vCDS: A Virtualized Cross Domain Solution Architecture

Nathan Daughety
90th COS
United States Air Force
San Antonio, USA
nathan.daughety@us.af.mil

Marcus Pendleton
90th COS
United States Air Force
San Antonio, USA
marcus.pendleton.2@us.af.mil

Shouhuai Xu
Dept. of Computer Science
University of Col. Col. Springs
Colorado Springs, USA
sxu@uccs.edu

Laurent Njilla
Cyber Assurance Branch
Air Force Research Laboratory
Rome, USA
laurent.njilla@us.af.mil

John Franco
Dept. of E.E. and C.S.
University of Cincinnati
Cincinnati, USA
franco@ucmail.uc.edu

***Abstract*—With the paradigm shift to cloud-based operations, reliable and secure access to and transfer of data between differing security domains has never been more essential. A Cross Domain Solution (CDS) is a guarded interface which serves to execute the secure access and/or transfer of data between isolated and/or differing security domains defined by an administrative security policy. Cross domain security requires trustworthiness at the confluence of the hardware and software components which implement a security policy. Security components must be relied upon to defend against widely encompassing threats – consider insider threats and nation state threat actors which can be both onsite and offsite threat actors – to information assurance. Current implementations of CDS systems use sub-optimal Trusted Computing Bases (TCB) without any formal verification proofs, confirming the gap between blind trust and trustworthiness. Moreover, most CDSs are exclusively operated by Department of Defense agencies and are not readily available to the commercial sectors, nor are they available for independent security verification. Still, more CDSs are only usable in physically isolated environments such as Sensitive Compartmented Information Facilities and are inconsistent with the paradigm shift to cloud environments. Our purpose is to address the question of how trustworthiness can be implemented in a remotely deployable CDS that also supports availability and accessibility to all sectors. In this paper, we present a novel CDS system architecture which is the first to use a formally verified TCB. Additionally, our CDS model is the first of its kind to utilize a computation-isolation approach which allows our CDS to be remotely deployable for use in cloud-based solutions.**

I. Introduction

In the context of military information systems, a *domain* is an environment which contains a set of computer-based systems, processes, data, controls, and security policies defined by a classification label, which serves in isolation from other systems and can only be accessed using a defined set of rules. Similarly, a *security domain* is a system or set of systems separated from other domains by a boundary defined by an administrative security policy. The objective of the security policy is to uphold classification level, information access and transfer regulations, and data ownership within a domain. Creating a secure connection between security domains necessitates the implementation of multifaceted security policies for information flow management in a CDS. *Cross domain* refers to the access to and/or transport of data across domains of isolated and/or differing classification levels. A *CDS* enforces a security policy on an interface between the discrete security domains. The terms *high* and *low* are used herein, to describe domains of higher and lower security classification levels.

Problem Statement. The DoD and the Intelligence Community (IC) manage CDS services, devices, and the standards which CDSs must abide by, almost exclusively, through the Unified Cross Domain Management Office (UCDMO), and the National Cross Domain Strategy Management Office (NCDSMO) [14]. Furthermore, [13] details the CDS needs outlined by the UCDMO that are exclusively written for DoD agencies, which is also the case for the security policies outlined by the National Security Agency (NSA) NCDSMO. While there are a few CDS solutions available outside the DoD/IC such as [47, 48], this community manages the CDS standards and technologies which are leveraged by these systems. This not only creates the problem of general CDS availability outside the DoD, but also means these systems are expensive and must be redesigned for specific environments [14].

Further challenges with the current status quo of CDS designs include reliability and assurance (trustworthiness), remote deployability, and accessibility [14, 18]. [22] also states that current CDS products are available only as secure appliance or “strong box” implementations meaning they reside in a physically isolated environment and are unfit for cloud environments as they are not remotely deployable. As a consequence, existing solutions are either highly specialized systems which cannot be applied to other use-case environments without incurring unreasonable modification and maintenance costs, or they are ad hoc solutions built upon technologies which lack assured security [13, 31, 32, 37].

These problems can be made manifest by the three CDS architectures of current CDS systems. The first architecture uses *physically isolated domains* to maintain one classification level per domain. This separation ensures that an authorized operator must maintain multiple physical infrastructures. One implementation is sometimes referred to as a "swivel-chair setup" because the operator could effectively swivel his or her chair to access each workstation while other implementations use a keyboard, video monitor, and mouse (KVM) switch to access different domains from a single workstation [25]. Oftentimes this architecture employs air gaps to transfer data using removable devices. The second architecture uses *partitioned workstations*, which relies on domain virtualization on top of a single host. The host regulates the separation between domains by running a corresponding virtual machine (VM) for each domain. The third architecture uses *data diodes*, which are analogous to electrical diodes, to restrict the flow of data in one direction (e.g. a domain of low classification may be permitted to transfer data to a domain of high classification but not the opposite) [25]. These tailored designs may explain why the Committee on National Security Systems (CNSS) called the current CDS architectures *niche CDSs* because they lack accessibility to commercial sectors outside of military/DoD designations [22, 42].

A critical observation regarding security systems, in general, is that many descriptions bundle the well-understood security objectives of confidentiality, integrity, availability, authenticity, and accountability (CIAAA) together. In practice, this not only forces impractical redesign, but enforcing each of these objectives is simply not necessary if the threat model does not require it. Furthermore, each security objective requires the addition of more technologies which may incur risk beyond the defined threat model. Our system focuses on data confidentiality – ensuring that data spillage does not occur.

Summarizing the problem, the facts presented above pose an elevated risk to data confidentiality for multiple reasons. First, a system that has not been mathematically proven trustworthy should not be trusted to securely maintain data. Second, the status quo in both commercial and DoD-specific CDS technology is inconsistent with the paradigm shift to cloud computing and does not allow for secure remote deployability. Third, *security through obscurity* is exposed as a failed security technique, and the need for independent validation of security properties is revealed.

Our Contributions. In this paper, we systematically examine and aim to correct the challenges of trustworthiness, remote deployablility, and accessibility, identified in the papers above. We present a novel architecture, called *virtual CDS* (vCDS), which overcomes the weaknesses of current CDS systems with three key contributions: (i) Trustworthiness through execution on top of a formally verified, TCB; (ii) Accessibility to commercial sectors using commodity software and hardware; and (iii) Secure remote deployability for offsite computing (e.g. cloud). Additionally, we propose a prototype instantiation of the vCDS architecture. To the best of our knowledge, we are the first to develop a general purpose CDS system, which leverages a TCB that is provably secure and has been comprehensively verified for functional correctness and security guarantees, that can be deployed in a variety of use-cases and environments.

II. Related Work

CDS systems are offered by a few technology companies [47, 48] but are managed and tested by the DoD/IC. These systems seem to conform to security through obscurity as their specifications and evaluation results are not available for independent verification. As a consequence, the specifications and evaluations of such systems may not be sound. This can be justified by the fact that evaluations based on the Common Criteria are problematic because: (i) "usability is ignored"; (ii) the paperwork, not the product, is the test subject; and (iii) these schemes are known to "squeeze a very volatile and competitive industry into a bureaucratic straightjacket, in order to provide purchasers with the illusion of stability" [10]. From a technical perspective, we design vCDS by leveraging the state-of-the-art microkernel, seL4, which has a number of formally verified properties [20, 26, 44].

III. Background

This section introduces the technological components commonly found in secure computing systems, the functions and objectives these components aim to meet, and the distinction between the concepts of trust and trustworthiness as they pertain to secure computing systems.

Trusted Execution Environment. A Trusted Execution Environment (TEE) "is a secure, integrity-protected processing environment, consisting of processing, memory, and storage capabilities" [19]. The goal of a TEE is to improve security through runtime-state protection and data restriction to ensure no sensitive data leaves the TEE. This can be achieved through the implementation of a dedicated VM [6] or an environment that runs alongside but is isolated from the main OS [23], such as a hardware-based enclave supported by security co-processors. A TEE must define mechanisms to "securely attest its trustworthiness", not allowing untrusted code or operations to cause, enable, or prevent any code execution, traps, exceptions, or interruptions [24]. [24] introduces five building blocks of TEEs: (i) *Secure Boot*: assure that only the correct, unmodified code can be loaded; (ii) *Secure Scheduling*: assure a balanced and efficient coordination between the TEE and the rest of the system so that tasks running in the TEE do not affect the responsiveness of the main OS; (iii) *Inter-Environment Communication*: interface for assuring authenticity in the communication between the TEE and other system components; (iv) *Secure Storage*: data confidentiality and integrity is preserved in storage; and (v) *Trusted I/O Path*: protect authenticity and confidentiality of the communication between the TEE and peripheral devices. TEEs are used to protect complex and interconnected systems that require a high level of security with protection against both physical and software-based attacks.

There has been an emergence of TEE technology in commodity systems. The most common of which includes Intel's

Software Guard Extensions (SGX), AMD's Secure Encrypted Virtualization/Secure Memory Encryption (SEV/SME), and ARM's TrustZone. TEE systems have evolved greatly from proprietary solutions to a standards-based approach for use in PC's, mobile devices, and other Internet-connected devices [29]. Furthermore, the push for a viable TEE solution for use in cloud computing environments has begun.

Trusted Computing Base. The core element for any security solution is the TCB. Historically, a TCB has referred to several types of computing bases including, but not limited to: a security kernel, a trusted operating system, a security filter or individual access control validation mechanism, or an entire trusted computing system [3, 4]. The integral components included in modern TCBs are the *reference monitor* and *security kernel*. The reference monitor serves to provide complete mediation of access, validating access to all objects by authorized subjects. The security kernel provides the lowest level of software to hardware abstraction and employs mechanisms to enforce security at differing trust boundaries.

In order for a TCB solution to be robust, its components must be sound. While TCB implementations have changed over the years, the objectives which determine a sound TCB have remained: the system should provide access controls and self-protecting mechanisms such that no system modification or interference can take place and it should be designed and functional correctness verified using formal and/or semi-formal methods [3]. The TCB is responsible for isolating security-bound components and upholding the security policy which describes "the conditions under which information and system resources can be made available to the users of the system" [3].

Therefore, a TCB, commonly implemented with a combination of hardware, firmware, and software, is the totality of protection mechanisms responsible for enforcing a security policy [4]. These protection mechanisms ensure that any system components which are not included in the TCB should not need to be trusted for the whole system to remain fully protected [4]. Emphasis in TCB design, as of late, is on assurance and trustworthiness through formal verification of trusted computing systems [1, 3, 9].

Trust vs Trustworthy. The DoD Trusted Computer System Evaluation Criteria (TCSEC), first released in 1983, states that the "assurance of correct and complete design and implementation for these systems is gained mostly through testing of the security-relevant portions of the system" where the security-relevant portion refers to the TCB [4]. Assurance is measured as confidence in the TCB to meet explicitly identified security expectations [12]. The Common Criteria, which replaced the aforementioned evaluation criteria in 2005, introduces the Evaluation Assurance Level (EAL) which goes beyond security testing to introduce formal specifications and formal verification of computing bases [12]. The goal of an EAL measurement is to exercise the distinction between trust and trustworthiness, where a trusted system is a system that is *believed* to be capable of handling a security event and a trustworthy system is *proven* to be able to handle a security event. In other words, trust can be broken but trustworthiness has been formally proven and cannot be broken. However, as we know from [10], vendors often find loopholes to game the evaluation system, rendering the "illusion of stability" without actually proving trustworthiness.

Guards. A guard is a trusted "application which is responsible for analyzing the content of the communication and determining whether this communication is in accordance with the system security policy" and is often implemented in CDS systems [8]. Typically, guard components are implemented as filters which can modify or delete messages, verifiers which can check data integrity, and isolation mechanisms to separate the data. Guards are located at the border of a component's ingress and egress data channels.

IV. The vCDS Architecture

A. System Model

The vCDS system model is based on the Bell-LaPadula (BLP) model whose primary concern is data confidentiality [5]. BLP ensures that subjects can only read objects at or below their own security level, subjects can only create objects at or above their own security level, and all access requests must be authorized based on an access control matrix that characterizes the security level and rights of each subject. Furthermore, BLP preserves the *principle of least privilege* assuring that a subject may only access the minimum resources necessary for a particular operation. In fact, the BLP model is a CDS model in that is was designed to confront the "operational needs to move information between networks of different classifications and sensitivity levels" [7]. vCDS faithfully implements this model because it has been rigorously studied to ensure that confidentiality of data is protected, despite compromises to other security objectives such as integrity, availability, authenticity, and/or accountability, and it allows data to flow safely between certain isolated security domains.

B. Applications

The above model abstracts a family of systems which can support a range of applications such as those listed below:

1) Stream Processor: The discussion in [22] aims to address the problem of using classified data and technology for a security task on an unclassified host without revealing any information about the classified resources to the host. Our system is adaptable to implement an Intrusion Detection System/Intrusion Prevention System (IDS/IPS) and/or firewall technologies which use highly classified tools, or Indicators of Compromise, such as cryptographic signatures and analytics tools, to defend a host that is either unclassified or resides at a low security boundary. For example, one implementation is a network sensor that is composed of classified technology but is required to examine traffic on an unclassified system. The sensor employs IDS/IPS technology which must not leak any information, including covert channel information such as time or power, about itself while operating in a low environment.

2) Data Sharing: Secure information sharing must be realized between compliant parties in systems such as Fusion Centers, Real Time Tactical Information Centers, and Real Time Crime Centers where threat intelligence sharing (TIS) is required. One application of our CDS is in a blockchain-based TIS environment for robust and automated cyber security management (CSM). Our solution can be adapted to a system architecture based upon the B2CSM architecture, described in [43], with the goal of achieving secure intelligence collaboration through data collection, aggregation, analysis, and threat-related information dissemination to critical parties.

3) Big Data/High Performance Computing: Today's information processing workload on cloud/big data infrastructure makes apparent the need for scalable, remotely deployable, and high performance CDS systems. The current funnelling approach to data manipulation across classification boundaries in Big Data/Cloud environments degrades system performance to the point of ineffectiveness. These platforms require too much data to move efficiently as current approaches do which is inconsistent with MapReduce – the distributed computing paradigm for big data where the computation is moved to the data nodes as opposed to the data being moved to the computation [21]. Our CDS system is adaptable to big data and high performance computing environments where distributed parallel processing, including the access and transfer, of large data sets is commonplace. In the case of a big data platform, our solution is consistent with MapReduce in that it moves the computation to the data. In the case of a high performance computing platform, our solution can move the data to the compute nodes. vCDS is a versatile solution that provides the needed scalability and low cost implementations in a field ruled only by highly specialized systems [13, 18]. Our solution can achieve high performance/BDP goals as a general purpose, security baseline solution with very little cost, and is adaptable to the particular use case.

C. Threat Model

The threat model focus is to prevent the compromise of data confidentiality through various information disclosure attacks, specifically focusing on attempts to leak data from the high side to the low side entities. Common vectors by which attackers are able to create data spillage in sensitive computer systems include covert channels, unauthorized access to resources, and disrupting the flow of data by forcing movement against the intent [42]. Furthermore, the threat model includes powerful adversaries who are given insider access and, therefore, are granted privileges that an outsider would not possess [24]. The attack surface includes all communications between the secure enclave and the high side target [22]. For example, powerful attackers who are able to spoof the secure enclave to intercept high-trust communications are a grave threat to data confidentiality. The model also "includes all software attacks and the physical attacks performed on the main memory and its non-volatile memory" [24] and, also, physical bus attacks.

Fig. 1 provides insight into the hardware and software vulnerabilities defined in our threat model as well as references to the various mitigations employed by each technology component (including an optional guard) of our system which are further discussed later in Security Analysis.

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]	

Fig. 1. Threat Model Matrix

D. Security Requirements

There are three essential property measurements expressed in a comprehensive CDS protection plan, as detailed by [15, 22]: (i) *defensive effectiveness*: timely and accurate prevention and response, system flexibility; (ii) *confidentiality*: data content protection from unauthorized parties; and (iii) *operational relevance*: usable and accessible in multiple operational environments. These three measurements re-enforce our primary security objective to protect confidentiality of data. While our system is equipped to protect and enforce other objectives in the CIAAA, these are orthogonal to our main objective. In order to achieve data confidentiality and enforce our protection plan in the presence of the threats mentioned above, the security design needs to ensure the following which are discussed in Security Analysis: Hardware Protections and Memory Encryption, Trustworthy Components, Decidable Object Security and Staticity, Computation Isolation, and Data Flow Restriction.

E. Architectural Design

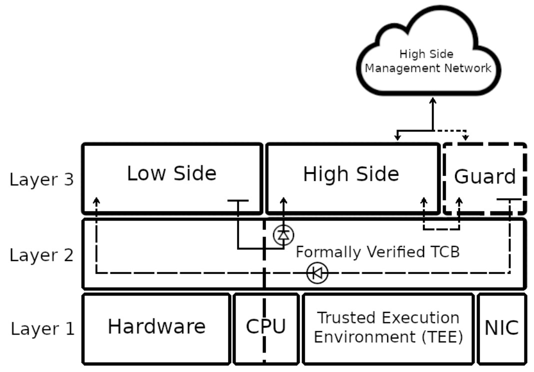

Fig. 2. General purpose vCDS architecture

In order to achieve the security objectives mentioned above, we propose the following CDS security architecture, depicted in Fig. 2, with three layers which include the (i) Hardware, (ii) Computing Base, and (iii) Software Components.

1) Hardware: Found in Layer 1, the hardware-based TEE capability corresponds to all High Side (Layer 3) data and computations while the Low Side (Layer 3) leverages the basic hardware capabilities. Within the hardware level there is also a central processing unit (CPU) and a network interface card (NIC). The CPU manages all processing of data for the components residing in Layers 2 and 3. The NIC functions only with the High Side and guard on Layer 3.

2) Computing Base: In Layer 2, on top of the TEE, is a formally verified, TCB which serves as the fundamental component in this system and allows our solution to be easily adapted to different implementation requirements and CDS architectures. The TCB ensures integrity and confidentiality through a trustworthy codebase and access controls providing isolation and *staticity* – further discussed in Security Analysis.

3) Software Components: In Layer 3, there are two separate processes running on top of the formally verified TCB: one that represents the High Side and one that represents the Low Side. The Low Side component manages the low classified data while the High Side manages the higher classified data.

The High Side leverages one device and driver in order to communicate with a high side management network (C2): a NIC. The High Side tunneling strategy allows the isolated high enclave to communicate with components of the same classification which, in our case, is an optional guard and the C2. Functions of the latter are relative to the system, for example, the C2 in a CDS with the primary purpose of analyzing and filtering network traffic would need to regularly push signature updates and blocking actions to sensitive intelligence sensors and traffic analyzers as well as receive alerts should a malicious packet be discovered. In a distributed computing CDS system, the C2 would regularly push code to each computing node which would then run an operation defined by the code and send the results back to the C2.

Before data passes from high to low, there is an optional guard that functions as a filter to ensure that no high sensitive data are passed to low. The guard is not required in implementations where the high does not pass data back to low. This guard can have several additional functions like sending alerts back to the High Side, but in any case, it has bi-directional communication with the High Side as it resides at the same classification level. This is an important distinction from the rest of the data flow model because the direction of data flow is restricted with a data diode as depicted with a diode symbol in Fig. 2. We further discuss the data diode in Security Analysis.

F. Security Analysis

Now we analyze how the security architecture achieves the security objectives and requirements mentioned above.

1) Hardware Protections and Memory Encryption: Relative to the High Side, the TEE is used to mitigate attacks from more privileged software and physical attacks with transparent memory encryption as well as protection of memory at rest, memory in transit, and memory in use which helps mitigate our threat model. Additionally, we add padding mechanisms to increase execution time of data processing to mitigate data leakage through timing analysis. Note that [36] presents a method to eliminate timing channels and cross-domain temporal interference in a formally verified microkernel, discussed in the following paragraph, though it has not yet been integrated as of this writing. It is important to note here that without the TEE, our solution provides complete formal verification but lacks the security measures required for a secure, remotely deployable system. With the TEE, our system achieves our objective of secure remote deployability, however, at the time of this writing, no effort has been made to formally verify the enhancement of our solution. The formal proofs for the enhancement are intended for future development.

2) Trustworthy Components: The formally verified code base assures that no software vulnerabilities exist in its operation and that the system is proven trustworthy.

3) Decidable Object Security and Staticity: A capability-based access control model governs all kernel services so that any applications wanting to perform an operation must invoke a capability that has sufficient access rights for the service making object security decidable [2, 11, 40]. There is no implicit memory allocation within the kernel, only explicit request via capability invocation [11]. Furthermore, all hardware resource partitioning is governed by capability distribution, that is, authority distribution. The component architecture model combines with the capability model to enforce *staticity* – a property which ensures that configurations occur before compile time so that all channels and privileges are pre-allocated and that no channels or added privileges can exist outside of what is predefined. Therefore, the threat vectors involving attacks stemming from dynamic creation of channels and propagation of privileges are mitigated. The access control model also allows the system to grant specific communication capabilities which enable authorized and controlled communication between components, thus enabling, with a high degree of assurance, component isolation [40]. Component isolation and communication authenticity further mitigate threat vectors outlined in our threat model by ensuring that no user or process can access resources without authorization.

4) Computation Isolation: In addition to kernel and kernel-enforced component isolation, we have component and computation isolation within the High and Low Side domains. On the Low Side, the necessary drivers and data management services and computation are isolated from the High Side. In contrast, the High Side hides all sensitive intelligence used to analyze the low data from the Low Side.

5) Data Flow Restriction: The data diode ensures that data can only travel from low to high and never back to low, thus mitigating the confidentiality threat model. If the data must travel from high to low through a corresponding data diode, the data will first pass through the guard which ensures that no sensitive data are passed to the low, again mitigating threat vectors in our threat model.

For whichever use case vCDS is implemented, secure communication between components of the same classification level helps to mitigate the threat vectors in our threat model, while providing a secure path of remote deployability.

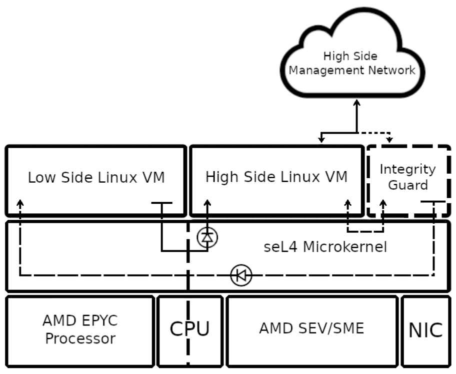

Fig. 3. vCDS tech architecture

V. The vCDS Implementation

The following section details the resilient, cross-layer implementation of vCDS and each system component, shown in Fig. 3, from our selection of hardware TEE to our use and development of software applications used by the high and low components. Note here that the described implementation is tailored to our stream processor application, which we call a network sensor (IPS/IDS), for the purposes of better understanding the architecture.

A. For Realizing Layer 1 in the Architecture: AMD EPYC with SEV/SME

1) Implementation: We selected the AMD EPYC processor with it's Secure Encrypted Virtualization (SEV) and Secure Memory Encryption (SME) technologies to use as our hardware-based TEE to secure the high components as it supports the protection against the threat vectors outlined in our threat model. The hardware accelerated memory encryption mechanism has two components, the AMD Secure Processor and the AES-128 hardware encryption engine [30]. The Secure Processor is a dedicated security processor which provides cryptographic functionality for secure key generation and key management. The encryption engine encrypts data as it is written to main memory and decrypts data as it is read from main memory when provided with the key. SME employs a single session-sensitive key, generated by the AMD Secure Processor at boot, to encrypt all of system memory. The functionality of SME provides encryption capabilities for data at rest, data in transit, and data in use. Furthermore, memory encryption is transparent so it can support any operating system. SEV leverages one cryptographic key per VM to enforce isolation between each VM, the host, and the hypervisor.

2) Security Analysis: VM isolation ensures that if an attacker has access to the hypervisor, host, or has control over one VM, the attacker will not be able to read the memory of any other VMs as the memory will be encrypted. Our system takes advantage of the per-VM encryption keys to ensure that High Side data confidentiality is maintained in the hardware. Additionally, when the CDS use-case calls for a high side remote management network (C2), the isolation of the VM from the host and the hypervisor supports our goal of remote deployability [27].

B. For Realizing Layer 2 in the Architecture: seL4

1) Implementation: We have chosen to implement the seL4 operating system microkernel as our TCB, in order to leverage the trustworthiness provided through its formal verification and security guarantees.

The access control model components used in seL4 are capabilities. Capabilities are "access tokens which support very fine-grained control over which entity can access a particular resource in a system" [44]. A capability, which is an immutable object reference, enforces the principle of least privilege by ensuring that the only way an operation can be performed on a component is by invoking the capability which is pointing to that object, thus restricting the granted rights to the absolute minimum required to perform the operation.

2) Security Analysis: seL4 is the only existing capability-based microkernel system with a proof of functional correctness which "guarantees that every behavior of the kernel is predicted by its formal abstract specification" [44]. The microkernel is also provably secure – "seL4 comes with further proofs of security enforcement" [20], and employs the Take-Grant protection model such that, "in a correctly configured seL4-based system, the kernel guarantees the classical security properties of confidentiality, integrity and availability" [44]. seL4 further ensures safety of time-critical systems by being the world's fastest performing microkernel while providing fine-grained access control [16, 44].

C. For Abstracting Layer 2 and Linking to Layer 3 Components: CAmkES

1) Implementation: In order to abstract away the low-level seL4 components, we have chosen to utilize the component architecture for microkernel-based embedded systems (CAmkES) framework. This component framework allows us to build and manipulate our CDS on top of the static architecture of seL4. CAmkES abstracts over low-level kernel mechanisms, providing communication primitives and support for decomposing a system into functional units [33]. CAmkES components use dataports, which are port interfaces, to enable one component to pass large amounts of data to another component. Dataports are made available to CAmkES components as shared memory regions at runtime.

2) Security Analysis: CAmkES grants our system the assurance that the components, interfaces, and connectors which have been specified in the architecture description language is an accurate representation of all possible interactions and that any interaction beyond what is specified will not materialize [44]. Additionally, our dataport configuration is an explicit data diode that allows us to pass data structures through the protected seL4 kernel via a unidirectional interface without the possibility of leaking information through the component or kernel layers.

D. For Realizing Layer 3 in the Architecture: Linux and seL4 Native Process

1) Implementation: To represent the High and Low Side domains, we elected to use virtual machines to run our custom-built Linux kernel due to practicality of implementation. The Low Side handles incoming traffic and transports the appropriate data to the High Side for processing. Communication from high to low within the network sensor application is a separate channel protected by a guard. The guard or filter we have developed is an optional guard component which we specifically elected to use in our stream processor application.

2) Security Analysis: One of the processes applied to the data on the High Side is an IPS/firewall, which, in our case, is Snort [49]. The filter adheres to the primary objective of integrity by implementing both an integrity guard and a firewall which we call a disposition guard. The integrity guard ensures that upon crossing a trust boundary, the data have not been modified. Our implementation leverages the speed and security of the Blake3 cryptographic hashing algorithm in order to check the integrity of the data. While Blake3 is not formally verified, we leverage *n-version programming* in order to improve the algorithm's reliability. The disposition guard filters packets in or out based on a list of source IP addresses, ports, and other properties to further mitigate threats.

VI. Analysis and Evaluation

In this section we demonstrate the performance of vCDS through carefully selected benchmarks which measure the throughput transfer of data from the Low Side to the High Side. Fig. 4 depicts the vCDS pipeline throughput, that is, the number of bytes which can be processed in one second of operation, compared to that of a native Linux process which uses shared memory as a transfer channel. There are two throughput measurements, a partial transfer where the data is available to high without any processing taking place, and a full transfer where high does process the data. Additionally, we include the performance overhead of vCDS.

Throughput (Bytes/Second)			
Pipeline Transfer	vCDS	Native Linux Process	vCDS Overhead
Partial	326,705,400	403,745,175	19.1%
Full	150,696,450	214,413,750	29.7%

Fig. 4. vCDS vs Native Linux Process Throughput in 1 Second

VII. Discussion

Limitations. The vCDS implementation has three limitations. First, the vCDS design inherits the limitations of its building-blocks. Specifically, the verified properties of seL4 are not complete in the sense that they do not capture time properties [34, 38]. This presents the risk of an attacker opening a covert timing channel which could be exploited to attack seL4 and thus vCDS. Additionally, the verification of seL4 comes with three assumptions described in [44]: (i) the hardware behaves as expected; (ii) the specification is correct; and (iii) the theorem prover is correct. It remains to be a challenge to validate these assumptions. Second, although CAmkES, leveraged on top of seL4, can offer formally verified security enforcements when correctly configured [20], there is, to the best of our knowledge, no existing formulation to verify that a CAmkES configuration is correct. Third, the use of AMD SEV/SME exposes our system to some potential threats that are not considered in our threat model [28, 39, 41, 46].

Future Work. In addition to addressing the limitations mentioned above, there are three research directions for future studies. First, in order to further enhance the security of vCDS systems while reducing their cost and addressing scalability limitations associated with vendor hardware, it is interesting to instantiate the abstract TEE used in the vCDS architecture via RISC-V based Keystone Enclave. This is possible because Keystone claims to solve many of the limitations surrounding AMD's SEV/SME and Intel's SGX schemes, and it is available for independent, hardware formal verification [45]. Second, it is useful to test the scalability and performance of vCDS in real-world big data and cloud computing environments. Third, to assure the implementation of proper access control configuration for any particular use case, it is useful to develop an auditing tool. The model validator will ensure that a specific CAmkES assembly guarantees no leak of information from high to low can occur with our data diode implementation.

VIII. Conclusion

In this paper we have examined the state-of-the-art in CDS technology and detailed the problems and concerns associated with the systems. As a response, we have presented a solution to the issues described herein with our system, vCDS. To the best of our knowledge, vCDS is the first CDS system built upon a formally verified TCB. The main objectives of this system are to ensure trustworthiness, promote multi-sector availability through effective, low-cost technologies and allow secure deployment of the system remotely. vCDS provides a new and much improved security baseline tailored to defensive effectiveness, data confidentiality, and operational relevance, allowing us to build systems which cater to many different environments (e.g. cloud), use-cases and functionalities on top of a security assured system. Furthermore, the inherent risks and limitations of our system are well understood and do not conform to the defacto standard of ad hoc solutions and security through obscurity. Our hope is that these limitations influence further evaluation and research on this topic.

Acknowledgement. This work is supported in part by ARO Grant #W911NF-17-1-0566, NSF Grants #2122631 (#1814825) and #2115134, and Colorado State Bill 18-086.

References

[1] P. Neumann et al. "A Provably Secure Operating System." In: 1975.

[2] R. J. Lipton and L. Snyder. "A Linear Time Algorithm for Deciding Subject Security". In: *ACM* (1977).

[3] G. H. Nibaldi. "Specification of a Trusted Computing Base (TCB)". In: The Mitre Corporation. 1979.

[4] Department of Defense. "DoD Trusted Computer System Evaluation Criteria". In: 1985.

[5] Ryan Ausanka-Crues. “Methods for Access Control: Advances and Limitations”. In: 2001.

[6] Tal Garfinkel et al. “Terra: A Virtual Machine-Based Platform for Trusted Computing”. In: ACM, 2003.

[7] D.E. Bell. “Looking back at the Bell-La Padula model”. In: *21st Annual Computer Security Applications Conference*. 2005.

[8] Jim Alves-foss et al. “The MILS architecture for high-assurance embedded systems”. In: *International Journal of Embedded Systems* (2006).

[9] Ed Colbert and Barry Boehm. “Cost Estimation and for Secure and Software and Systems”. In: Center for Systems & Software Engineering, University of Southern California. 2006.

[10] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. 2nd ed. Wiley Publishing, 2008.

[11] Dhammika Elkaduwe, Gerwin Klein, and Kevin Elphinstone. “Verified Protection Model of the seL4 Microkernel”. In: 2008.

[12] Common Criteria. “Common Criteria for Information Technology Security Evaluation”. In: 2009.

[13] B. Farroha, M. Whitfield, and D. Farroha. “Enabling Net-Centricity through Cross Domain Information Sharing”. In: IEEE, 2009.

[14] Bassam S. Farroha Deborah L. Farroha et al. “Challenges and Alternatives in Building a Secure Information Sharing Environment through a Community-Driven Cross Domain Infrastructure”. In: IEEE, 2009.

[15] K. Scarfone and P. Mell. “Guide to Intrusion and Detection and Prevention Systems and (IDPS)”. In: NIST, 2009.

[16] B. Blackham et al. “Timing Analysis of a Protected Operating System Kernel”. In: *IEEE 32nd Real-Time Systems Symposium*. 2011.

[17] Michael Hanspach and Jorg Keller. “In Guards We Trust: Security and Privacy in Operating Systems Revisited”. In: IEEE, 2013.

[18] Bernard F. Koelsch and Army War College Carlisle Barracks PA. *Solving the Cross-Domain Conundrum*. 2013.

[19] N. Asokan et al. “Mobile Trusted Computing”. In: *Proceedings of the IEEE* (2014).

[20] Gerwin Klein et al. “Comprehensive Formal Verification of an OS Microkernel”. In: *ACM* (2014).

[21] Feng Li et al. “Distributed Data Management Using MapReduce”. In: *ACM* (2014).

[22] B. M. Thomas and N. L. Ziring. “Using Classified Intelligence to Defend Unclassified Networks”. In: 2014.

[23] Brian McGillion et al. “Open-TEE – An Open Virtual Trusted Execution Environment”. In: IEEE, 2015.

[24] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. “Trusted Execution Environment: What It is, and What It is Not”. In: IEEE, 2015.

[25] Scott Smith. “Shedding Light on Cross Domain Solutions”. In: *SANS Institute Information Security Reading Room* (2015).

[26] Gernot Heiser and Kevin Elphinstone. “L4 Microkernels: The Lessons from 20 Years of Research and Deployment”. In: *ACM* (2016).

[27] D. Kaplan, J. Powell, and T. Woller. “AMD Memory Encryption”. In: *AMD Developer Central* (2016).

[28] Zhao-Hui Du et al. “Secure Encrypted Virtualization is Unsecure”. In: (2017).

[29] Secure Technology Alliance. “Trusted Execution Environment (TEE) 101: A Primer”. In: Secure Technology Alliance, 2018.

[30] AMD. *Enhance your Cloud Security with AMD EPYC™ Hardware Memory Encryption*. Tech. rep. 2018.

[31] United States Government US Army. *JP 3-12 Cyberspace Operations*. CreateSpace Independent Publishing Platform, 2018.

[32] CloudShield. “CloudShield CS-4000: Trusted Network Security Platform (TNSP)”. In: (2018).

[33] Gerwin Klein et al. “Formally Verified Software in the Real World”. In: *ACM* (2018).

[34] Anna Lyons et al. “Scheduling-context capabilities: a principled, light-weight operating-system mechanism for managing time”. In: 2018.

[35] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. “Insecure Until Proven Updated: Analyzing AMD SEV’s Remote Attestation”. In: 2019.

[36] Qian Ge et al. *Time Protection: The Missing OS Abstraction*. 2019.

[37] Looking Glass. “LookingGlass IRD-100 Data Sheet: Stealth Threat Response at the Network Edge”. In: (2019).

[38] Gernot Heiser, Gerwin Klein, and Toby Murray. “Can We Prove Time Protection?” In: ACM, 2019.

[39] Mengyuan Li et al. “Exploiting Unprotected I/O Operations in AMD’s Secure Encrypted Virtualization”. In: *28th USENIX Security Symposium*. 2019.

[40] seL4 Reference Manual Version 11.0.0. “Data61 Trustworthy Systems https://ts.data61.csiro.au/projects/TS”. In: 2019.

[41] AMD. *AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More*. Tech. rep. 2020.

[42] Australian Cyber Security Center. “Fundamentals of Cross Domain Solutions”. In: *MENA Report* (2020).

[43] Songlin He et al. “Blockchain-Based Automated Cyber Security Management”. In: (2020).

[44] Gernot Heiser. “The seL4 Microkernel – An Introduction”. In: 2020.

[45] Dayeol Lee et al. “Keystone: an open framework for architecting trusted execution environments”. In: 2020.

[46] Phillip Mestas. “Securing AMD SEV”. In: 2020.

[47] OWL Cyber Defense. *OWL Cyber Defense Cross Domain Solutions*. https://owlcyberdefense.com/. 2021.

[48] Forcepoint. *Forcepoint Cross Domain Solutions*. https://www.forcepoint.com/. 2021.

[49] Snort. *Snort: Open Source Intrustion Prevention System*. https://www.snort.org/. 2021.