TrApps: Secure Compartments in the Evil Cloud

Stefan Brenner
TU Braunschweig, Germany
brenner@ibr.cs.tu-bs.de

David Goltzsche
TU Braunschweig, Germany
goltzsche@ibr.cs.tu-bs.de

Rüdiger Kapitza
TU Braunschweig, Germany
rrkapitz@ibr.cs.tu-bs.de

ABSTRACT

The cloud computing paradigm enables the flexible and scalable outsourcing of workloads. However, cloud customers are often reluctant to entrust their sensitive data with cloud providers. This is due to the fact that the infrastructure is owned by another company and a resulting loss of control. With the recent advent of powerful ARM hardware targeted for data centres, there is the opportunity of using trusted execution technology provided by ARM TrustZone to enhance the protection of cloud customer's data.

In this paper we propose *TrApps*, a secure platform for general-purpose trusted execution in an untrusted cloud with multiple isolated tenants based on the ARM TrustZone technology. Our system targets the parallel execution of partitioned applications of distinct tenants with lean security-sensitive components, and is based on a minimal trusted code base in the secure world of ARM TrustZone when compared to similar systems. In our evaluation we show the feasibility of our approach, and demonstrate its performance with trusted execution of *memcached* with an overhead of only 36.9% compared to the vanilla implementation and execution.

CCS Concepts

•Security and privacy → Distributed systems security;

Keywords

Cloud Computing, ARM TrustZone, Trusted Execution

1. INTRODUCTION

The economic impact of the cloud computing paradigm is ever-growing, due to benefits such as cost-efficiency, scalability and flexibility for both, cloud providers and customers [1]. However, in many cases a cloud deployment is not a suitable option: legal issues, geographical constraints as well as business secrets contained in the data are just a few of many reasons. The lack of trust in cloud providers by their customers is due to the providers ability to access and even change all code and data processed within the scope of their infrastructure [2, 3].

XDOM0'17, April 23, 2017, Belgrade, Serbia

DOI: http://dx.doi.org/10.1145/3071064.3071069

One way to tackle this problem is the use of trusted execution technology. *ARM TrustZone* provides trusted execution on the ARM architecture, by splitting the execution environment in a secure and an insecure compartment [4]. This technology also allows a cloud customer to establish trust in applications running in the secure world of TrustZone by the usage of remote attestation. Even though the ARM architecture has been more popular for embedded and mobile devices in the past, there has been a recent push towards powerful server-class platforms [5, 6]. For example, the AMD Opteron A1100 features a powerful multi-core 64 Bit ARM platform [7], and cloud providers have already started to adopt the ARM platform due to its cost and energy efficiency [8–10].

Remote attestation is key in establishing trust into a remote trusted execution environment, by measuring the code running inside it and proving the result to a remote party. However, naively attesting the whole software stack is impractical in a cloud scenario: the cloud provider would be forced to publish the source code of all his management and infrastructure software including the hypervisor to allow this. However, this is problematic since the competition between cloud providers results in highly-customized software [11]. In addition, any updates to attested software would require remote attestation again. Furthermore, the attested software stack would be huge, resulting in a higher probability for exploitable security vulnerabilities [12].

Opposed to that, our goal is to minimize the Trusted Code Base (TCB) by partitioning the application's code base, and running only small sensitive application components in a trusted environment. Hence, we aim at excluding the cloud provider's software stack from the TCB, in order to significantly reduce the TCB and avoid the above outlined security and flexibility issues.

In this paper we propose TrApps (**Tr**usted **Apps**), a platform for partitioned applications, tailored to an untrusted cloud environment. Our approach is based on the ARM architecture and makes use of TrustZone as an implementation of trusted execution. The TrApps platform comprises the lightweight micro kernel-based Genode Operating System (OS) framework [13] and features multiplexing TrustZone's secure world to run multiple secure application components simultaneously and isolated from each other. In contrast to the slim secure OS which is part of the TCB, we have a Linux system running in the untrusted environment, which is supposed to host the cloud provider's software stack. Our approach also contains an efficient solution to the technically challenging cross-world communication between the two worlds of TrustZone. The resulting system allows the execution of partitioned applications with small trusted compartments protecting sensitive data. We have proposed the application partitioning approach in our earlier *SecureKeeper* project [14]. In the evaluation of TrApps, we show

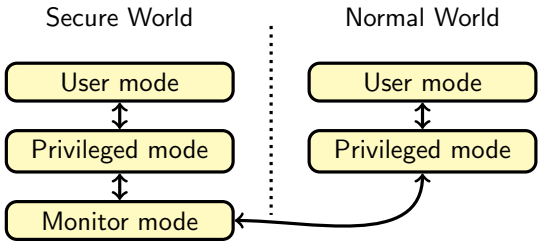

Figure 1: ARM TrustZone System Architecture.

the feasibility of our approach and measure its performance by the example of memcached[1] as a use case application of TrApps.

This paper is structured as follows: firstly, we provide the required background of our work in Section 2, followed by a detailed description of the design and implementation of TrApps in Section 3. Next, we describe our use case application memcached in Section 4 and show the results of our evaluation in Section 5. In Section 6 we summarise existing work and compare it with TrApps, and conclude the paper in Section 7.

2. BACKGROUND AND THREAT MODEL

In this section we briefly describe the fundamentals of the ARM TrustZone technology and the Genode OS framework that we used as a secure world operating system.

2.1 ARM TrustZone in a Nutshell

ARM TrustZone [4] is a security hardware extension for ARM processors that implements two virtual processors, each assigned to either the *normal world* or the *secure world*. This allows hardware-based isolation between these to worlds, comprising the system bus and peripheral's configuration. By this, two software components can run in parallel on the system, with the secure world being the more powerful one. The secure world is able to access normal world memory, while the normal world can only access the memory regions designated to normal world usage at system boot time.

During the early boot process, the processor first starts in secure mode and boots the secure operating system kernel which can be measured by hardware means for future attestation. Afterwards, the so called *monitor* must be initialised to support world switching. This involves switching processor registers during the world switch in order to resume the suspended execution. Finally, the boot process continues in normal world and a normal world OS can be booted, running concurrently with the secure world OS.

Depending on the configuration of the TrustZone Interrupt Controller (TZIC), interrupts can be assigned to one of the two worlds. Then, interrupts cause a world switch to the according world allowing interrupt handling by the respective world's software. For example, a timer interrupt can be assigned to secure world, to ensure liveness of secure world, while Network Interface Controller (NIC) interrupts may be assigned to the normal world.

Apart from implicit world switches caused by interrupts, there is also a mechanism that allows explicit world switches. This can be done by issuing a *secure monitor call* using the `smc` instruction as part of the TrustZone CPU extension. Once `smc` is called, the CPU switches to *monitor mode*, which is implemented by the system architect and responsible for saving and restoring the CPU context of normal and secure world, respectively.

During the boot process, the available system memory can be assigned to either the secure or the normal world. Any accesses from normal world to secure world memory are prevented by the architecture. However, the secure world can freely access all memory of the normal world. Hence, communication between the two worlds of TrustZone can be done via shared memory, which must be part of the normal world memory space as it is accessible by both worlds. Furthermore, shared memory between worlds must be CPU cache-agnostic, as the secure world and normal world CPU contexts maintain individual caches.

[1] http://memcached.org/

2.2 Genode OS Framework

The Genode OS framework [13] allows building secure special-purpose operating systems based on a micro kernel architecture. While initially supporting the L4 family of micro kernels, recent releases include Genode's independent micro kernel *base-hw*.

Genode provides strong process isolation by imposing a strict organizational structure on processes. By implementing capability-based security [15], Genode enforces security policies. Additionally, Genode enables secure Inter-process Communication (IPC) based on capabilities, while interfaces are defined in a Remote procedure call (RPC)-like fashion: one process announces an RPC server providing functions that can be called by other processes.

TrustZone support is implemented by running the normal world as a Virtual Machine (VM) represented as a process within Genode which is managed by the scheduler just like any other process. The TrustZone VM involves a user-level Virtual Machine Monitor (VMM) running on top of Genode, which controls and manages world switches, and handles hypercalls from the normal world. The amount of normal world memory can be configured by Genode and defines the size of available memory of the VM representing TrustZone's normal world.

2.3 Assumptions and Threat Model

In this paper we assume a *cloud provider* offering a cloud platform to their *users*. The cloud provider operates her infrastructure but is untrusted, she can install arbitrary software in the cloud and eavesdrop and alter the network traffic. As the cloud provider offers a multi-tenant cloud and users may have conflictive interests, users distrust each other mutually.

We assume the attacker of TrApps being the cloud provider, another user or an external party. Thus, we allow certain (basic) physical attacks including boot loader replacements or booting other systems for example. However, we exclude cold boot attacks as TrustZone does not support transparent memory encryption like Intel SGX [16]. Same as for other trusted execution technologies such as Intel SGX, we explicitly exclude Denial of Service (DoS) and any kind of side channel attacks.

3. DESIGN AND IMPLEMENTATION

In this section we give an overview of TrApps and describe its system architecture, which has been designed with our threat model in mind (See Section 2.3). Additionally, we describe the programming model of secure application compartments built on top of the TrApps platform.

With the TrApps platform we target securing applications and services in an untrusted cloud environment in order to allow sensitive data processing in the cloud without trusting the cloud provider. The usage of trusted execution environments (TEEs) allows establishing trust into application components by means of remote attestation, and storage of secrets in the trusted environment after successful verification of the TEE. TrustZone allows the creation of such TEEs, however, it only supports one TEE per platform—the secure world. Therefore, with TrApps we aim at circumventing this limitation which is crucial in a multi-tenant cloud scenario

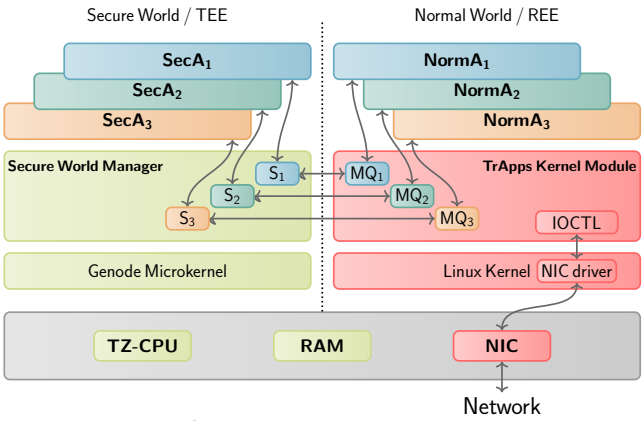

Figure 2: Architecture Details of TrApps.

and allow multiple isolated components inside the secure world of TrustZone. This allows parallel execution of applications of various cloud tenants each with their own secure and strongly isolated compartments on TrustZone platforms. Hence, our notion of secure compartments is very similar to enclaves in the context of Intel SGX [16] applications.

Compared to Intel SGX, the TrApps approach on TrustZone has several advantages: available memory for Intel SGX enclaves is currently limited by the architecture to 128 MB for all enclaves on a platform, exceeding this limit causes high performance penalties [14]. In case of TrustZone an arbitrary share of system memory can be dedicated to secure world, allowing large trusted execution environments. However, ARM TrustZone does not support transparent memory encryption like Intel SGX or AMD SME [17].

3.1 System Architecture

The architecture of TrApps comprises the Genode OS running in secure world and multiplexing the secure world in order to support trusted compartments by various individual cloud customers. In the normal world, we run a standard Linux system which manages the majority of peripherals and especially the network card, and thus, is the only entity with network access.

Large parts of secure TrApps applications are supposed to run in normal world, where they can easily create sockets and receive connections from the internet, while very small parts of their application logic are offloaded as a secure compartment to the secure world—implemented as a Genode child process. This requires the partitioning of applications and leads to a very small TCB, and thus, a small attack surface and a high level of security. The rationale of this *partitioning approach* has been shown in our earlier work already: our SecureKeeper [14] project is a secure Apache ZooKeeper [18] instance for execution on an Intel SGX platform by offloading small parts of its application logic to enclaves.

For our TrApps platform we introduce the notion of Divided Applications (DivAs), that comprise a regular Linux application—called Normal Application (NormA)—with one or multiple trusted compartments called Secure Application (SecA). A NormA is able to upload a SecA binary to secure world and can communicate with it by using the *message queues* provided by TrApps (Section 3.3).

Figure 2 illustrates the architecture of our TrApps platform and shows the Genode OS together with various SecAs running in secure world, and the Linux system with the TrApps kernel module and multiple NormAs running in normal world.

Interaction with TrApps from a user space application of the normal world is done via a Linux kernel module that provides the

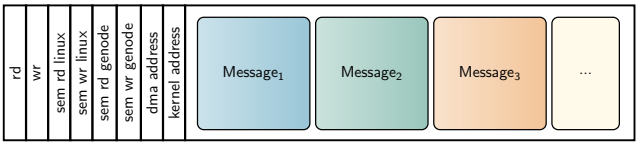

Figure 3: MQueue Memory Range.

above described functionality via `ioctl` operations. This allows multiple DivAs on a platform to use trusted execution capabilities simultaneously and isolated from each other. The TrApps Linux kernel module maintains a distinct connection for each SecA held by a normal world application. For each connection, the interaction with TrApps from the NormA, is done via *ioctl* calls to our Linux kernel module. The TrApps kernel module features a simple interface that allows opening and uploading a SecA binary, and closing a SecA, as well as reading and writing data via the message queue as illustrated in Listing 1.

```
int fops_open(struct inode* inode, struct file* fp);
long ioc_put(put_args_t* args, struct file* fp);
long ioc_read(read_args_t* args, struct file* fp);
long ioc_write(write_args_t* args, struct file* fp);
int fops_close(struct inode* inode, struct file* fp);
```

Listing 1: TrApps Kernel Module Interface.

In the secure world, SecAs run in process isolation as provided by Genode. Their lifecycle and the message queues for cross-world communication are managed by the Secure World Manager (SWM) of TrApps. A SecA can access the SWM and wait for messages from its associated NormA by using RPC calls provided by the SWM for reading and writing data from and to the message queues.

3.2 TrApps in the Cloud Context

The idea of TrApps is to integrate orthogonally into the existing software stack of a cloud provider: during the boot process of a TrustZone hardware platform, secure world is initialised before normal world. After Genode has finished booting in secure world, the boot process will continue initialising the normal world, which is supposed to host the software stack of a cloud provider comprising their custom Linux kernel and management and virtualisation software such as Docker[2] for example. This allows to keep the whole cloud provider's software stack out of the TCB, and thus untrusted, and also allows updates of the cloud provider's software without the need of anew remote attestation of the secure world software. Furthermore, with this approach the business secrets that a cloud provider's kernel or management software may contain are not subject to remote attestation and not required to be published by the cloud provider.

3.3 Cross-world Communication

TrApps establishes a bidirectional cross-world communication channel by allocating a pair of *message queues* (read- and write-queue) per DivA. As depicted in Figure 2, the memory used for message queues resides in normal world memory, thus is accessible from both worlds. We use cache-agnostic DMA memory, to prevent cache conflicts between both worlds since the secure world and normal world CPU contexts maintain distinct caches.

In addition to the messages themselves, message queues hold meta data and state information within their memory range (Figure 3), consisting of 8 fields: Relative read and write pointers `rd`

[2] http://docker.io

and `wr`, that point to the memory location of the next message to be read/written. Pointers to semaphores `sem_*` for synchronization purposes, and the `dma_address`, a pointer to the virtual address of the DMA memory for the message queue. The `kernel_address` points to the same address, but is kernel-virtual.

As a new message is written to the message queue, in order to minimise response time we immediately notify the respective receiver about the new message. This is done using a hypercall when notifying a SecA—strictly speaking the SWM—about a new message, and by injecting an interrupt which is handled by the TrApps kernel module when notifying a NormA about a new message.

3.4 Programming Model

When creating DivAs for the TrApps platform, the developer is supposed to identify critical parts of application logic processing sensitive data, which have to be offloaded to a SecA—we called this *application partitioning approach* in earlier works already [14]. The majority of an application's code base should remain in the NormA in order to keep the TCB of the resulting DivA as small as possible as this increases the security of the application by limiting the attack surface and probability of exploitable vulnerabilities.

In general, a DivA can launch an arbitrary number of different SecAs at any time during its life cycle, and communicate with each of them using our TrApps kernel module by exchanging arbitrary messages on the message queue. The SecA life cycle always starts with opening a connection to the TrApps kernel module and uploading a binary with the respective SecA code. Also, SecAs can be stopped and terminated, releasing their memory at any time during the NormA life cycle, and will be killed when the connection to the kernel module closes (e.g. in case of a crash of the NormA).

4. USE CASE: SECURE MEMCACHED

In order to demonstrate usage of our TrApps platform for securing real-world applications we partitioned *memcached*[3]. Memcached is an excellent example for a protagonist service featuring a simple key-value database which can be used in cloud applications.

In order to secure the data stored inside memcached, we use the "black box"-approach, that we have proposed in previous work for securing Java-based services in an untrusted cloud environment [14]. The rationale of the approach is to re-encrypt the data right after receiving it from the network before inserting it into the data store. By this, the service only sees encrypted data and handles it as a black box. The re-encryption of the data is done in a TEE and comprises decryption of whole network packages (c.f. Transport Layer Security (TLS)) and re-encryption of the individual keys and values after parsing the received message.

Essentially, two different types of encryption are used in the black box approach: while complete data packets are encrypted during transmission via the network (c.f. TLS), only sensitive parts of the requests and responses are encrypted when the data is processed and stored on the server-side.

When porting memcached onto the TrApps platform, we forward received network packages to a SecA. These packages are encrypted for transmission via the network and contain complete memcached requests (e.g. `get()`, `set()`). Inside the SecA we first decrypt the complete message and analyse it in order to identify sensitive data. Next, the sensitive data is encrypted by the SecA which comprises the key and value of a data set in case of memcached. Finally, the request is forwarded to the memcached application which is untrusted and outside of the SecA running as a NormA. As all sensitive data is encrypted, but except for that still

[3]http://memcached.org/

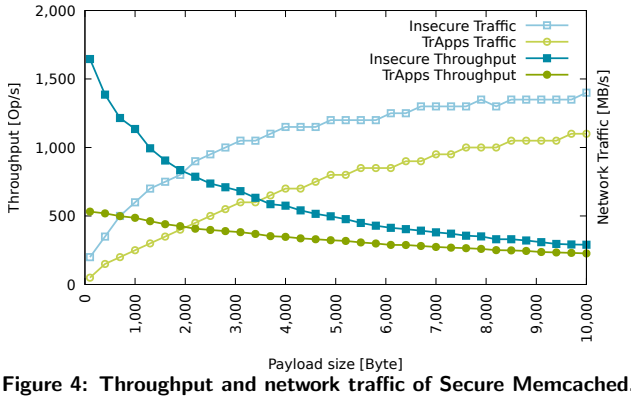

Figure 4: Throughput and network traffic of Secure Memcached.

is a valid memcached request, it can be safely handled by the untrusted code of memcached. This procedure is done after the arrival of each request and before transmitting a response, and essentially replaces the TLS endpoint in order to keep the data encrypted until it reaches the secure execution environment of the SecA.

We implemented the memcached use case application based on the original memcached code where we placed hooks to the SecA after client requests are received from the network before they are processed by memcached, and before responses prepared by memcached are sent to the client via the network. All trusted application logic is implemented as one SecA running in the secure world, which processes the exchanged packets (all requests and responses) as described in the previous paragraph. In order to implement the cryptographic operations we have used the *libsodium*[4] cryptographic library in the secure world. By using TrApps, our secure memcached use case application is able to protect the confidentiality and the integrity of all data stored in memcached at a minimal of the SecA of only 481 Source Line of Code (SLOC).

5. EVALUATION

To measure the performance overhead when using TrApps to secure real-world applications, we partitioned memcached (See Section 4) and measured the resulting performance impact.

The evaluation has been done on the i.MX53-QSB development board by NXP—formerly Freescale—with an open ARM TrustZone implementation. We ran our TrApps platform on the board and hosting the Secure Memcached application with an integrated SecA. Then, we measured the performance of Secure Memcached from another machine connected via the network. In order to measure the performance, we used the benchmark application provided by memcached, called *memslap*, which executes a mixture of `get()` and `set()` operations. This experiment has been repeated for various payload sizes between 0 and 8192 Bytes, executing several thousands of requests for each individual payload size for which memslap calculates the throughput per second.

The results of our experiment are illustrated in Figure 4. As can be seen, the relative overhead decreases for larger payloads, as the overhead is mostly due to the constant world switching time of ARM TrustZone. On average, usage of TrApps imposes an overhead of 36.9% on Secure Memcached, however, for larger payloads the overhead is much lower (only 21.7% for 8k payloads). Equally, the network traffic of Secure Memcached is lower, as less requests are transmitted per second.

[4]https://download.libsodium.org/doc/

6. RELATED WORK

Protecting the rich OS against userland threats: A few systems using TrustZone do not support general-purpose secure compartements, but only aim at protecting the normal world from userland threats. ARMLock proposes a TrustZone architecture aiming at software fault isolation [19], while *SPROBES* [20] instruments the normal world OS kernel to protect against rootkits by implementing normal world kernel introspection mechanisms protected by TrustZone. Azab et al. propose an approach that implements monitoring the normal world kernel from secure world to increase security [21]. It follows a very different goal than TrApps, and while targeting an embedded scenario, does not support trusted execution for user-defined general-purpose components in secure world but only protects several parts of the normal world OSs from the secure world. Finally, *SeCReT* [22] includes special protection of the communication channel between normal and secure world components. However, this is unnecessary in the case of TrApps, as the normal world is completely untrusted, including the untrusted kernel and the cloud provider's software stack. Furthermore, in contrast to the above approaches, TrApps does not aim at protecting the normal world, but secures the application components running in secure world from each other and the cloud provider.

Secure Runtime Systems: Various existing approaches target generic secure runtime platforms. Firstly, Winter et al. [23] propose a trusted computing approach on ARM TrustZone, but opposing our principle of minimising the TCB they use a full-blown Linux kernel in secure world. Nokia proposed On-board credentials [24] which is implemented on M-Shield technology instead of ARM TrustZone and targets mobile and embedded scenarios. In contrast to Winter et al. [23], Luo et al. [25] in fact use a very small secure OS, but their cross-world communication scheme does not offer the benefits and guarantees of our message queue implementation, as it relies on the developer to take care of synchronisation of cross-world message exchange. In 2013, Samsung presented the Knox [26] system which uses ARM TrustZone, but it isolates two environments based on Android from each other and targets mobile usage, for example isolating business and personal use of the same smartphone. The Trusted Language Runtime (TLR) by Santos et al. [27] goes beyond the system by Luo et al. [25], offering a more controlled cross-world communication mechanism, however, while their TCB is much larger than ours, they do not support multi-tenant isolated cross-world communication. With the advent of the Intel SGX technology, Baumann et al. proposed a system that allows the execution of unchanged legacy applications with their Haven system [28]. While the benefits of running legacy applications unchanged in a secure environment may sound appealing, the system suffers from a very large TCB due to the included library OS and other dependencies inside the secure container which decreases the level of security and results in a large attack surface. Similarly, SCONE [29] proposed by Arnautov et al., that allows running Docker containers in Intel SGX enclaves, comprises a significantly sized TCB. Brito et al. [30] proposed a secure image processing system in a cloud scenario based on ARM TrustZone. In contrast to TrApps, their approach does not allow general purpose execution and independent multi-tenant cross-world communication. Rubinov et al. [31] propose an approach which is also similar to TrApps, but targeting an embedded environment with Android running in normal world. This in turn requires all normal world applications to be written in Java and interact with the secure component using Java Native Interface (JNI). TrApps is more generic in this regard and does not limit normal world applications to Java. Also the TCB of their secure OS comprising SierraTEE[5] is about two times larger compared to TrApps's Genode. Finally, Lee et al. proposed a TrustZone-based platform for securing small applets in personal home routers [32]. Apart from targeting cloud scenarios instead of home routers, TrApps features high resource efficiency by dynamically scheduling secure and normal world instead of pinning CPU cores statically to one of the two worlds.

While most works in ARM TrustZone target embedded and mobile scenarios, to the best of our knowledge, TrApps is the first platform featuring a low TCB while supporting efficient general-purpose execution of secure compartments with strong isolation in a multi-tenant cloud scenario without trusting the cloud provider.

Trusted Hypervisors: Existing works comprise many different forms of trusted hyervisors (most of them) based on ARM TrustZone targeting various goals. *Terra* [33] is one of the earlier trusted hypervisors and allows the creation of trusted VMs which in turn contain a whole trusted OS leading to an enormous TCB. McCune et al. proposed *TrustVisor* [34] that allows the trusted execution of small pieces of sensitive code. One step further advances *CloudVisor* [35] in a cloud scenario, using nested-virtualization to put another tiny hypervisor underneath the commodity hypervisor, but the system focuses on securing complete VMs instead of small components only, again leading to a large attack vector and TCB. Not in the cloud context but in the environment of a untrusted OS, *InkTag* [36] aims at the integration of high-assurance processes using virtualization technology. Recently, Jang et al. [37] proposed *PrivateZone*, a system very similar to our TrApps. While they also support small trusted components, their system comprises a larger TCB than our TrApps and requires hardware supporting the ARM virtualisation technology. Furthermore, in contrast to the above trusted hypervisors, the goal of TrApps is the transparent integration into the existing cloud software stack, and support for small manageable trusted components. As described in our design section (See Section 3), our goal is to enhance the existing software stack of the cloud provider, even allowing updates to the untrusted software stack that do not affect the secure world, instead of replacing the cloud provider's software stack.

7. CONCLUSION

Tackling the security and privacy issues of modern cloud computing environments, we have shown TrApps, a secure TrustZone-based cloud runtime for partitioned applications. While the partitioning approach minimises the TCB, our architecture allows multi-tenant execution of various secure components owned by different tenants in parallel. To the best of our knowledge TrApps is the first ARM-based platform featuring trusted execution tailored for untrusted clouds, and incorporating highly efficient cross-world communication at the same time. Our evaluation shows the feasibility of our approach and a low performance overhead of only 36.9% in our use case application memcached.

Acknowledgments

This project received funding from the European Union's Horizon 2020 research and innovation programme under the SERECA project (Grand No. 645011).

References

[1] IDC, *Worldwide Cloud IT Infrastructure Market Growth Expected to Accelerate [...]* http://www.idc.com/getdoc.jsp?containerId=prUS25576415. 2015.

[5] https://www.sierraware.com/open-source-ARM-TrustZone.html

[2] K. R. Jayaram, D. Safford, U. Sharma, V. Naik, D. Pendarakis, and S. Tao, *Trustworthy Geographically Fenced Hybrid Clouds*, in *15th International Middleware Conference*, Middleware, 2014.

[3] S. Pearson and A. Benameur, *Privacy, Security and Trust Issues Arising from Cloud Computing*, CloudCom, 2010.

[4] ARM, *Building a Secure System using TrustZone Technology*. 2009.

[5] *Windows Server on ARM: It's happening*, http://www.zdnet.com/article/windows-server-on-arm-its-happening/, 2017.

[6] *Microsoft Pledges to Use ARM Server Chips, Threatening Intel's Dominance*, https://www.bloomberg.com/news/articles/2017-03-08/microsoft-pledges-to-use-arm-server-chips-threatening-intel-s-dominance, 2017.

[7] *AMD Opteron™A-Series Processors*, http://www.amd.com/en-gb/products/server/opteron-a-series, 2016.

[8] *Scaleway*, https://www.scaleway.com/features/, 2016.

[9] R. Delgado, *ARM-based servers: The next evolution of the cloud?* http://www.cloudcomputing-news.net/news/2015/apr/17/arm-based-servers-next-evolution-cloud/, 2015.

[10] Y. Sverdlik, *French Web Host Builds ARM-Powered Bare Metal Cloud*, http://www.datacenterknowledge.com/archives/2014/11/19/french-web-host-builds-bare-metal-arm-server-cloud/, 2014.

[11] A. Web, S. Overview, and S. P. May, "Amazon Web Services: Overview of Security Processes," Tech. Rep., 2011.

[12] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, *Flicker: an execution infrastructure for tcb minimization*, in *EuroSys*, 2008.

[13] *Genode Operating System Framework*, https://genode.org/documentation/general-overview.

[14] S. Brenner, C. Wulf, M. Lorenz, N. Weichbrodt, D. Goltzsche, C. Fetzer, P. Pietzuch, and R. Kapitza, *SecureKeeper: Confidential ZooKeeper using Intel SGX*, Middleware, 2016.

[15] J. B. Dennis and E. C. Van Horn, *Programming semantics for multiprogrammed computations*, *Communications of the ACM*, vol. 9, no. 3, 1966.

[16] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, *Innovative Instructions and Software Model for Isolated Execution*, in *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP, 2013.

[17] T. W. David Kaplan Jeremy Powell, "AMD Memory Encryption," Tech. Rep., 2016.

[18] P. Hunt, M. Konar, F. Junqueira, and B. Reed, *ZooKeeper: wait-free coordination for internet-scale systems*, in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, 2010.

[19] Y. Zhou, X. Wang, Y. Chen, and Z. Wang, *ARMlock*, in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*, 2014.

[20] X. Ge, H. Vijayakumar, and T. Jaeger, *Sprobes: Enforcing kernel code integrity on the trustzone architecture*, *Mobile Security Technologies 2014 Workshop*, 2014.

[21] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, *Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world*, in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, 2014.

[22] J. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, *SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment*, in *NDSS*, 2015.

[23] J. Winter, *Trusted computing building blocks for embedded linux-based ARM trustzone platforms*, *Proceedings of the 3rd ACM workshop on Scalable trusted computing - STC '08*, 2008.

[24] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala, *On-board credentials with open provisioning*, in *ASIACCS*, 2009.

[25] J. Luo, C. Jiang, and X. Yang, *Design and implementation of security OS based on TrustZone*, in *ICEMI*, 2013.

[26] Samsung Electronics, *An Overview of Samsung KNOX™*, no. June, 2013.

[27] N. Santos, H. Raj, S. Saroiu, and A. Wolman, *Using ARM TrustZone to Build a Trusted Language Runtime for Mobile Applications*, in *ASPLOS*, 2014.

[28] A. Baumann, M. Peinado, and G. Hunt, *Shielding applications from an untrusted cloud with Haven*, in *OSDI*, 2014.

[29] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, *et al.*, *SCONE: Secure linux containers with Intel SGX*, 12th USENIX Symp. Operating Systems Design and Implementation, 2016.

[30] T. Brito, N. O. Duarte, and N. Santos, *ARM TrustZone for Secure Image Processing on the Cloud*, in *2016 IEEE 35th Symposium on Reliable Distributed Systems Workshops*, 2016.

[31] K. Rubinov, L. Rosculete, T. Mitra, and A. Roychoudhury, *Automated partitioning of android applications for trusted execution environments*, in *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, 2016.

[32] S.-s. Lee, H. Shi, K. Tan, Y. Liu, S. Lee, and Y. Cui, *Smart and Secure: Preserving Privacy in Untrusted Home Routers*, in *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems - APSys '16*, 2016.

[33] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, *Terra: A virtual machine-based platform for trusted computing*, in *SOSP*, vol. 37, 2003.

[34] J. M. J. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, *TrustVisor: Efficient TCB reduction and attestation*, in *Security and Privacy ...*, 2010.

[35] F. Zhang, J. Chen, H. Chen, and B. Zang, *CloudVisor*, in *SOSP*, 2011.

[36] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel, *InkTag: Secure Applications on an Untrusted Operating System.* In *ASPLOS*, 2013.

[37] J. Jang, C. Choi, J. Lee, N. Kwak, S. Lee, Y. Choi, and B. Kang, *PrivateZone: Providing a Private Execution Environment using ARM TrustZone*, *IEEE Transactions on Dependable and Secure Computing*, 2016.