CSE 598
Secure Microkernel Design

Secure Microkernel Design: Trends & Forces

Prof. Michel A. Kinsy

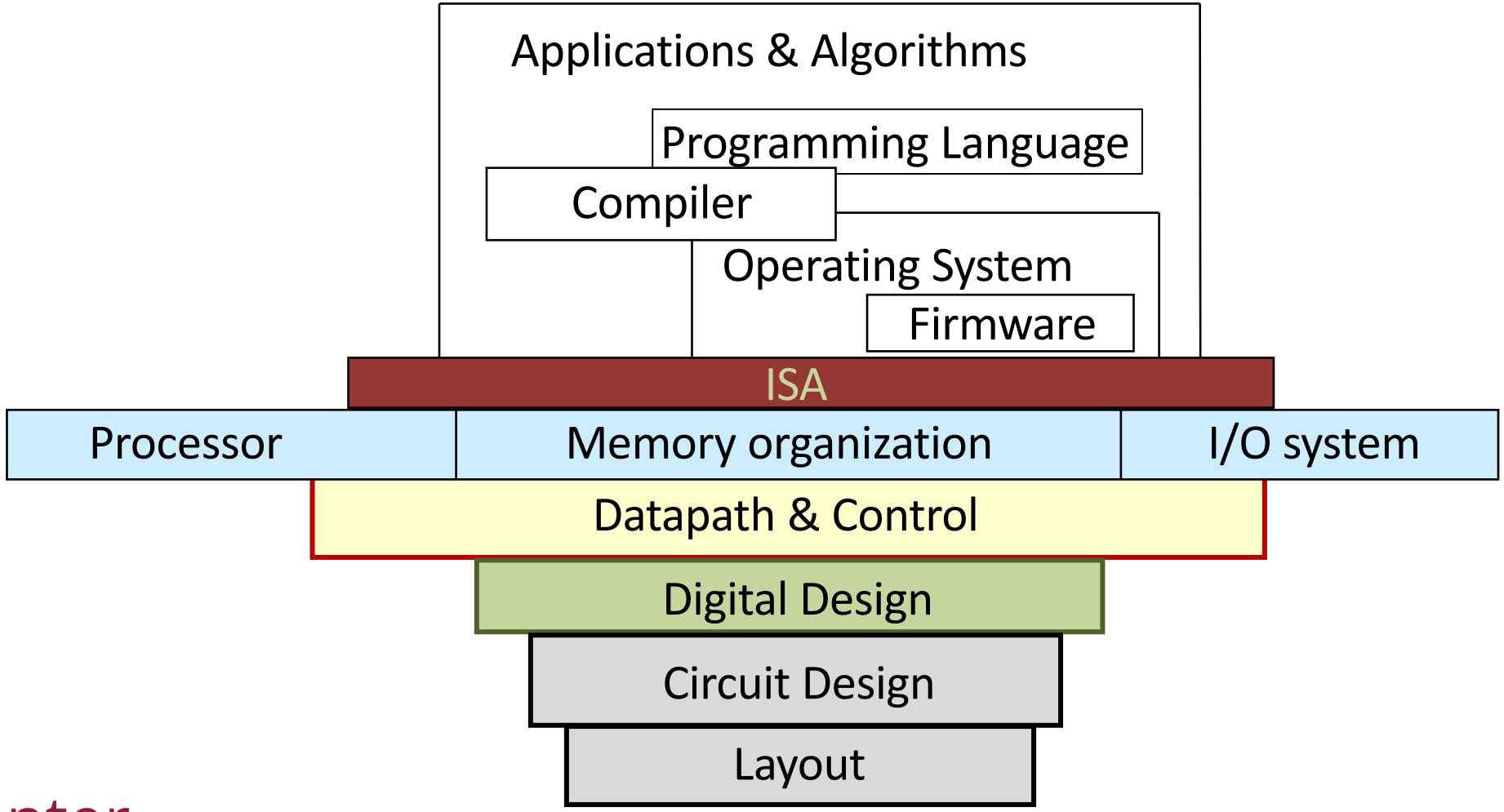
Major Topics

- Topic 1: Introduction to the fundamentals of computer architecture using the RISC-V ISA
  - CPU design (datapath and control, pipelining),
  - Memory systems including caching and virtual memory
  - Peripherals–I/O
- Topic 2: LVM Overview: a modern compiler infrastructure
  - Brief introduction to LLVM analysis and instrumentation passes
  - LLVM-based symbolic execution
  - LLVM intermediate representation (LLVM IR) programming
- Topic 3: Overview of Operating Systems and Microkernels
  - Differences between microkernel and monolithic
  - Advantages of a microkernel architecture in terms of security, safety, and reliability
- Topic 4: OS/microkernel structures and privileged operations
  - Characteristics/properties of microkernel-based operating systems
  - Exokernel, L3 microkernel, sel4 microkernel
  - Microkernel-implemented capabilities
  - Implementation of microkernel internals on the RISC-V architecture.

Major Topics

- Topic 5: Virtualization techniques
  - Memory Virtualization
  - CPU and Device virtualization
- Topic 6: Microkernel process management
  - Synchronization, communication, and scheduling
  - Lightweight message passing interface (MPI) and remote procedure calls (RPC)
  - Shared memory multiprocessor
- Topic 7: File system design fundamental concepts
  - Linked and indexed file allocation, mounting, virtual filesystem layer, memory mapping, and journaling
  - Goals of different filesystems and virtual filesystem (VFS)
  - Filesystems performance optimizations

Computer System Abstraction Layers



Traditional Computer System Performance Evaluation

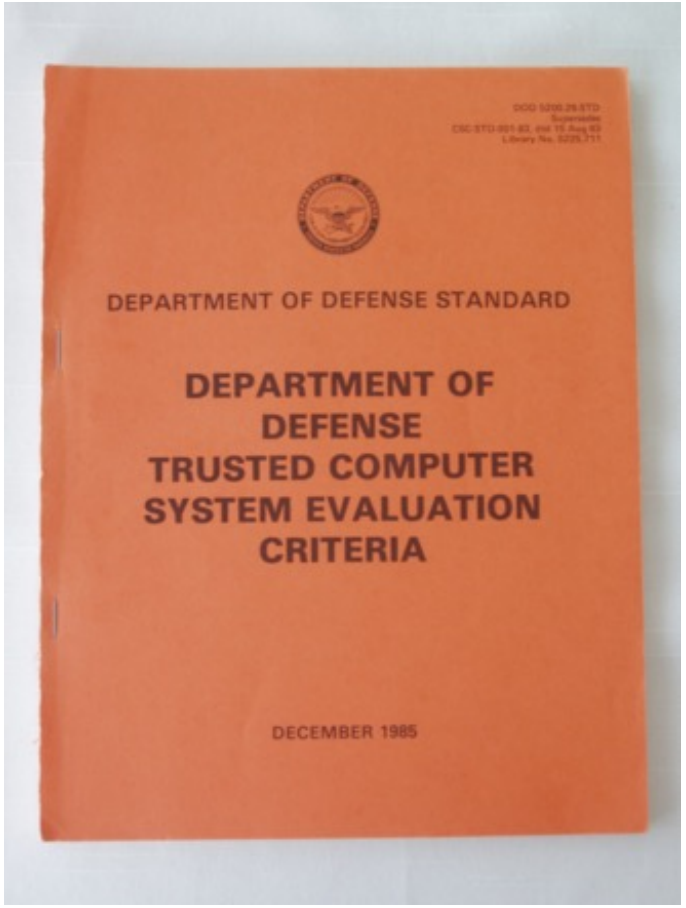
- Instructions per program depends on source code, compiler technology and ISA
- Cycles per instructions (CPI) depends upon the ISA and the microarchitecture
- Time per cycle depends upon the microarchitecture and the base technology

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- Security was overlooked!

Department of Defense Orange Book

- *"The ability of a trusted computing base to enforce correctly a unified security policy depends on the correctness of the mechanisms within the trusted computing base, the protection of those mechanisms to ensure their correctness, and the correct input of parameters related to the security policy." **



* P. S. Tasker et al., "Department of Defense Trusted Computer System Evaluation Criteria", 1985.

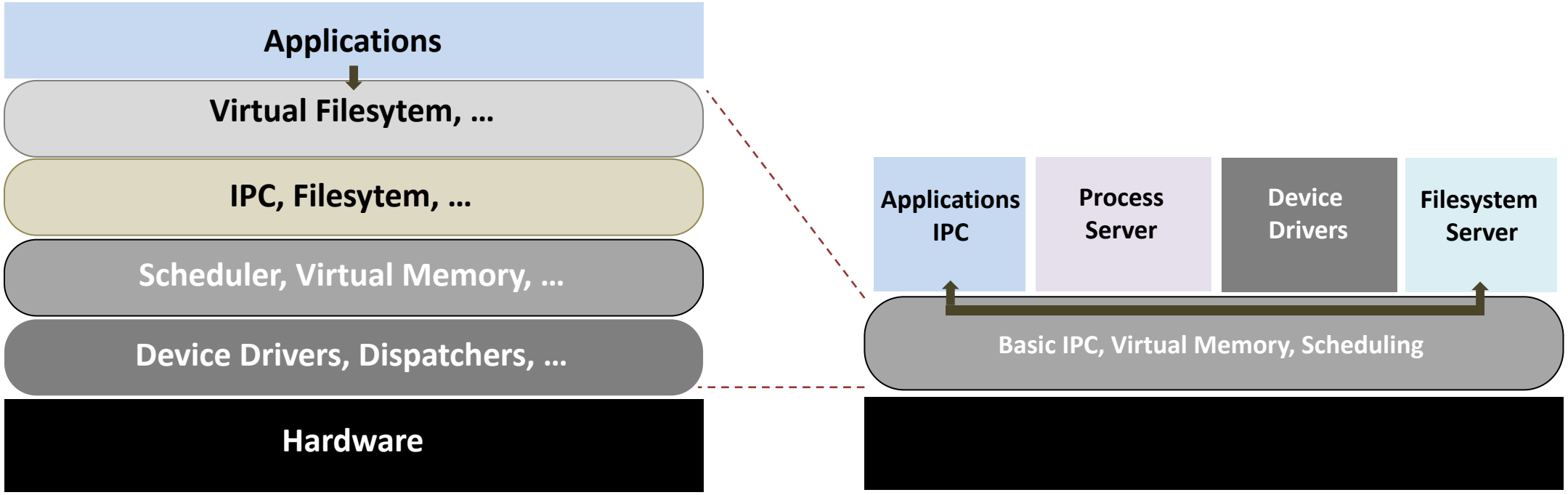
Trusted Computing Base (TCB)

- The trusted computing base = {hardware, firmware, software} that are critical to the system's security.
- Vulnerabilities in the TCB can jeopardize the security of the entire system.
- "Any code executing in privileged mode can bypass security, and it is therefore inherently part of a system's trusted computing base."*

* J. M. Rushby. 1981. Design and verification of secure systems. SIGOPS Oper. Syst. Rev. 15, 5 (December 1981), 12–21.

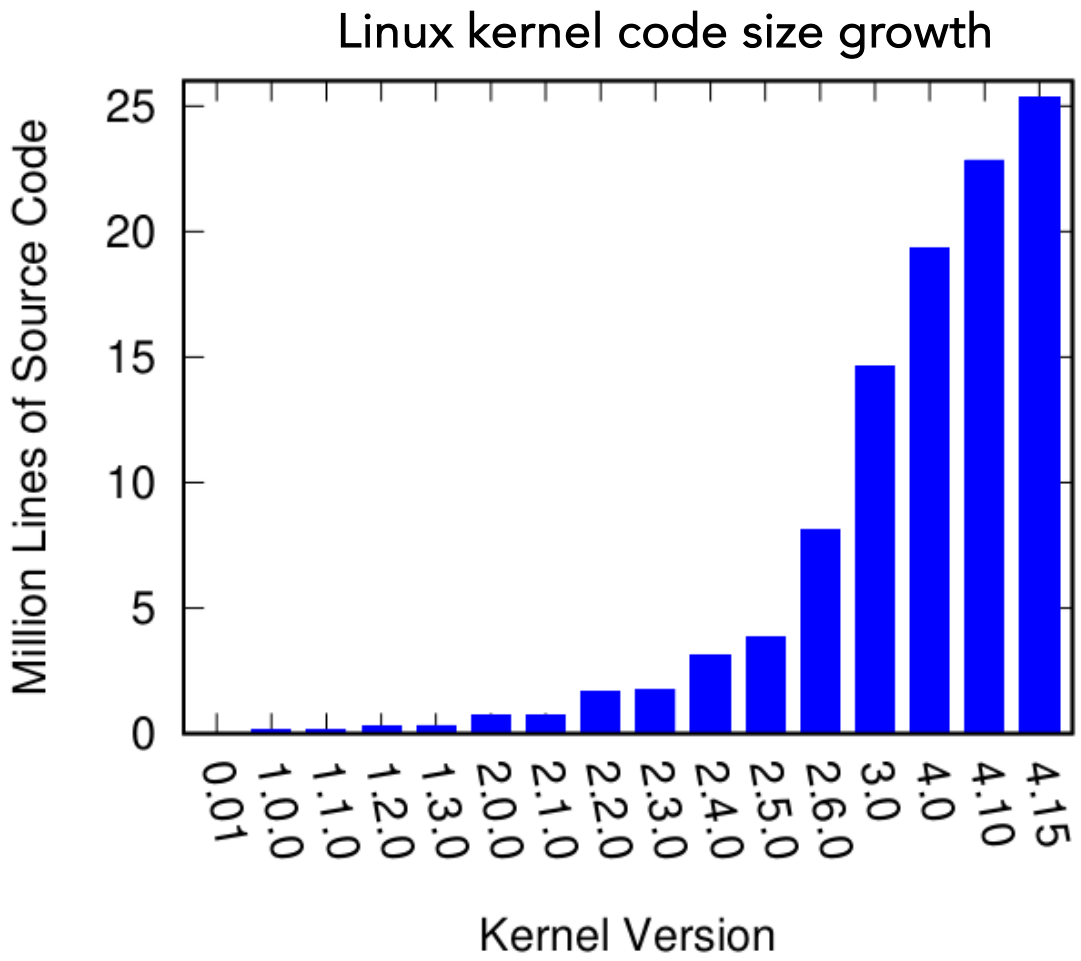
Monolithic vs. Microkernel-based OS Designs

- S. Biggs, et al., "The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-based Designs Improve Security"
  - 9th Asia-Pacific Workshop on Systems (APSys '18)



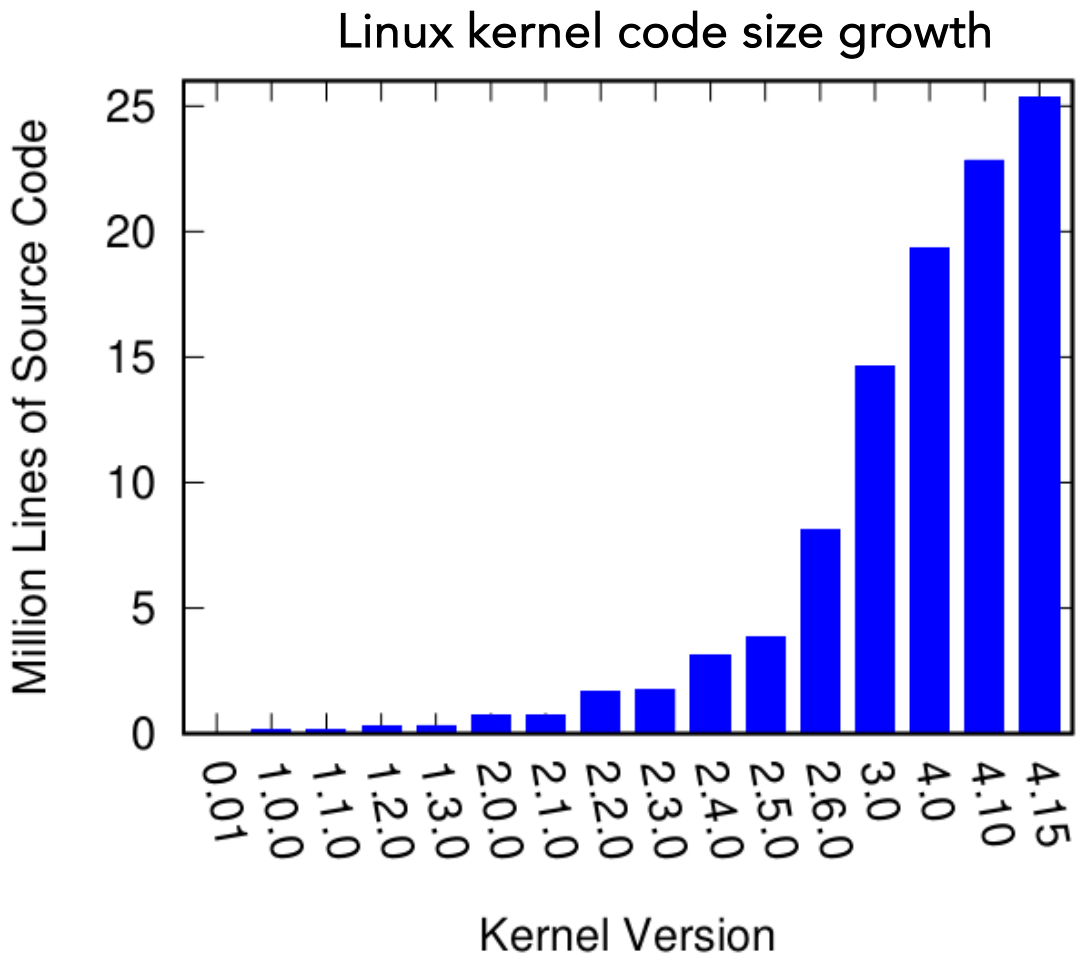
Monolithic Kernel Security Woes

- Monolithic kernels are big!
  - 26 MSLOC for Linux
  - 65 MSLOC for Windows
  - 80 MSLOC for Mac OS X 10.4 Tiger



Larger Kernel => Larger TCB => More Vulnerabilities

- "OS vulnerabilities are a key enabler of cyber crime, the cost of which is estimated to reach $6 trillion by 2021." [1]
- Average estimates of fault density in code is 0.5-3/kSLOC [2] [3]
  - 13,000 vulnerabilities in Linux
  - 32,500 vulnerabilities in Windows
  - 40,000 vulnerabilities in Mac OS X



[1] R. Mardisalu. "14 Most Alarming Cyber Security Stati
2020" https://thebestvpn.com/cyber-security-statistics-2
[2] Hatton, Les. "Reexamining the Fault Density-Compor
Connection." IEEE Softw. 14 (1997): 89-97.
[3] P. Mohagheghi, et al., "An Empirical Study of Software Reuse vs. Defect-Density and Stability". In the 26th International Conference on Software Engineering (ICSE '04), 2004.

How much could a μ-kernel fix?

- Formally verified μ-kernels are about 6-10 kSLOC
- Linux kernel Critical Vulnerabilities and Exposures (CVE) fixed in a μ-kernel design
  - 96% of Linux CVE would no longer be rated critical
  - 40% of CVEs would be completely eliminated with a verified μ-kernel
    - 29% of CVE are eliminated with an unverified μ-kernel

Assessment	Short	#	Fract.	Cumul.
Eliminated	Yes	33	29%	29%
Eliminated with verification	FV	12	11%	40%
Strongly mitigated	A	19	17%	57%
Weakly mitigated	C/I	43	38%	96%
Unaffected	No	5	4%	100%
Total:		112	100%	100%

S. Biggs, et al., "The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-based Designs Improve Security" 9th Asia-Pacific Workshop on Systems (APSys '18)

Common Vulnerability Scoring System (CVSS)

- Base metrics:
  - Exploitability: Ease of exploiting the vulnerability (remote, local, physical access, complexity, privileges required,…)
  - Scope: Is the exploit confined to a single component? Does it compromise multiple components?
  - Impact: Assess degree to which Confidentiality (C), Integrity (I), or Availability (A) is impacted.
    - Confidentiality: Is any information stolen?
    - Integrity: Is there unauthorized modification of data?
    - Availability: Is there a crash of the system or application?

Common Vulnerability Scoring System (CVSS)

- Temporal metrics:
  - Exploit maturity: How publicly available or pervasive is the exploit?
  - Remediation level: Are there any defenses available?
  - Report confidence: How verified is the exploit?
- Environmental metrics:
  - Security requirements: How important is the affected asset in terms of the CIA properties?
  - Modified base: Catch-all modifier for any special circumstances

Illustrative Examples: CVE-2015-4001

- CVE-2015-4001: Signed integer error in OZWPAN driver
  - This driver is a USB host controller device driver that communicates with a wireless peripheral over Wi-Fi.
  - Signed integer error => subtraction becoming negative => Causes memcpy operation to copy network data into a heap buffer.
  - An attacker can insert a payload into a packet to inject data into the heap.
  - Since Linux loads this driver into the kernel, it could cause DOS by crashing the kernel or execute code with kernel privileges.

Result: Eliminated by a μ-kernel

- The driver would run as a user-level server in its own address space
  - Cannot overwrite kernel memory to cause
    - A system crash
    - Information leakage
    - Data corruption
- Injected code would only have minimal privileges
- High containment: Driver only communicates with Wi-Fi user-level server to interact with the device

Illustrative Examples: CVE-2014-9803

- CVE-2014-9803: The Linux kernel on some Nexus devices mishandles execute-only pages
- Allows a malicious application to gain kernel privileges
  - Causes a total compromise of the system integrity
  - Can render the system completely unavailable
- Result: Eliminated by a μ-kernel with Formal Verification
  - The operation must be performed in kernel mode, so it could occur in a μ-kernel, but not if the μ-kernel was formally verified.

Illustrative Examples: CVE-2015-8961

- CVE-2015-8961: There is a use-after-free possibility in __ext4_journal_stop() that can result in full file-system disclosure (C) or a kernel crash (A).
- Result: Partially mitigated with a μ-kernel
  - The file system would be implemented as a user-level server => there would no longer be a kernel crash
  - Access to the file system and confidential data could still be possible, but if the data was encrypted and the file system does not have access to the key, then data could be safe from being read.
    - This still would not protect against a ransomware attack though.

Illustrative Examples: CVE-2017-0563

- CVE-2017-0563: An elevation of privilege vulnerability in the HTC touchscreen driver could enable a local malicious application to execute arbitrary code within the context of the kernel.
- It is possible to re-flash the system-on-chip firmware or bootloader, both of which were unsigned.
- Result: No protection possible from a formally verified μ-kernel
  - The attacker gains control before the formally verified kernel is even loaded.

Next Learning Module

- Topic 1: Fundamentals of Computer System Architecture