AFRL

High Assurance Containerization for CDS in the Cloud

May 10, 2024

Dr. Nathan Daughety
AFRL/RIGA
Information Directorate

Outline

- CDS Background
- vCDS
  - vCDS Architecture
  - vCDS Implementation
  - Use-cases
- Auditing tool
- Containers
- Goals
  - High Assurance Containerization and Orchestration
  - CDS Containerization

Cross Domain Solution (CDS)

- CDS: a system which supports the access to and/or transport of data between domains of differing classification levels
    - Enforces a security policy
    - Uses security isolation mechanisms:
        - Data separation
        - Authorized information flow
        - Sanitization
        - Damage limitation

[2]

CDS Use

- CDS are used in government/military, banking/finance, energy/utility, healthcare, telecommunications, transportation/aviation
  - Secure information sharing/aggregation
  - Enhanced decision making
  - Collaboration between various departments/agencies
  - Adherence to regulatory compliance for data exchanges

- Government has a particular need for CDS which other entities see as insurance problems

Policy Challenges for CDS

- Raise the Bar
  - Design and implementation standards are slow to evolve
  - Little room for product modification as needs arise (i.e. CDS products often require rebuilding from scratch – time consuming, expensive, etc.)
  - CDS testing under RTB is extensively time consuming (waiting list + length of testing)

- Common Criteria evaluation problems [11]
  - “Usability is ignored”
  - “squeeze a very volatile and competitive industry into a bureaucratic straightjacket, in order to provide purchasers with the illusion of stability”
  - Paperwork is the test subject, rather than the product
  - Security through obscurity
  - Does not guarantee security, only that claims about product were independently verified

vCDS: A Virtualized Cross Domain Solution Architecture

Meeting needs and combating challenges

Why vCDS?

The status quo in CDS technology lacks:

- **Trustworthiness**: security and functionality not mathematically proven
- **Commercial availability**: expensive, DoD controlled/owned
- **Remote Deployability**: inconsistency with paradigm shift to cloud computing
- **Versatility**: highly specialized, single use-case

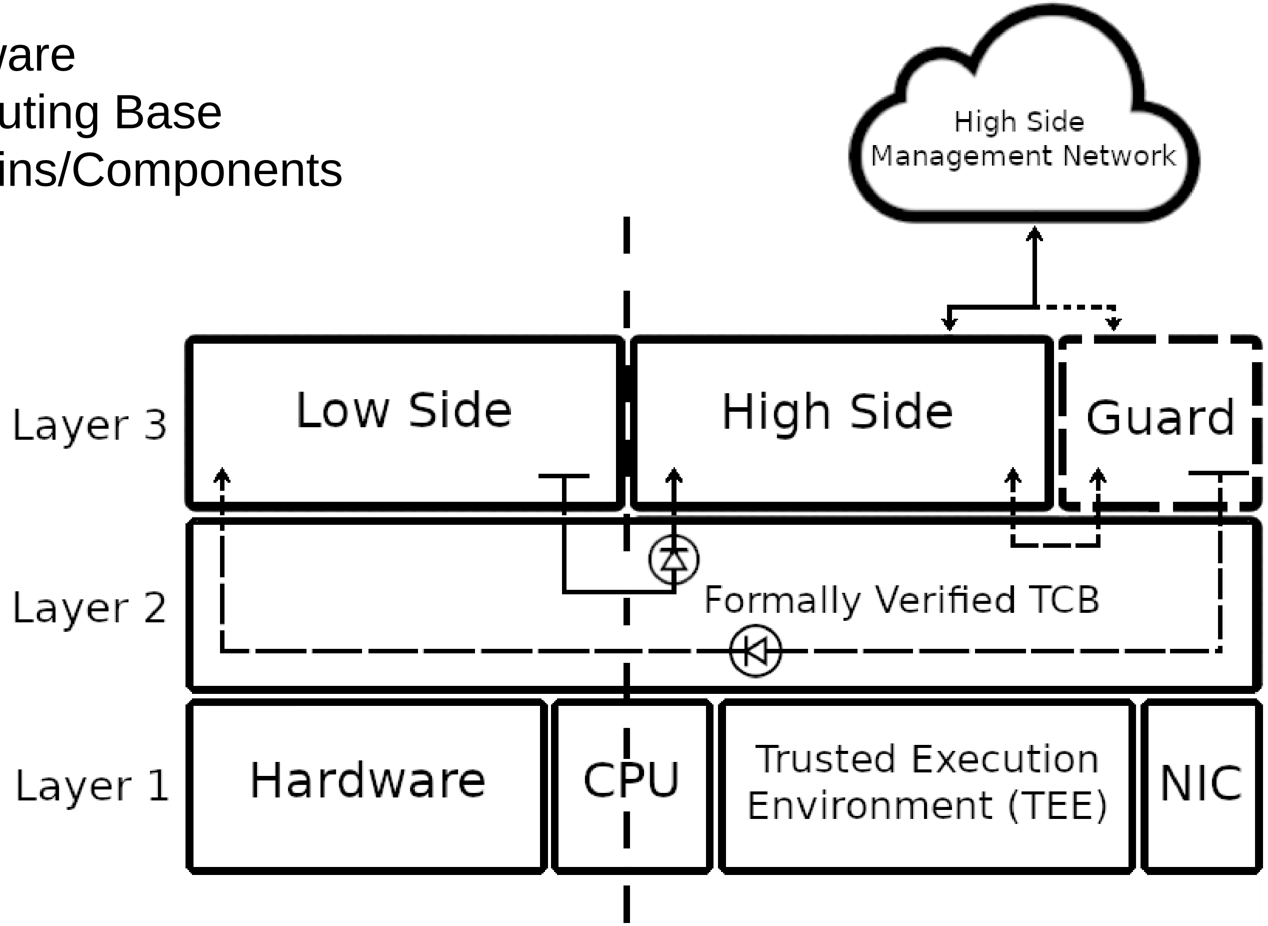
vCDS solves these limitations:

- **Trustworthy:** built upon a comprehensively formally verified TCB
- **Commercially available:** open source and commodity building blocks
- **Remotely Deployable:** TEE allows for use in offsite/cloud computing environments
- **Versatile:** adaptable to multiple use-cases and environments without significant costs or modifications

[1]

vCDS Architecture

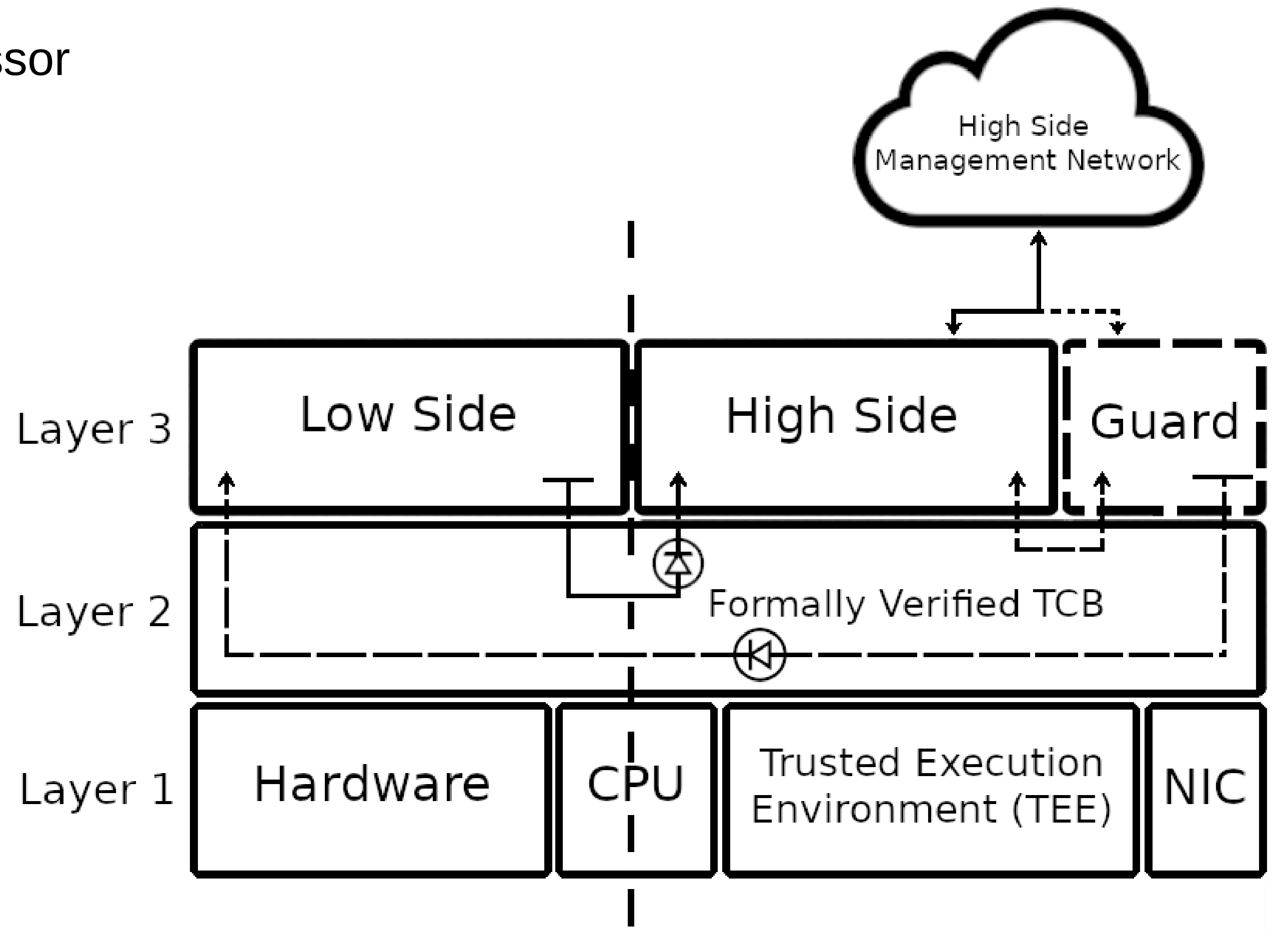
- Layer 1: Hardware
- Layer 2: Computing Base
- Layer 3: Domains/Components



[1]

vCDS Use-Cases

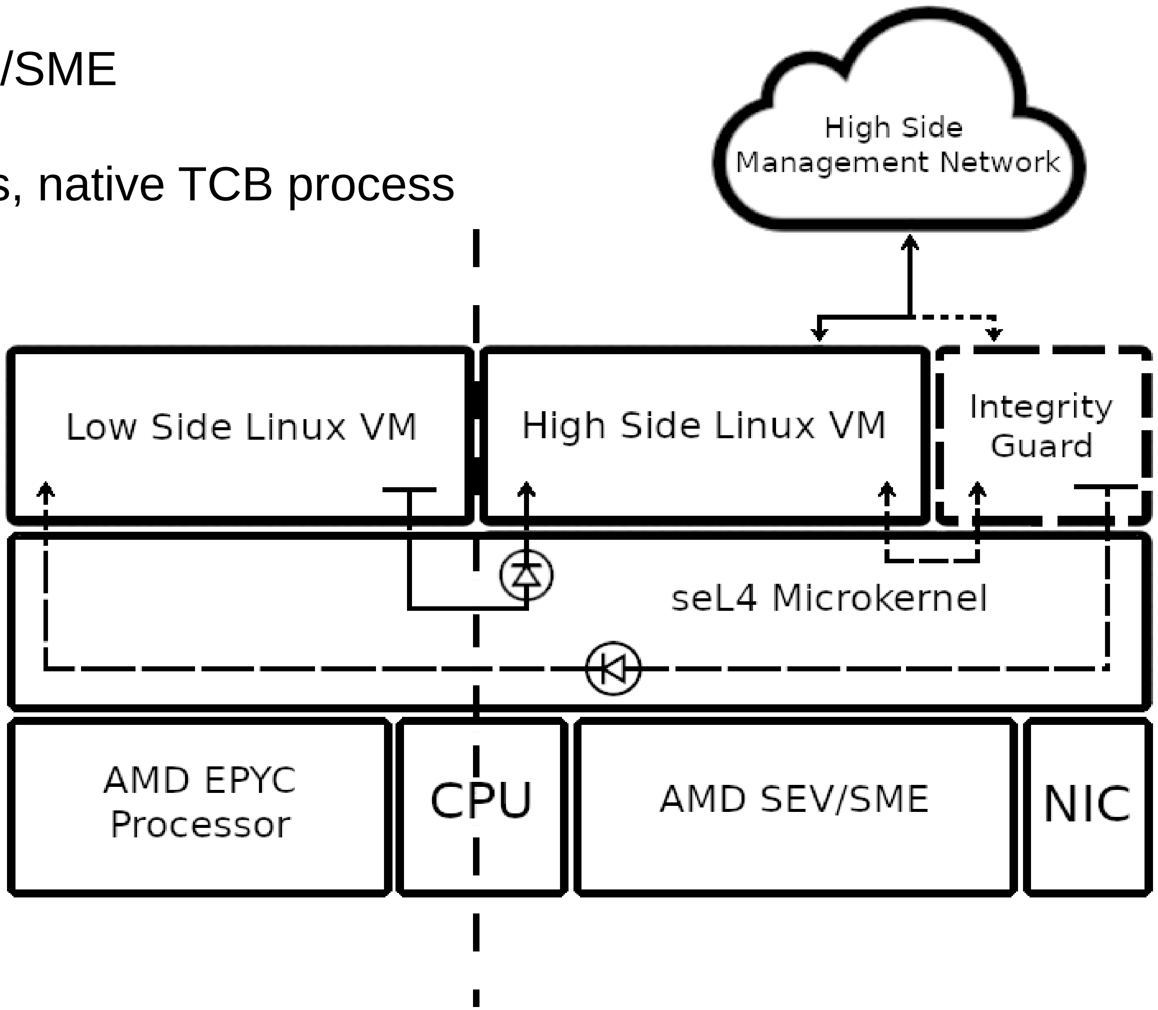
- Stream Processor
- Data Sharing
- Big Data/HPC



[1]

vCDS Implementation

- Layer 1: AMD SEV/SME
- Layer 2: seL4
- Layer 3: Linux VMs, native TCB process



[1]

Threat Model Protections

- Threat model includes all threat vectors which seek to compromise data confidentiality

Vulnerabilities	Components		
	TEE	TCB	Guard
Side Channels	*	[36, 44]	[17]
Disclosure, Spillage, Manipulation	[30, 39]	[44]	[17]
Logic Errors		[40, 44]	+
VM Breakout	[30, 35, 39]	[44]	
Control Hijacking, Injection	[30, 35]	[44]	
Communication, Spoofing	[30, 35]	[40, 44]	

* additional TEE protections against side channels
+ N-version programming to combat logic errors

Security Analysis of vCDS for Deployment in Untrusted Cloud

- Trustworthy Components
  - Formally verified for functional correctness
  - No bugs, protects data confidentiality
  - Proofs provided; available for independent verification
- Data Flow Restriction
  - Data diode ensures unidirectional data flow (when combined with trustworthy components, prevents spillage)
  - Proofs provided
- Computation Isolation
  - All computations are contained within the respective components
  - Proofs provided

[1]

Security Analysis of vCDS for Deployment in Untrusted Cloud

- Hardware Protections and Memory Encryption
  - Transparent memory encryption
  - Encryption for data: at rest, in transit, in use
  - Virtualization security -- computation isolation
- Decidable Object Security and Staticity
  - Explicit memory allocation (through capability invocation)
  - Staticity -- configurations occur *before* compile time so that all channels and privileges are pre-allocated, i.e. no channels or added privileges can exist outside of what is predefined

[1]

Security Limitation

- A system built upon a formally verified TCB does not mean it is secure out of the box.
  - Susceptible to security misconfigurations
  - Security guarantees of vCDS depend on a correct security configuration.

- Security and information flow auditing is required for a trustworthy vCDS instantiation
  - vCDS implementation must be verified against the ADL specification

vCDS Security Model

- vCDS Security Model is Decidable
  - Isolation Theorem [10] – proves that subsystems cannot exceed or leak authority over memory or communication channels to other subsystems
- Security enforcement – ADL describes explicit access controls, data flow
- Custom ADL tailoring – custom labels propagate down through the ADL (triggering appropriate protection models) allowing audit algorithm to check constraints
- ADL + customized labels allow for security control auditing

[3]

vCDS Security Model Auditing Tool

- Application of Isolation Theorem:
  - Existing authority cannot increase within a system
  - Function through which the system description may pass to determine system safety; output is any possible authority propagations (none in a correctly configured system)
- Audit Tool Phases
  - Collection
  - Audit

[3]

vCDS Limitations

- vCDS inherits limitations of building blocks
  - TCB assumes hardware behaves as expected, specification is correct, theorem prover is correct
  - Timing channels not captured by the formal specification
- VMs
  - Poor scalability
  - Heavy weight, longer startup time, not the most efficient
  - Static

[1]

Where we are going

- Started with VMs now shifting towards containerization and orchestration because of the benefits

- However, doing so brings with it several concerns

What containers are…

- Containers are portable, but restricted, computing environments packaged with the bare requirements necessary for an application and/or service to run

- Benefits:
  - Efficiency, deployment speed, agility, isolation (hmm), and management
  - Support for DevSecOps workflows and CI/CD pipelines
  - Allows for rapid capability deployment, facilitating connectivity, and adjusting to dynamic operational priorities

Threat Model

Threat Category	Threat Description	At-Risk Component(s) in Tech Stack
DDOS/DOS Threats	Overwhelming application, container, or host with traffic	Application code, middleware, container runtime, kernel, network config
Privilege Escalation and Access Control	Exploiting vulnerabilities to gain unauthorized access	Container runtime, kernel, operating system, orchestration platform
Cross-Tenant, Cross-Container, Container Escape	Breaking out of containers, accessing data or resources of other containers	Kernel, container runtime, host OS, hypervisor, orchestration platform
Data Breaches	Data exposure, inference, exfiltration, unauthorized storage access	Encryption, access controls, network security, storage management
Large Tech Footprint and Tech Stack	Complexity leading to misconfigurations, increased attack surface	Container runtime, orchestration platform, networking, storage
Orchestration-Specific Threats	Exploiting orchestration platform vulnerabilities or misconfigurations	Orchestration platform, API server, authentication mechanisms

What containers are not…

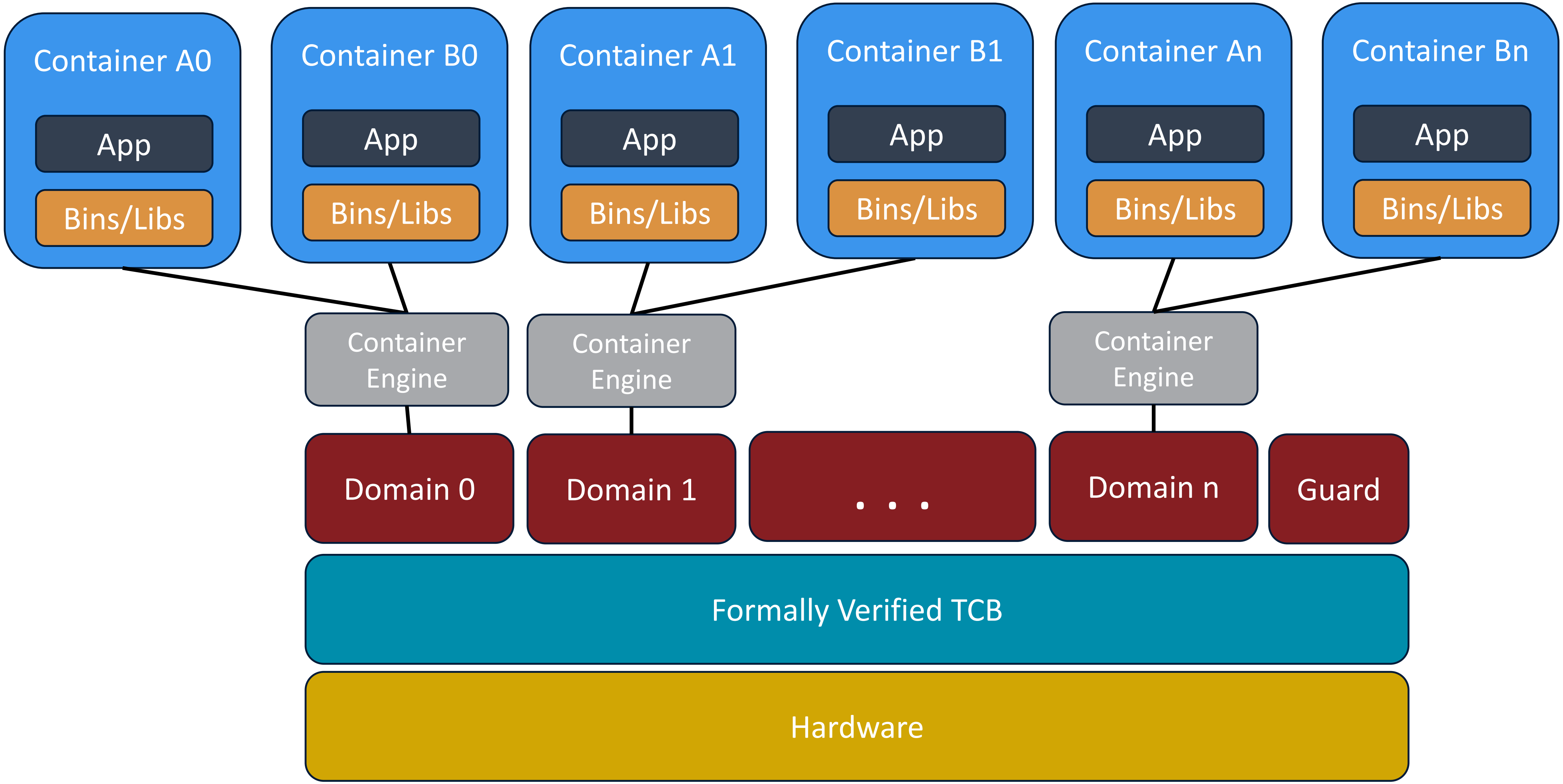
- Secure
    - Weak isolation
    - Any kernel-related vulnerabilities can break the isolation layer
    - Large attack surface between container and host

- Containers are mistaken for security boundaries [5]
- Security is the main barrier to widespread container adoption in contested operational environments [6, 7]
- Deployment technology security is the weakest link in the DevSecOps approach [8, 9].

Goals

1. High assurance/Trustworthy containerization and orchestration ecosystem
2. Support existing technology stacks, high usability and easy adoption
3. Should not rely on the trustworthiness of a container to enforce assured isolation
4. Support dynamic mission requirements (i.e. spin up/tear down capabilities at the speed of relevance)
5. Support CDS cloud applications

Remotely Deployable, High Assurance, Containerized CDS?



Challenges for CDS containerization

- Selecting the Domain Architecture:
  - Building containers directly on TCB (e.g. Library OS, Unikernel)
  - Building on top of layers (e.g. Containers on VMs/gVisor, Container-in-Container, Container on X)
  - What about drivers?
  - What about multi-container applications (microservices)?

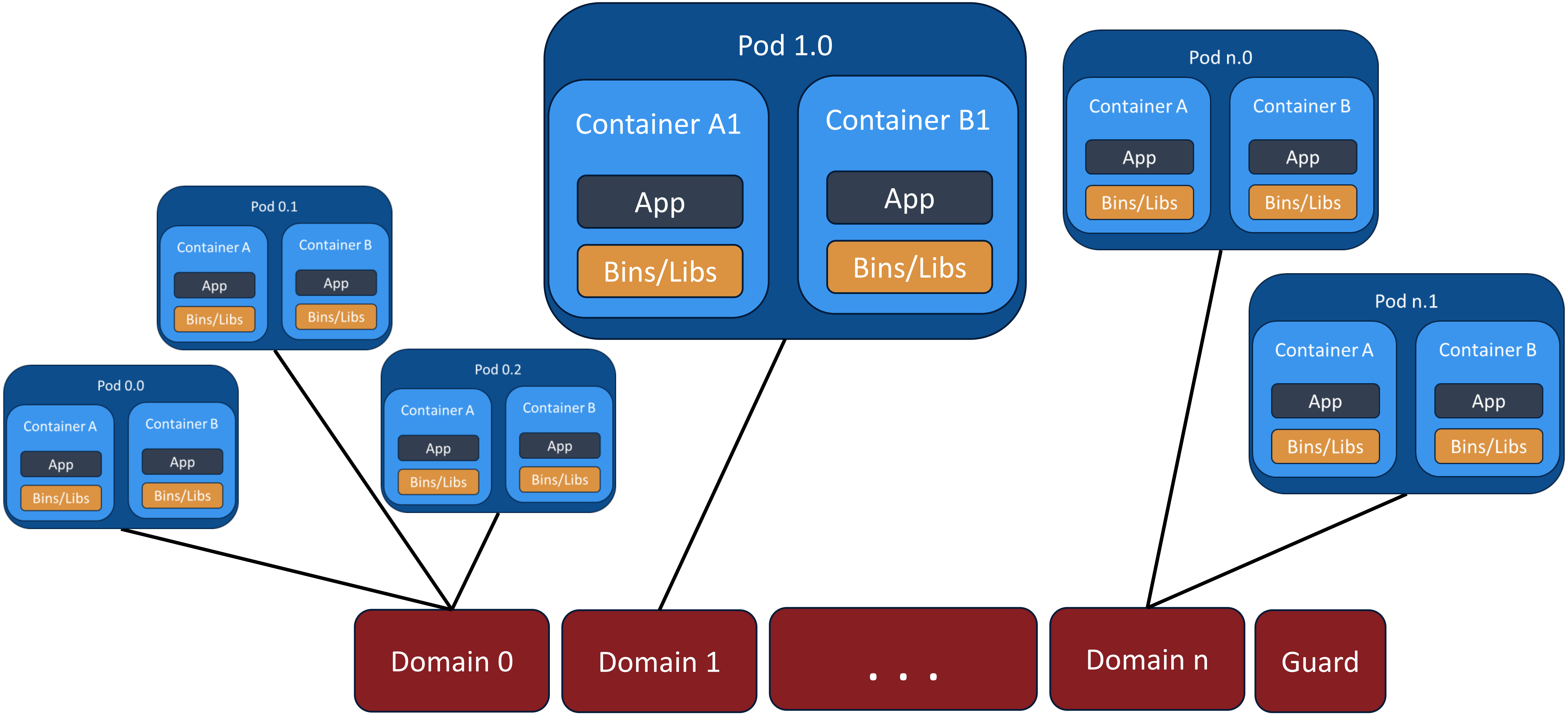
- Maintain security guarantees in a dynamic ecosystem that can be changed very quickly (i.e. new capabilities spun up, old capabilities torn down, scaling)
- Enforcing high assurance within existing technology stacks (e.g. Docker, K8s, etc.)

- NSA's RTB – design and implementation standards are slow to evolve to real-time and connectivity needs (i.e. CDS containerization is not permissible)

Goals cont.

1. High assurance/Trustworthy containerization and orchestration ecosystem
2. Support existing technology stacks, high usability and easy adoption
3. Does not rely on the trustworthiness of a container to enforce assured isolation
4. Support dynamic mission requirements (i.e. spin up/tear down capabilities at the speed of relevance)
5. Support CDS cloud applications
6. Multi-application CDS (e.g. stream processor + data sharing)
7. Adopt auditing tool for orchestration to enforce controls on all methods/means by which containers can pass information to each other

Remotely Deployable, High Assurance, CDS Orchestration?



Conclusion

- CDS are essential technologies for which the DoD has a unique need
- Containerization is becoming necessary due to mission requirements
  - But technical and policy challenges make high assurance containerization difficult
- Exploring ways to realize the described containerization and orchestration architecture that achieves remote deployability, agility, and high assurance necessary for modern CDS applications

Thank you! Questions?

Dr. Nathan Daughety
AFRL/RIGA
nathan.daughety@us.af.mil

Dr. Marcus Pendleton
AFRL/RIGA
marcus.pendleton.2@us.af.mil

Dr. Hui Lu
The University of Texas at Arlington (UTA)

Kaesi Manakkal
The University of Texas at Arlington (UTA)

References

1. N. Daughety, M. Pendleton, S. Xu, L. Njilla and J. Franco, "vCDS: A Virtualized Cross Domain Solution Architecture," *IEEE MILCOM,* 2021.
2. N. Daughety, Design and Analysis of a Trustworthy, Cross Domain Solution Architecture, University of Cincinnati, USA: Ph.D. Dissertation., 2022.
3. N. Daughety, M. Pendleton, R. Perez, S. Xu and J. Franco, "Auditing a Software-Defined Cross Domain Solution Architecture," *IEEE CSR,* 2022.
4. N. Daughety, M. Pendleton, S. Xu, L. Njilla and T. Reuther, "Cross Domain Solution Architecture". Patent P2012US/00581744, 2023.
5. Bélair et al., "Leveraging Kernel Security Mechanisms to Improve Container Security: a Survey," ARES '19, 2019.
6. S. Sultan, I. Ahmad and T. Dimitriou, "Container Security: Issues, Challenges, and The Road Ahead," *IEEE Access,* 2019.
7. A. Bettini, "Vulnerability Exploitation in Docker Container Environments," *Black Hat,* 2015.
8. C. Security, "The State of DevSecOps," 2020.
9. T. E. Gasiba et al., "Raising Security Awareness of Cloud Deployments using Infrastructure as Code through Cybersecurity Challenges," *ACM,* 2021.
10. Elkaduwe et al. “Verified Protection Model of the seL4 Microkernel”. In: University of New South Wales Sydney, Australia, 2008.
11. Ross J. Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems. 2nd ed. Wiley Publishing, 2008.