SecureCloud: Secure Big Data Processing in Untrusted Clouds

Florian Kelbert*, Franz Gregor†, Rafael Pires‡, Stefan Köpsell†, Marcelo Pasin‡,
Aurélien Havet‡, Valerio Schiavoni‡, Pascal Felber‡, Christof Fetzer†, Peter Pietzuch*

*Imperial College London, United Kingdom, {fkelbert, prp}@imperial.ac.uk
†TU Dresden, Germany, {firstname.lastname}@tu-dresden.de
‡University of Neuchâtel, Switzerland, {firstname.lastname}@unine.ch

***Abstract*—We present the SecureCloud EU Horizon 2020 project, whose goal is to enable new big data applications that use sensitive data in the cloud without compromising data security and privacy. For this, SecureCloud designs and develops a layered architecture that allows for (i) the secure creation and deployment of secure micro-services; (ii) the secure integration of individual micro-services to full-fledged big data applications; and (iii) the secure execution of these applications within untrusted cloud environments. To provide security guarantees, SecureCloud leverages novel security mechanisms present in recent commodity CPUs, in particular, Intel's Software Guard Extensions (SGX). SecureCloud applies this architecture to big data applications in the context of smart grids. We describe the SecureCloud approach, initial results, and considered use cases.**

I. Introduction

Despite a steady increase in cloud adoption over the past few years, some challenges remain. Confidentiality, integrity, and availability of applications and their data are of immediate concern to organisations that use cloud computing. This is particularly true for organisations that must comply with strict policies, including those which process personal data or that support society's most critical infrastructures, such as finance, health care, and smart grids. The goal of SecureCloud is to address such concerns by providing solutions that allow for the secure processing of sensitive data within untrusted clouds.

The primary area of application of the developed solutions is in the field of critical infrastructures, whose operators have legitimate concerns about the dependability of applications hosted in third-party clouds. Despite security guarantees given by cloud operators, dependability concerns increasingly become a barrier to the broad adoption of cloud computing. The cloud therefore becomes itself a critical infrastructure for which we need to provide sufficient guarantees so that we can justifiably place our trust in their hosted applications.

The overall goal of our work is to develop a platform that enables the dependable implementation, deployment and execution of critical applications within untrusted cloud environments. Our objectives are:

1) substantially improve the state-of-the-art in cloud dependability by developing innovative and effective mechanisms to enforce security, covering integrity and confidentiality, as well as availability and reliability;

2) seamlessly integrate new dependability features into a standard cloud stack to encourage easy migration of critical (as well as non-critical) applications to the cloud without compromising application dependability; and

3) convincingly validate and demonstrate the benefits of our approach by applying it to realistic and demanding big data use cases in the domain of critical infrastructures (smart grids).

II. Existing Approaches to Cloud Security

A modern public cloud is home to a hardware and software stack consisting of many devices, a large codebase and large frameworks that are often immature, rapidly evolving, and full of bugs and configuration errors that can be exploited by attackers. The challenge for operators is to convince potential clients that it is safe to execute their applications and store their data in such a dangerous environment.

One approach taken by operators is to define a Trusted Computing Base (TCB) within their stack [1]. Typically, the TCB includes most of the basic middleware, operating system (OS), and networking facilities of the data centre, as well as its hardware platform. Establishing the credibility of the TCB amounts to verifying the correctness and security of a large and complex hardware and software system. High costs aside and repeating it on a continuous basis as the hardware and software evolve, the goal of a truly "trustworthy" TCB has proven elusive [2], [3]. Even if it were possible to remove all bugs from the TCB, this alone would not ensure their security—given insufficient physical security or a malicious system administrator, there could still be unauthorized access to customer data when it is unencrypted in memory.

An approach focused specifically on securing application data from access by both external and internal malicious agents is based on homomorphic encryption—a technique intended to allow encrypted computations to be carried out on encrypted data [4], [5]. Since unencrypted computations and data would never be present in the cloud, they would never be exposed to attacks. As of now, the realisation of homomorphic encryption is proving as elusive [6] and impractical for virtually all real-world applications due to its immense overheads, precluding its use in timely demanding applications.

A third approach is the use of specialised security co-processors [7]. Such processors consider the chip area as a

trust boundary, treating everything outside as subject to attacks and potentially compromised. The instructions and data are stored encrypted in the memory. Once read by the processor, they are decrypted and the instructions carried out on plain-text data. Since everything outside the chip can be tampered with, the processor never outputs plaintext data, encrypting it before writing to the system bus. While secure processors provide good security guarantees, specialised hardware use is counter to the general principle of data centre "scale out" notion, which advocates the use of large numbers of commodity components.

III. SecureCloud Approach

A. Intel SGX and Small Trusted Computing Base

The innovative approach to cloud dependability pursued by SecureCloud uses novel cryptographic hardware found in upcoming commodity CPUs—in particular, Intel SGX [8], [9]. It allows protected execution on encrypted data where the corresponding plaintext is only known inside the processor. The *enclave* is a secure area in which the processing of the plaintext data happens. Applications are thus isolated not only from other applications but also from the underlying operating system and hypervisor. Users run sensitive applications in public clouds without unconditionally trusting the cloud provider.

B. Layered Architecture

The SecureCloud approach tackles secure processing in the cloud using SGX from a full stack perspective. The architecture builds on several layers and various technologies:

(1) Secure containers for QoS-aware applications. While many hardware extensions for CPU/IO/memory virtualization have reduced the overhead of virtual machines, container frameworks such as Docker[1] are still more efficient, although less secure since their security is directly linked to the underlying host OS. We address this tension by designing and implementing a solution for secure containers. The developed components also monitor hardware usage to detect resource bottlenecks and allow for accounting and billing.

(2) Dependable micro-services for the cloud utilise these secure containers. For this, we design and implement a framework and related interfaces which allow for the development of arbitrary, yet secure, micro-services. We further implement a few common micro-services.

(3) Secure distributed big data applications on the basis of secure micro-services address big data processing. The developed big data processing components are leveraged by the application level demonstrators in the context of smart grids as described in Section VI. Examples of developed components are secure structured data stores, map/reduce based computations, schedulers, as well as components for efficient transmission of large amounts of data.

IV. SecureCloud Infrastructure

Figure 1 shows the baseline infrastructure of SecureCloud. An application consists of a set of micro-services connected via an event bus. The application logic of each micro-service lives within an enclave. The micro-service runtime exists outside of the enclave. These runtime functions only access encrypted data. Encryption and decryption of this data is performed automatically and transparently within the enclave. This approach limits the amount of code added to the TCB.

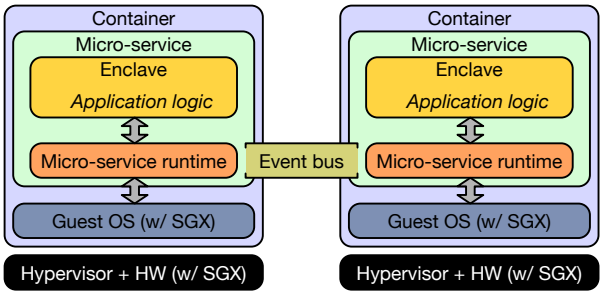

Figure 1. SecureCloud applications consist of a set of micro-services connected by an event bus. Our main focus is to enhance the security of containers. These containers may share the host with virtual machines, i.e., the system still contains a hypervisor.

To deploy the micro-service, we offer *secure containers* on top of the untrusted stack of the cloud provider: a secure container adds confidentiality and integrity to Docker containers. This enables system administrators to build secure container images within a trusted environment and to run them in an untrusted cloud. To facilitate the creation of secure containers, we designed and developed a Secure Linux Container Environment (SCONE) [10] that secures existing applications with SGX.

To the micro-service, SCONE exposes an *external system call based interface*, which is shielded from attacks. To protect itself from user space attacks, SCONE performs sanity checks and copies all memory-based return values to the inside of the enclave before passing the arguments to the micro-service. SCONE further (i) transparently encrypts and authenticates data that is processed via file descriptors, and (ii) provides acceptable performance by implementing tailored threading and an asynchronous system call interface.

SCONE integrates with existing Docker environments, and ensures that secure containers are compatible with standard containers. The host OS, however, must include a Linux SGX driver and, to boost performance, a SCONE kernel module.

With respect to Docker container deployment and scheduling, SecureCloud contributes GenPack [11], a scheduling and monitoring framework that leverages principles from generational *garbage collection* (GC) [12]. The core idea of GenPack is to partition the servers into several groups, named *generations*. It combines runtime monitoring of system containers to learn their requirements and properties, and a scheduler that manages different generations of servers.

V. Preliminary Results

SecureCloud already developed prototypes: an unmodified Docker ecosystem to securely deploy micro-services (Section V-A) and SCBR [13], a secure messaging system over

[1] https://www.docker.com/

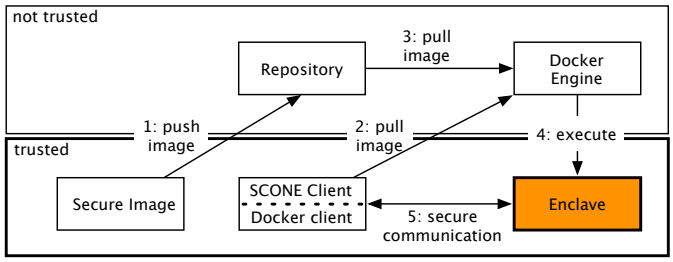

Figure 2. Using secure containers with Docker

content-based routing that allows to securely hook-up individual micro-services to full-fledged applications (Section V-B).

A. Secure Docker Containers

Micro-services need a runtime environment. In line with Section III-B, we chose to deploy micro-services using a containerized *Docker* infrastructure, a currently popular and widely used platform. Each micro-service is executed within a *secure container*—a dedicated Docker container that runs their code protected by SGX enclaves. From the perspective of the Docker infrastructure, secure containers are indistinguishable from regular containers.

The integration of secure containers with Docker requires changes to the image build process. We further provide a wrapper for the Docker client, called *SCONE client*, which provides functionalities for spawning secure containers and for secure communication with containers. Note that we do not require modifications to the Docker Engine or its API. SCONE supports the typical Docker workflow: developers publish a Docker image featuring their micro-service; end-users can customize this image by adding additional file system layers.

We assume that Docker images contain micro-services that are created in a trusted environment (see Figure 2). The image creator must be familiar with the security-relevant aspects of the micro-service, e.g., which files must be protected. Next we explain the secure container image creation process.

First, the image creator builds a protected executable of the micro-service by statically compiling against its library dependencies and the SCONE library—a C library that ensures that the application is only executed inside of an SGX enclave. SCONE does not support shared libraries by design to ensure that all enclave code is verified by SGX upon enclave creation.

Second, the image creator uses the SCONE client to protect the image's file system (FS). The SCONE client encrypts all files that must be protected and creates an FS protection file, which contains the message authentication codes (MACs) for file chunks as well as the encryption keys. The FS protection file itself is then encrypted and added to the image.

Lastly, the secure image is published using the standard Docker registry. As all security-relevant parts of the image are protected by the FS protection file, we do not need to trust the Docker registry. To allow for the secure image's further customization, the image creator would only sign the FS protection file, but not encrypt it. This way, the image's integrity is ensured. Confidentiality can then only be assured after finishing the customization process.

Each secure container requires a startup configuration file (SCF). The SCF contains keys to encrypt standard I/O streams, the hash and encryption key of the FS protection file, application arguments, as well as environment variables. Only an enclave whose identity has been verified can access the SCF, which is received through a TLS-protected connection that is established during enclave startup.

B. Secure Content Based Routing

Content-based routing (CBR) is a flexible and powerful paradigm for scalable communication among distributed processes. It decouples data producers from consumers, and routes messages based on their content. Although extensively studied [14], the publish/subscribe communication model still fails to reach wide deployment due to privacy concerns.

To perform efficient routing, a CBR router must see the content of the messages, as well as subscriptions by data consumers, a clear threat to privacy. We provide a secure CBR engine called *SCBR* [13]. It exploits SGX to perform this matching step. Hence, the compute-intensive CBR operations can operate on decrypted data shielded by enclaves and leverage efficient matching algorithms.

Outside of secure enclaves, both publications and subscriptions are encrypted and signed, thus protecting the system from unauthorised parties observing or tampering with the information. SCBR combines a key exchange protocol and a state-of-the-art routing engine to provide both security and performance while executing under the protection of an enclave. Performance is enhanced by storing subscriptions in data structures that exploit containment relations between filters. Therefore, a reduced number of comparisons is required whenever a message must be matched against them.

We evaluate SCRB with several workloads to observe the sources of performance overheads and trade-offs of SGX. Our time measurements inside/outside of enclaves highlighted performance degrades when cache misses rate increase (i.e. data must be evicted or fetched to/from system memory, causing SGX to perform encryption, decryption, integrity and freshness checks). While cache misses imposes some limited overhead, they are less critical than memory swapping.

Since the enclave page cache (EPC) memory is limited to 128MB, pages must be evicted from the protected area to the main (untrusted) memory whenever more space is required. Memory swapping is serviced by the operating system, which causes higher overheads when compared to cache misses. Figure 3 shows this effect by showing the combined results of matching times when executing the same code inside and outside secure enclaves. Performance degrades to nearly 18× for a subscription database of 200MB. Even if EPC size was set to 128MB (marked by the vertical line), the performance drop is evident before due to the use of protected memory for SGX internal data structures.

These first results open the way for further research to minimise memory footprint and build an enclave-efficient

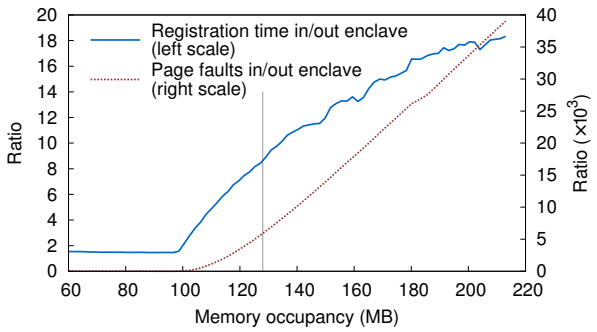

Figure 3. Effect of memory swapping

system. We intend to optimise our data structures to avoid paging and cache misses. We expect these optimisations to further decrease the overhead of running inside an enclave.

VI. Application Use Cases

To demonstrate the need and the feasibility of using secure clouds, the project considers use cases in the area of smart grids, which offer opportunities to tackle many of the requirements that sensitive big data applications may face when executing in the cloud. First, they account for a growing volume of data, as meters and sensors for monitoring distribution and transmission grids continuously collect and transmit data. Second, the energy distributors' data analyses require access to consumers' detailed information about energy consumption, which represents an enormous privacy risk (their activities and behaviours can be inferred [15]). Finally, data analysis can trigger reactions that interfere with the physical world (load control or consumer notifications). Adversaries could thus have devastating effects on the power system.

In our first use case, smart meters collect detailed power consumption data from residential and industrial consumers. Collecting data at sub-minute granularities enables for sophisticated applications, such as power theft prevention and early detection of power quality issues. Nowadays, such applications are deployed on dedicated servers maintained by utilities and system integrators. Hence, several customers cannot use them, because it would require a large data storage and processing infrastructure. Cloud computing offers such an infrastructure. Nevertheless, once this data is under control of a cloud provider, an adversary who compromises this provider's infrastructure could gain access to them, hence the need to be stored and processed securely.

The second use case considers applications that affect energy delivery and fault detection. For these applications, data sources may be public, but the data needs to be reliable and the processing tasks that trigger actions in the smart grid must be executed in a timely fashion. These applications will be supervised using monitoring services. Orchestration services detect anomalies within milliseconds, which requires adaptations to the virtual infrastructure that hosts the application. This fine-granular and highly responsive orchestration system will enforce quality-of-service guarantees without interfering with security and privacy requirements, and can even provide better energy efficiency. Our experiments with GenPack [11] show that up to 23% energy savings are possible for typical data-center workloads.

VII. Conclusions

The EU SecureCloud project designs and develops technologies to enhance the dependability of future cloud environments that host critical infrastructures such as the smart grid. Building upon Intel's SGX technology, the developed solutions allow for the secure creation, deployment, and execution of big data applications in untrusted clouds. The developed solutions are applied to the area of smart grids, which accounts for an ever-growing volume of data and demands for reliable, secure, and timely data processing. Our initial SGX-based prototypes for secure and efficient data processing and content routing demonstrate the promise of the SecureCloud approach.

Acknowledgements. The SecureCloud project has received funding from the European Union's Horizon 2020 research and innovation programme and was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under grant agreement number 690111. Rafael Pires is also sponsored by CNPq, National Counsel of Technological and Scientific Development, Brazil.

References

[1] Trusted Computing Group, "Trusted Platform Module Main Specification, version 1.2, revision 116," 2011.

[2] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri, "How low can you go?: Recommendations for hardware-supported minimal tcb code execution," *SIGARCH Comput. Archit. News*, vol. 36, no. 1, 2008.

[3] A.-R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy, "Tcg inside?: A note on tpm specification compliance," in *Proc. First ACM Workshop on Scalable Trusted Computing*, 2006, pp. 47–56.

[4] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st ACM Symposium on Theory of Computing*, 2009, pp. 169–178.

[5] M. Tebaa, S. E. Hajji, and A. E. Ghazi, "Homomorphic encryption method applied to cloud computing," in *Network Security and Systems (JNS2), 2012 National Days of*, April 2012, pp. 86–89.

[6] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM workshop on Cloud computing security workshop*. ACM, 2011, pp. 113–124.

[7] M. Lindemann, R. Perez, R. Sailer, L. van Doorn, and S. Smith, "Building the IBM 4758 Secure Coprocessor," *Computer*, vol. 34, no. 10, 2001.

[8] V. Costan and S. Devadas, "Intel SGX Explained," Tech. Rep.

[9] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proc. 2nd Intl. Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[10] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure Linux Containers with Intel SGX," in *OSDI*, 2016.

[11] A. Havet, V. Schiavoni, P. Felber, M. Colmant, R. Rouvoy, and C. Fetzer, "GenPack: A generational scheduler for cloud data centers," in *IEEE International Conference on Cloud Engineering 2017 (to appear)*, 2017.

[12] H. Lieberman and C. Hewitt, "A real-time garbage collector based on the lifetimes of objects," *Commun. ACM*, vol. 26, no. 6, pp. 419–429, Jun. 1983.

[13] R. Pires, M. Pasin, P. Felber, and C. Fetzer, "Secure content-based routing using intel software guard extensions," in *ACM/IFIP/USENIX 17th International Middleware Conference*, 2016.

[14] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, 2003.

[15] U. Greveler, P. Glösekötterz, B. Justusy, and D. Loehr, "Multimedia content identification through smart meter power usage profiles," in *Proc. Intl. Conf. on Information and Knowledge Engineering*, 2012, p. 1.