Confidential Execution of Cloud Services

Tommaso Cucinotta, Davide Cherubini, Eric Jul
Bell Laboratories, Alcatel-Lucent, Dublin, Ireland
{name.surname}@alcatel-lucent.com

Keywords: Security, Cloud Computing, Confidentiality

Abstract: In this paper, we present *Confidential Domain of Execution* (CDE), a mechanism for achieving confidential execution of software in an otherwise untrusted environment, e.g., at a Cloud Service Provider. This is achieved by using an *isolated execution environment* in which any communication with the outside untrusted world is *forcibly encrypted* by trusted hardware. The mechanism can be useful to overcome the challenging issues in guaranteeing confidential execution in virtualized infrastructures, including cloud computing and virtualized network functions, among other scenarios. Moreover, the proposed mechanism does not suffer from the performance drawbacks typical of other solutions proposed for secure computing, as highlighted by the presented novel validation results.

1 INTRODUCTION

Information and communication technologies (ICT) are undergoing a continuous and steep evolution and the wide availability of high-speed broadband connections is causing an inescapable shift towards distributed computing models. The traditional idea of a personal computer able to process data locally, is giving way to alternative computing paradigms, e.g., cloud computing, where personal computing devices are merely the point of access for data and computing services offered elsewhere. In this context, more and more often virtualization technology plays a major role in enabling new ICT scenarios. For example, in cloud computing it is used to enable deployment of multiple Virtual Machines (VMs) running on the same physical hosts. Network technologies are following the same trend, with the introduction of concepts such as Network function Virtualization (NfV) (NFV Industry Specif. Group, 2012), Software-Defined Networking (SDN) (McKeown et al., 2008; O. M. E. Committee, 2012) and Network Virtualization (Anderson et al., 2005).

Virtualizing (and sharing) network functions and infrastructures allow network operators to (1) minimize capital investments due to less expensive-longer life cycle hardware, (2) decrease operational expenditures due to, among others, reduced energy consumption, maintenance and co-location costs, and (3) speed-up the deployment of services. While *operational confidentiality* in shared network infrastructure has been widely investigated (Fukushima et al., 2011), a similar problem needs to be addressed when multiple operators are co-hosted and implement their network activities within the same system.

For example, in mobile communication, solutions have been presented to create multiple virtual base stations (V-BTS) on a single base station hardware platform (Sachs and Baucke, 2008; Chapin, 2002) or to enable cellular processing virtualization in data centers (Bhaumik et al., 2012). If a physical device is compromised, an attacker can potentially eavesdrop conversations, disrupt normal operations or even steal cryptographic material. Also, malicious or buggy software running on one virtual instance may leverage possible security weaknesses in the isolation mechanisms to compromise the confidentiality of other co-located virtual instances.
Similar attacks can potentially be performed in a SDN infrastructure when multiple controllers (e.g., OpenFlow controllers) are deployed and co-hosted onto remote shared servers. One of the major problems hindering the uptake of such scenarios is therefore ensuring the proper level of security and isolation for software deployed by independent customers (a.k.a., *tenants*) within a shared physical equipment owned by others. Similarly, in cloud Infrastructure-as-a-Service (IaaS) provisioning, customers deploy Virtual Machines (VMs) within physical infrastructures owned by the provider (see Figure 1). If a user is using the cloud merely as a remote storage provider, confidentiality can readily be guaranteed through client-side encryption. Often, this scenario is obtained when deploying *cloud encryption gateways*, often coupled with Software-as-a-Service (SaaS) clouds. However, this way users cannot exploit computing facilities in

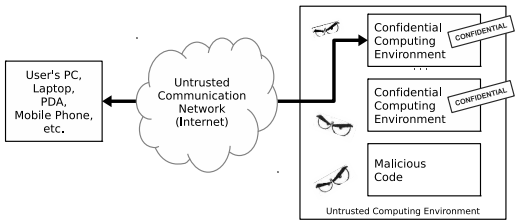

Figure 1: The problem of confidential execution of cloud services.

the cloud. The types of attack described above can be mitigated by using *trusted computing technologies* coupled with a secure boot process, in which a node is allowed to boot and run only after verifying that its software and hardware configuration has not been modified. However, the *Trusted Computing Base* (TCB), i.e., the set of hardware, firmware and software that a user needs to trust to keep data confidential, may be quite large (see Section 2).

In this paper, we illustrate a minimum set of features required to ensure confidential execution in an otherwise untrusted physical infrastructure. Our point is that all we need is a mechanism that completely isolates a user's execution environment, making its data *unreachable* from the outside, and that guarantees that all data exiting the environment is *forcibly encrypted*. In the proposed architecture, called *Confidential Domain of Execution* (CDE) (Cucinotta et al., 2014), the set of functionality to be trusted to ensure confidentiality is so limited that it is straightforward to envision a hardware-only implementation. Therefore, we can reduce the software part of the TCB of the provider to *zero*, and leave to the end-user full control of the code to deploy within the execution environment. Our solution neither allows the owner or administrator(s) of the physical infrastructure to control anyway, nor to spy upon, the software and data being deployed and processed within the CDE, so a CDE is also protected from malicious insiders.

Note that, in addition to summarizing the general architecture of our CDE solution that is going to appear on (Cucinotta et al., 2014), in this paper we also present novel results that validate the approach, showing that the CDE is not affected by the serious performance drawbacks typical of other proposed solutions for secure processing in untrusted environments.

2 RELATED WORK

We provide below a brief overview of solutions for trustworthy execution of software in untrusted environments, in the three areas of OS/run-time robustness, trusted computing, and secure processors.

OS/run-time Robustness – Traditionally, memory protection hardware mechanisms (Memory Management Units—MMUs) guarantee trust and isolation between execution environments. MMUs allow an Operating System (OS) kernel to isolate the execution environments of different users. However, the CPU can still enter a special mode of operation (the so-called *Ring 0*) in which malicious software can bypass normal OS security. Bugs in the OS kernel, in system processes, and in system calls implementations, can be exploited to gain administrator's privileges and subvert any security policy in the system or execute malicious code (Duflot et al., 2006). In addition, a malicious system administrator can overcome any security restriction.

Similarly, in virtualized environments, memory protection (including virtualization extensions (Uhlig et al., 2005; Advanced Micro Devices, Inc., 2008; Abramson et al., 2006)) can be leveraged to isolate the execution of different Virtual Machines (VMs). Still, the hypervisor embeds code that exploits the special mode of operation of the available processor(s) as to perform system management actions, thus bugs in the hypervisor can be exploited to break the VMs isolation. To completely rule out such a possibility, Szefer et al. (Szefer et al., 2011a) propose exploiting multi-core platforms in a way that allows a VM to run without the hypervisor mediation on a subset of the available hardware resources. However, the administrator of the infrastructure can still arbitrarily access any data managed by the hosted VMs. Novel cryptographic mechanisms, such as *homomorphic encryption* (Popa et al., 2011; Brenner et al., 2011), allow a provider to perform computations on encrypted data without being able to understand the data contents. However, applicability and wide adoption of such technique seem limited.

Trusted Computing – Numerous projects (e.g., see (Singaravelu et al., 2006)) have shown how to design a small software TCB, decreasing the likelihood of attacks. In OS designs, the historical idea of microkernels (Liedtke, 1995; Rashid, 1986), proposed as an alternative to monolithic OS architectures, aims to reduce the size of the critical part of a kernel which the system stability relies upon. In (Steinberg and Kauer, 2010), the authors apply a similar idea to hypervisors, developing the concept of *micro-hypervisor*. In this case, however, one still has to trust a non-trivial software TCB, i.e., the micro-hypervisor, albeit it is smaller than regular hypervisors.

A partial remedy to these problems, comes from the use of *Trusted Platform Modules* (TPMs) (TPM, 2011). For example, in (Correia, 2012), a Trusted Virtualization Environment (TVE) has been proposed,

leveraging TPM technologies to allow remote users check the software stack of a remote physical machine before deploying their VMs. However, TPMs provide no guarantees when bugs that can cause leakage of confidential information are present in the TCB (Hao and Cai, 2011). In (Keller et al., 2010), NoHype was presented, an architecture where the hypervisor is removed and the VMs have direct control on system resources that are automatically reserved to each of them. However, NoHype assumes that the server provider and the guest operating system are fully trusted and therefore it does not provide confidentiality of data and code running on the hosted VMs.

Secure Processors — In secure processor architectures (Lie et al., 2000; Suh et al., 2003b; Chhabra et al., 2010), data is kept in the main RAM memory in encrypted form: it is decrypted only within the secure processor when needed, then it is re-encrypted when it is written back to memory. Therefore, a physical attack aiming at spying on the traffic on the bus would only manage to see encrypted data. The main problem resides in the performance penalty incurred by the processor each time data has to be exchanged with the main-memory, i.e., at every cache-miss (for the cache level(s) residing inside the processor chip), causing the processor to stall waiting for data to be fetched and decrypted. To mitigate this, counter-mode encryption has been proposed (Suh et al., 2003a; Yang et al., 2003), in which the encryption scheme is realized as a XOR of the plain-text (cipher-text) and a one-time pad. The degradation of performance can be estimated as the additional cycles required for the XOR operation. However, counter-mode solutions have problems related to the management of the pads used for the encryption processes, and may limit the possibility to move blocks in the memory. Also, for the scheme to be effective, the encryption operations must complete within the memory access cycle, what makes the encryption engine expensive to be realized.

Our CDE solution aims to guarantee confidentiality without requiring expensive encryption and decryption operations at every memory access. It leverages a hardware-based enforced encryption of *the final output only* of the computations performed by the software running within the CDE. The remote user of the CDE has total control over the whole software stack running within the CDE. Furthermore, it is not possible to use any special mode of operation of any processor outside of a CDE, to spy on its memory.

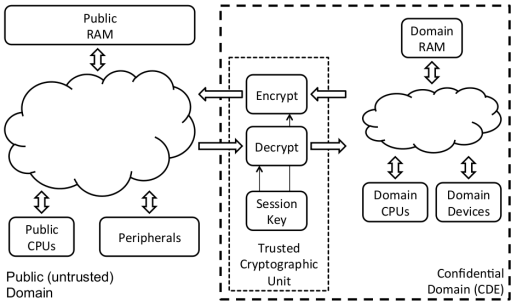

Figure 2: Overview of the Confidential Domains of Execution (CDE).

3 PROPOSED APPROACH

In this section, we briefly describe *Confidential Domain of Execution*(CDE), an abstract architecture that guarantees confidentiality in computing environments. For further details, the reader can refer to (Cucinotta et al., 2014).

In the remaining of the paper, we use the following conventions: the term *owner* (or *provider*, or *administrator*) indicates the entity with *physical* control over the computing machines. The term *user* (or *customer*) is the *remote user* interested in handing over computations to the provider's computing machines.

Informal CDE Description. Figure 1 illustrates the basic idea behind the CDE: to create, on a physical computing machine, one or more protected computing areas (CDE) that can be exclusively accessed via an encrypted channel. As shown in Figure 2, each CDE includes one or more processing units along with their caches (Domain CPUs), RAM memory (Domain RAM), and peripherals (Domain Devices) used for computations and data movement. The behavior of a CDE is summarized as follows:

1. inside the CDE, confidential data/code is processed/executed in plain-text form (*unencrypted*), at the *native computing speed* of the physical platform;
2. any data flowing out of the CDE to the (potentially untrusted) outside world is *forcibly encrypted* and only the remote user (and other authorized users) can decrypt it; similarly, any data entering the CDE must be in encrypted form, and is decrypted when injected within;
3. all encryption operations are performed by a hardware Trusted Cryptographic Unit (TCU). Its function can neither be disabled nor worked around; not even the administrator of the physical machines is able to overcome the security features of a CDE;

4. a CDE can be reconfigured (reset) at any time by its owner. Then, the entire contents of the CDE including any memory and all CPU state is *forcibly cleared* by the CDE hardware;
5. the cryptographic material, used by the TCU to encrypt/decrypt any data crossing the CDE boundaries, is established by the remote user by using a cryptographic protocol (Cucinotta et al., 2014), that guarantees that the TCU is the *only* entity having access to it;
6. the CDE is manufactured with a built-in asymmetric key pair, where the private key is embedded in the TCU and is present nowhere else and the public key is made available through a Public Key Infrastructure.

Summarizing, as shown in Figure 1, a user can exploit the proposed mechanism to move confidential code and data for processing purposes onto a remote untrusted server owned by a (either trusted or untrusted provider). Due to the use of the built-in cryptographic material, the remote user can confidentially communicate with the target CDE, despite the traversal of untrusted networks and untrusted computing elements on the same physical machine where the CDE resides (i.e., without the use of secure communication).

CDE Implementation. The CDE architecture as depicted in Figure 2, can be implemented as a single System-on-Chip (SoC) or in more cost effective, but less secure, way by using Commercial-Off-The-Shelf (COTS) hardware elements. For a more exhaustive discussion of the different CDE implementations, including the protocol used to exchange the cryptographic keys, see (Cucinotta et al., 2014).

4 VALIDATION

Our proposed technique may also have a positive performance impact when compared to other solutions discussed in Section 2. Particularly relevant is comparison with secure processor architectures, where the memory is kept in encrypted form in main memory, and it is decrypted on-the-fly at every cache-miss.

We validated our approach using micro-benchmark use-cases that are used to give a rough estimate of the performance advantage we achieve. In the considered micro-benchmarks, the private computations handed over to the remote computing environment consist of: 1) **Matrix Inversion:** this can be a common operation to be used for solving big linear equations problems; 2) **Eigenvalue Computation:** this may constitute a recurrent operation in the context of physical simulations, where systems of integral/differential equations need to be solved; 3) **2D Fast Fourier Transform:** this is a common operation in image and video processing, particularly when applying linear filters. These benchmarks have been realized using the open-source tool GNU Octave[1], and invoking the functions `inv()`, `eig()` and `fft2()` on randomly generated square matrices.

Additionally, we considered a "macro-benchmark" service consisting of a video transcoder useful to change the resolution of a video so as to scale it down to a number of lower resolutions. We used the open-source `ffmpeg` software. All benchmarks have been performed on a laptop equipped with 4 logical cores Intel© Core™ i5-2520M and 4 GB of RAM, under the following conditions: 1) CPU clock frequency locked to 2.50 GHz; 2) 3 logical cores disabled at kernel level; 3) real-time priority (to minimize the impact of other running system services). L2 cache-misses have been measured by running the `stat` command of the Linux `perf` profiler tool. Assuming a strong cipher such as AES-128 (AES, 2001), implemented in CBC mode, and a 32-bit data bus used to transfer a 64-byte data block, the overall encryption and decryption operation delays have been measured to be 97 and 141 CPU cycles, respectively (Szefer et al., 2011b). Therefore, given the above assumptions, the average number of cache misses CM, the CPU clock frequency CPU_{freq} and the average execution time $time_{AV}$, then the percentage encryption overhead of the XOM-type architectures (Lie et al., 2000; Chhabra et al., 2010), can be roughly estimated as:

$$OH_{enc} = 97 \cdot \frac{CM}{CPU_{freq} \cdot time_{AV}} \cdot 100 \qquad (1)$$

and the correspondent percentage decryption overhead is:

$$OH_{dec} = 141 \cdot \frac{CM}{CPU_{freq} \cdot time_{AV}} \cdot 100 \qquad (2)$$

The considered benchmarks, along with the key used parameters and the encryption and decryption overheads, are summarized in Table 1. Results show that the impact on performance of the additional cryptographic operations is significant for most of the performed benchmarks. Nevertheless, in case of CPU greedy computations such as `eig()`, the time required by the crypto-computation is larger (by a factor of two) than the time required to perform the whole operation in plain-text. The proposed CDE solution

[1]More information is available at: `www.gnu.org/software/octave/`.

Table 1: Summary of the used benchmarks and configuration parameters, and obtained performance data.

Benchmark	Matrix Size	Average Time [s]	Cache-miss Average	Cache-miss Ratio	Encryption Overhead	Decryption Overhead
inv()	2048x2048	3.894	21737705.7	16.15%	21.66%	31.48%
inv()	4096x4096	27.907	180510029.5	26.81%	25.10%	36.48%
eig()	512x512	1.278	22529316.4	22.90%	68.40%	99.43%
eig()	1024x1024	11.06	430773713.7	55.37%	151.12%	219.67%
fft2()	1024x1024	0.32	1402260.0	17.73%	17.00%	24.71%
fft2()	2048x2048	0.47	3447163.9	26.57%	28.46%	41.37%
Benchmark	Matrix Size	Average Time [s]	Cache-miss Average	Cache-miss Ratio	Encryption Overhead	Decryption Overhead
ffmpeg	320x240	0.510	5704824.0	49.87%	43.40%	63.09%
ffmpeg	640x480	0.652	6729660.8	51.92%	40.05%	58.21%

does not have performance degradation due to cryptographic overhead because the whole computation is performed in plain text and cryptographic operations are required only when data or instructions enter or leave the protected area. On the other hand, as mentioned in Section 2, a significant performance improvement can be achieved using counter-mode AES architectures. In this case the encryption/decryption overhead can be estimated in the additional one-cycle XOR operation (Yang et al., 2003) for every cache miss. Still, the CDE presents advantages as compared to this last solution. Indeed, the CDE does not require the additional cycle operation, but more importantly it does not present the memory management issues of counter-mode AES architectures.

5 CONCLUSIONS

Virtualization of network functions onto standard servers creates potential confidentiality issues when the infrastructure provider is either untrusted or trusted but curious. Multi-tenancy adds a dimension of complexity to the confidentiality problem because an attack might actually come from software deployed by other customers.

We have presented a solution enabling confidential processing of data and code in an untrusted environment and shown that it keeps native performance of the underlying hardware at the same time as guaranteeing high security levels. With our proposal, the software TCB that an infrastructure provider must provide is reduced to *zero—we do not have to trust a single line of code provided by the service provider*. What we do have to trust is the hardware manufacturer to accurately and reliably produce the necessary hardware. Our less expensive realization using COTS components requires us to place some extra trust in the provider.

In the future, we plan to realize a working prototype on FPGA, to get concrete overhead data, and compare with other proposals. Also, a firmware-based implementation of the TCU has to be investigated, to provide upgradability of the cryptographic algorithms in use.

References

(2001). Federal Information Processing Standards Publication 197 – Specification for the Advanced Encryption Standard (AES). U.S. Governement.

(2011). *TPM Main - Part 1 - Design Principles - Specification Version 1.2 - Revision 116*. Trusted Computing Group, Incorporated.

Abramson, D. et al. (2006). Intel® Virtualization Technology for Directed I/O. *Intel Technology Journal*, 10(3):179–192.

Advanced Micro Devices, Inc. (2008). AMD-V™ Nested Paging. AMD Techincal White Paper.

Anderson, T., Peterson, L., Shenker, S., and Turner, J. (2005). Overcoming the Internet Impasse through Virtualization. *Computer*, 38(4):34–41.

Bhaumik, S., Chandrabose, S. P., Jataprolu, M. K., Kumar, G., Muralidhar, A., Polakos, P., Srinivasan, V., and Woo, T. (2012). CloudIQ: a framework for processing base stations in a data center. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 125–136.

Brenner, M., Wiebelitz, J., von Voigt, G., and Smith, M. (2011). Secret program execution in the cloud applying homomorphic encryption. In *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, pages 114–119.

Chapin, J. (2002). Overview of vanu software radio.

Chhabra, S., Solihin, Y., Lal, R., and Hoekstra, M. (2010). An Analysis of Secure Processor Architectures. In

Gavrilova, M. and Tan, C., editors, *Trans. on Computational Science VII*, volume 5890 of *LNCS*, pages 101–121. Springer.

Correia, M. (2012). Software execution protection in the cloud. In *Proceedings of the 1st European Workshop on Dependable Cloud Computing*, EWDCC '12, New York, NY, USA. ACM.

Cucinotta, T., Cherubini, D., and Jul, E. (2014). Confidential Domains of Execution. *to appear in Bell Labs Technical Journal*, 19(1).

Duflot, L., Etiemble, D., and Grumelard, O. (2006). Using CPU System Management Mode to Circumvent Operating System Security Functions. In *CanSecWest*.

Fukushima, M., Hasegawa, T., Hasegawa, T., and Nakao, A. (2011). Minimum Disclosure Routing for network virtualization. In *Proc. of* 14^{th} *Global Internet Symposium (GI) 2011 at IEEE INFOCOM 2011*.

Hao, J. and Cai, W. (2011). Trusted Block as a Service: Towards Sensitive Applications on the Cloud. In *Trust, Security and Privacy in Computing and Communications (TrustCom), Proc. of* 10^{th} *Int. Conf. on*, pages 73–82.

Keller, E., Szefer, J., Rexford, J., and Lee, R. B. (2010). Nohype: virtualized cloud infrastructure without the virtualization. *SIGARCH Comput. Archit. News*, 38(3):350–361.

Lie, D., Thekkath, C. A., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. C., and Horowitz, M. (2000). Architectural Support for Copy and Tamper Resistant Software. In *ASPLOS*, pages 168–177. ACM Press.

Liedtke, J. (1995). On micro-kernel construction. *SIGOPS Oper. Syst. Rev.*, 29(5):237–250.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

NFV Industry Specif. Group (2012). Network Functions Virtualisation. Introductory White Paper.

O. M. E. Committee (2012). Software-defined Networking: The New Norm for Networks. Open Networking Foundation.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). CryptDB: protecting confidentiality with encrypted query processing. In *Proc. of the* 23^{rd} *ACM Symp. on Operating Systems Principles*, SOSP '11, pages 85–100.

Rashid, R. F. (1986). From RIG to Accent to Mach: the evolution of a network operating system. In *Proc. of 1986 ACM Fall joint computer conference*, ACM '86, pages 1128–1137.

Sachs, J. and Baucke, S. (2008). Virtual radio: a framework for configurable radio networks. In *Proceedings of the* 4^{th} *Annual International Conference on Wireless Internet*, WICON '08, pages 61:1–61:7.

Singaravelu, L., Pu, C., Härtig, H., and Helmuth, C. (2006). Reducing TCB complexity for security-sensitive applications: three case studies. *SIGOPS Oper. Syst. Rev.*, 40(4):161–174.

Steinberg, U. and Kauer, B. (2010). NOVA: a microhypervisor-based secure virtualization architecture. In *Proc. of the* 5^{th} *European Conf. on Computer systems*, EuroSys '10. ACM.

Suh, G. E., Clarke, D., Gassend, B., Dijk, M. v., and Devadas, S. (2003a). Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proc. of the* 36^{th} *annual IEEE/ACM Int. Symp. on Microarchitecture*, MICRO 36, Washington, DC, USA. IEEE Computer Society.

Suh, G. E., Clarke, D., Gassend, B., van Dijk, M., and Devadas, S. (2003b). AEGIS: architecture for tamper-evident and tamper-resistant processing. In *ICS '03: Proc. of the* 17^{th} *annual Int. Conf. on Supercomputing*, New York, NY, USA. ACM.

Szefer, J., Keller, E., Lee, R. B., and Rexford, J. (2011a). Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In *Proc. of CCS 2011*, Chicago, Illinois, USA.

Szefer, J., Zhang, W., Chen, Y.-Y., Champagne, D., Chan, K., Li, W. X. Y., Cheung, R. C. C., and Lee, R. B. (2011b). Rapid single-chip secure processor prototyping on the OpenSPARC FPGA platform. In *Int. Symp. on Rapid System Prototyping*, pages 38–44.

Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C. M., Anderson, A. V., Bennett, S. M., Kagi, A., Leung, F. H., and Smith, L. (2005). Intel Virtualization Technology. *Computer*, 38(5):48–56.

Yang, J., Zhang, Y., and Gao, L. (2003). Fast Secure Processor for Inhibiting Software Piracy and Tampering. In *Proc. of the* 36^{th} *annual IEEE/ACM Int. Symp.on Microarchitecture*, MICRO 36, pages 351–, Washington, DC, USA. IEEE Computer Society.