrence Esswood developed CheriOS, a clean-slate CHERI-based OS. Both of these are primarily documented in the technical reports of their PhD dissertations (see
https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/), although we are urging Hesham Almatary to write a conference paper on CheriFreeRTOS.

Compartmentalization was one of the three main topics of the DARPA I2O ETC, Enabling Technologies for Compartmentalization. The work in this report on co-process compartmentalization and library compartmentalization, and their evaluation plans, which are expected to improve performance and ease of use. This effort is seem as a valuable precursor for our existing four-year CPM DEC project.

Compartmentalization is one of the topics for the DARPA I2O MTSS seedling project, Memory and Type Safety at Scale, which turned out also to be anticipating the DARPA I2O CPM DEC project, Deploying Effective Compartmentalization. The DEC project is expected to begin near the end of March 2024, and involves developing application of CHERI compartmentalization to large existing systems.

Appendix C

Broad Retrospective Position Statements

In this document-in-progress, I am seeking to find a few niches among what is generally perceived to be an impossible problem under various realistic adversarial threat models. Here are two strong position pieces that have been volunteered by long-time friends and colleagues.

C.1 Steven M. Bellovin

Although I have no stake in the issue, I think that spending too much effort on MLS is a very bad ideaI think it's a security dead end. Outside of DoD settings (and probably within it), it's too inflexible and cannot be used without trusted guards, and there the problem is more semantic than anything an MLS system will protect. What is needed to secure anything like this is better security architectures, and perhaps a different security model on the host, but one that is probably not MLS.

Consider any web site that handles sensitive data. Perhaps it's an e-commerce site with credit card numbers and purchasing habits, or a hospital's health records site, or a financial institution. All of these are really a web server (plus load balancers and TLS front ends) plus multiple databases. Some random user connects across the Internet. Now, per the TCP/IP spec (and for very good reasons, if you believe in covert channels), both ends of the connection have to be at the same level and compartments. So: random users are at UNCLASS, which means that the web server has to be running at UNCLASS. It now has to get information from an MLS database, but at what level? Obviously, UNCLASS. But that's wrong; we really need a compartment per user, since we want to protect each user's data from everyone else. Do we have a separate compartment for everyone on the planet?

It gets worse. If we're going to have that sort of compartment structure, we have to be able to authenticate everyone and verify their compartment. How can that happen? Do we assert that every device on the entire net is trustworthy? There are also implications for family computers, and while the ideal might be for every member of the family to have a separate login on the family computer (an ability that Windows has had since Windows 95), I don't think that that would be the most common type of household behavior. Ignore that,

thoughthis technical scenario would pretty much have to rely on a global PKI, with all that that implies for privacy. (Enrolling new users on a site is also a challenge.)

It also means that the web server, or at least part of it, has to be fully trusted, to authenticate the user and then fork a new instance at the proper level and compartment. Do you trust web servers that much? I sure don't, though again, the authentication piece can be small and separate, albeit with a huge impact on performance. (The load balancers/ TLS front ends pose other difficult questions that I'll leave as exercises for the reader.) Now I want to buy something from the e-commerce site. Perhaps my web server instance is the only one that can read my credit card numbers, but how does billing work? What level and compartment will handle the credit card payment? And remember that the network connection to the billing computer has to be at a single levelwhat level is that? Maybe we have a trusted component doing such things, but that goes back to the system architecture question.

Health records? What permissions are needed to populate a user's test results or appointments Note that appointment data can itself be very sensitive in some situations, e.g., an abortion clinic or an STD clinic or even something that a random public figure does not want disclosed. Some years ago, I was talking with a health record security specialists hereclinical faculty at the Columbia University Medical Center - and he said that there were (as I recall) 160 different permission bits. In fact, there were actually twice as many, since some patients have the "VIP bit" set, in which case their records are more carefully guarded. Clearly, the ability to write such things is sensitive, but we have to deal with No Write Down. We also need to label per-patient data properly, which also demands privilege and high security in all labs.

I could go on, but I'll conclude with two anecdotes. First, about 25 years ago at AT&T, I tried building an MLS-based system to control access to very sensitive internal data. I gave upit needed guards all over the place to move stuff between levels, and I had no assurance that those would be correct and secure; there were too many modules that would have to be written by too many people. As we all know, it's devilishly hard to get code correct, even security code. (Btw, have you seen https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf, the Root Cause Analysis of the CrowdStrike failure?)

The other anecdote concerns the Internet Worm of 1988, at a time when the Orange Book was current and was the best advice on how to build secure systems. After the worm hit and was analyzed, I reread the Orange Book and concluded that nothing it would have stopped the worm. I went to someone considerably more senior and asked him about it. "On, B3 would have, since it requires a thorough search for bugs." Umm, noI was too much a student of Fred Brooks and Dave Parnas to believe in such things. More generally, and I didn't fully realize this until a decade later, when I was on the NRC committee that did "Trust in Cyberspace", the whole MLS model was based on trusted kernels and untrusted user space. But the security problems we've seen most of in recent years are at the application level; they don't implicate the kernel.

So what is the answer? Viewed generically, in the metaphor I've been using for a fair number of years, our security structures are based on walls and doors. A wall might be the

kernel/user level boundary or the isolation of different processes on Unix(-like) system or VMs and the hypervisor. In the network world, the firewall is the canonical wall. We're pretty good, though by no means perfect, at building walls, and things like SPECTER and row hammer show what can happen when your system model isn't complete. But doors are the way we pass information through walls, system and hypervisor calls, network requests, protocols that can pass through firewalls, etc. These doors are the real problem, both in the policy specifications and in their implementation. (Some years ago, I asked a top security person at a large company how many authorized connections they allowed through their firewall(s). "About 500." And how many unauthorized connectionsin my terminology, unauthorized doors - do you think you have? "Probably another 500." And that's assuming that the firewall rules for the authorized pass-throughs were specified correctly and that the firewall code for them was implemented correctly – have you ever looked at what it would take to handle SIP correctly, to give just one example?

As I've implied, I think that the correct answer is a correct system architecture. It's a question I've been trying to answer for almost years; I think I have a handle on the answer now.

Steve Bellovin

C.2 Steve Lipner

I spent a lot of my career (1970-75, 1981-92, occasionally thereafter) working on MLS systems. I'm substantially in agreement with Steve Bellovin's position paper in this Appendix section.

The report includes a reference to my paper "The Birth and Death of the Orange Book" (https://www.stevelipner.org/links/resources/The%20Birth%20and%20Death%20of%20the%20Orange%20Book.pdf) that was published in the IEEE Annals of the History of Computing in 2015. The history of the Orange Book is largely a history of MLS, so I consider the paper a resource that should be required reading for people who are interested in revisiting MLS. That paper describes the origin story of the early MLS efforts by the Air Force and MITRE and the way those efforts led to the Bell-LaPadula model. It also refers to the usability problems that resulted from the constraint that the Military Message Experiment software provide a user interface compatible with the Bell-LaPadula model: a user who wants to send a lower classified reply to a higher classified message is faced with the need to jump through user interface hoops "declassify" the reply message. The need to "write down" occurs in many common real-world scenarios and even the Military Airlift Command and Air Force Data Services Center (AFDSC) applications would have faced it.

Making the user interface to an MLS system more palatable inevitably requires expanding the Trusted Computing Base (TCB) for the system - giving more code the ability to compromise classified information. The Multics system deployed at AFDSC included an extension to the Multics email utility that did just that. The extension was claimed to have a limited impact on the TCB, and while I no longer recall the details, I recall being skeptical at the time. (The AFDSC Multics system was targeted at roughly B2-level assurance where

there's no requirement to minimize the TCB, so the extension was acceptable.)

As Tom Van Vleck says, a major objective of MLS was to avoid having to provide each user with multiple computers, but the pain that resulted from the MLS solutions and the diminishing costs of personal computers and network connectivity made MLS systems unacceptable or unpalatable. Of course, commercial organizations' security policies don't work like the government's policies for handling classified information, so there's no commercial demand for MLS. The end result: the government didn't buy the MLS systems that the industry attempted to build, and no one else did either.

The MLS work was an interesting technical challenge, and we came up with creative solutions to a lot of problems, but without a market, the projects were canceled. I think that a lot of the work done during the MLS era has been lost - the DEC work on VAX SVS is documented in a handful of professional papers - from the IEEE Oakland conference in 1990 and 1991 and from IEEE Security and Privacy in 2012 [footnote to the citation below my signature] and some patents. But the detailed design documentation and code seem to be long gone. I particularly believed that our rigorous application of a layered software architecture to a system that then seemed to be of reasonable size (about 50K lines of code) was a significant accomplishment, and the published papers include little detail of what we did and how we did it. I don't know about the other MLS projects from the late 80s and early 90s, but I'd not be surprised if that documentation and source code too were long gone.

I'm happy to assist in research that illuminates what was done and to answer questions about what we did on MLS "back in the day" and I definitely advocate making public any of the work done back then that hasn't already been released (including details of VAX SVS if they could be found). But I think that real-world considerations of usability and complexity make the MLS model impractical or insufficient for almost all applications- that was probably a major reason why not enough customers wanted to buy.

See also Lessons from VAX SVS for High Assurance VM Systems, with Trent Jaeger and Mary Ellen Zurko, IEEE Security and Privacy, November-December 2012

[Steve added some additional details on 23 December 2024:]

As to the VAX SVS effort, I believe that the biggest contribution was the use of rigorous layering to design and implement the system. That work built on prior research including a MITRE system (Venus?) that I believe acknowledged a debt to Dijkstra's work in the 60s, the early PSOS design, and of course Roger Schell's work at NPS that eventually culminated in GemSOS. One can build a layered system without MLS of course. In fact, Microsoft restructured Windows as a strictly layered system around the Vista-Windows 7 timeframe. I probably pointed the Windows team in that direction, but they did it successfully on their own. As far as I know, those changes have endured and subsequent adaptations of Windows (new versions and features, Phone, other hardware platforms, other new interfaces) were acknowledged to be much easier to create and test because of the layering.

Regarding the 1982 paper [on suggested non-miliitary/intelligence uses of MLS), I believe it's still required reading for the CISSP exam, but I don't think anyone has ever done anything with it. Both that paper and the Clark and Wilson model seem to have sunk without trace, at least as far as people using the models to support an organization's information

security policy.

I imagine you know that Windows implements a variation on Biba integrity as a mechanism to protect system data. As with all other mandatory policy implementations, the need for exceptions to the model (in this case, I'm running at low integrity, but if I want to change a system setting or install a new component, I need to be at a higher integrity level) poses a usability challenge. The tradeoffs between rigorous enforcement and usability were explored and tuned over several releases, starting with Vista. Steve Lipner

C.3 Tom Van Vleck

Steve Lipner taught me that MLS was an attempt to create and use 1960s computers in a way that complied with specific United States rules about classification of information and clearance of people, in a fashion that was less expensive than one computer per compartment and less expensive than having armed security guards watching every operation.

The history I found about MLS is in section 1.4 of
https://multicians.org/b2.html which mentions the Ware Report and the Anderson Report.

In practice, trying to use MLS on a big computer utility was very expensive, frustrating, and ineffective. NSA's attempts to use MLS on Dockmaster led to site-local modifications of the rules that were probably "illegal." – e.g., the `extend_high` hack noted in Section 1.7.2.

Minor point: covert channels and the "confinement problem." As I wrote in 1990in
https://multicians.org/timing-chn.html. "every shared resource is a channel." Multics-era systems worked hard to share most resources for efficiency and cost reasons, making complete analysis very complex. The Orange Book said that at level B2 we should identify all covert channels and eliminate those with greater than 1bps bandwidth. (Orange Book page 80)

Since those days, other covert channels have been found, e.g. power drain. Can one make an MLS design that does not share any resources between compartments?

Minor point: Footnote 7. True, the Multics clock would run fine past 2000, but there were Y2K bugs date formatting etc, which were fixed by the one site that ran past 2000, in the Canada Department of National Defence.

Multics was built without the benefit of powerful modeling and logic like PSOS used. It did not have a proof of correctness. This meant it could not be an A1 system. (Although I helped implement the Multics MLS features, I never used a Multics with MLS enabled. Should interview some actual users of MLS systems, e.g. the Canada DND guys.)

(My office mate at Tandem went on to Sun, and worked on their Compartmented-Mode Workstation product.)

(At Bell Labs, Doug McIlroy did an MLS version of Unix that had multilevel windows:
https://ftp.fibranet.cat/UnixArchive/Documentation/TechReports/Bell_Labs/CSTRs/163c.pdf

[More from Van Vleck:]

On page 33, Jon Callas discusses Loepere's opinion that one bit per second is acceptable in a covert channel. A 128-bit key is thus two minutes, and as you note, there are plenty

of places where a single bit could give the game away. At the same time, though, I once helped design a secure file system and documented a lot of covert channels and the customer basically said, ”awww, covert channels, isn’t that sweet” with an affectionate noogie but I was left wondering what the response might have been if I didn’t include a covert channel discussion. I believe that Loepere’s basic point, that we can’t eliminate covert channels, only throttle them is true. If that is true, the discussion is about where to put the threshold, only.

I thought the 1bps number came from the Orange Book, section 8.0, page 80, and Keith was just repeating it.

(I think the only valid bps limit for timing channels is 0 bps. Suppose a trojan horse program signals 1 if a particular name ever spoke to a grand jury.) THVV

C.4 Marvin Schaefer

That number for covert channel bandwidth came full-born on Roger Schell’s forehead. Dan Edwards and I didn’t subscribe to it; Sheilaqu Brand was a diciple of Roger. I was happy at the time with the notion of a covert channel being able to drive a 1401 line printer at its then top speed, which we’d demonstrated. Marv

Concerning Tom, A1 only called for a formal spec (i.e., a spec written in a

`formal` language that could, eventually, be submitted by a formal verification system. A2 had not been defined at the time it was written, and as more was learned it could well have had to be moved into A2+. This was a point of debate by more than a few of us as we began to explore the different kinds of specifications and spec languages. And that, as we later came to understand, was a significant area for research into modeling of time and asynchrony became apparent [and the ensuing risks! PGN]Steve Lipner

Appendix D

Miscellaneous Slides

Eventually, the most innovative of the earlier MLS kernel efforts may have some more details. For now, here is one slide on the fundamental principles for trustworthiness, followed by two tables relevant to the PSOS design abstraction and would-be evaluation hierarchies.

Table D.1: Fundamental Principles for System Trustworthiness

Overall Architectural Principles for Trustworthiness
0. Trustworthiness must be defined with respect to its constituents,
e.g., security, integrity, reliability, human safety, etc.
1. Sound conceptual system architecture, with realistic implementability
and requirements for sufficiently high assurance
2. Minimization of what must be trusted – e.g., avoidance of dependence on
components that must be trusted despite not being trustworthy, while
being respectful of the Einstein Principle:
Everything should be made as simple as possible, but no simpler.
3. Open design (e.g., Kerchhoff for cryptography, but protecting keys)

Design and Implementation Principles (Including Security)
4. Complete mediation (including nonbypassability and unforgeability)
5. Least privilege
6. Intentional use of rights (avoiding the confused deputy problem)
7. Layered and predictably sound compositional assurance
8. Robust dependency despite potentially questionable components
9. Abstraction
10. Modularity
11. Encapsulation (e.g., information and control hiding)
12. Object and type integrity
13. Separation of privileges
14. Separation of domains
15. Separation of policy and mechanisms
16. Separation of roles
17. Separation of duties
18. Economy of mechanisms
19. Least common mechanism
20. Fail-safe defaults
21. Sound authentication
22. Sound authorization
23. Compromise recording (and analysis)

Administrative/Operational Principles for Trustworthiness
24. Administrative control and integrity
25. Comprehensive accountability

Principles for Usability and Complexity Issues
26. Psychological acceptability & ease of use (avoiding bad user interfaces)
27. Compromise work factor (especially for cryptography and passwords)

Table D.2: Simplified PSOS Abstraction Hierarchy

PSOS Abstractions	Layers
Command interpretation	16
User input-output	15
User environments	14
Procedure records	13
User processes, user I/O	12
User objects (e.g., MLS)	11
Directories	10
Abstract object manager	9
Virtual segmentation	8
Paging	7
System processes	6
Primitive input-output	5
Arithmetic, other ops	4
Clocks	3
Interrupts	2
Registers, absolute memory	1
Capability instructions	0

Table D.3: PSOS Trustworthiness Properties

Layer	Properties to be proved
17	Soundness of user type managers
15	Search path flaw avoidance
12	Process isolation, I-O soundness
11	No lost objects (w/o capabilities)
10	Strongly-typed MLS objects (OPTION 1)
9	Generic type safety
8	Correct segmentation, no residues
6	Interrupts properly masked
4	Correctness of the low-layer hardware math/FP/interrupts/etc.
0	Unforgeable, nonbypassable, nonalterable capabilities (MLS tags, OPTION 2)