PROTECTING PRIVATE DATA IN THE CLOUD

Lars Rasmusson and Mudassar Aslam
Swedish Institute of Computer Science, Kista, Sweden

Keywords: Cloud Computing, Virtualization, Trusted Computing, Binary Translation.

Abstract: Companies that process business critical and secret data are reluctant to use utility and cloud computing for the risk that their data gets stolen by rogue system administrators at the hosting company. We describe a system organization that prevents host administrators from directly accessing or installing eaves-dropping software on the machine that holds the client's valuable data. Clients are monitored via machine code probes that are inlined into the clients' programs at runtime. The system enables the cloud provider to install and remove software probes into the machine code without stopping the client's program, and it prevents the provider from installing probes not granted by the client.

1 INTRODUCTION

In cloud computing, computing resources are provided by one or more organizations, and are used by other organizations. This is usually the only shared interest between the providers and the clients, and that distinguishes cloud computing from regular distributed computing within a company. The limited shared interest in cloud computing is the fundamental reason for the client to distrust the cloud provider, as well as for the provider to distrust the client.

However, cloud computing's large potential for more efficient computing, in both technical and economical sense, is a strong reason to try to overcome this mutual distrust.

Research in security for cloud computing has focused on isolation of the clients' computation inside virtual machines (VM) on various levels to protect the cloud infrastructure from attacks by clients, and to protect clients from attacks from each other (Christodorescu et al., 2009). This problem is now routinely solved with virtualization.

Current solutions have not done enough to address the issue of how to protect the client from the provider. The lack of protection against the provider has prevented the adoption of cloud computing for clients that have private, business critical or confidential data or algorithms i.e. the financial, government, health, pharma, and movie sectors. The risk clients face of having their data or programs leaked or stolen by the provider, either deliberately, by mistake, or by a disgruntled or black-mailed employee, has so far been considered too large. In (Kuttikrishnan, 2011) several studies of cloud adoption are summaprized and reporting as one obstacle that cloud adoption "involves developing trust and overcoming the fear of change and loss of control over data and processes."

While the risk from attacks by the provider are not unique to the cloud (they are already present in hosted environments), they are increased by the more short-lived and anonymous relationships between the parties in clouds with multiple providers and automatic migration of data and computation.

We propose a novel solution to the problem of mutual distrust. It is based on using trusted computing to prevent the cloud provider from installing eaves-dropping software on the platform, on using a binary translation framework to let the provider install software probes into the clients software, and on an initial negotiation phase to determines which probes may actually be installed on the platform to satisfy both provider and client. So, while the provider is locked out from the client's VM she is provided with another tool that enables her monitor the client's VM for malicious behavior without compromising the client's need for data protection.

The next section describes the system architecture. Section 3 explains the current implementation. It is followed by an analysis of the security of the system in section 4. The paper is concluded in section 5 with related work and the contributions made in this system.

Rasmusson L. and Aslam M..
PROTECTING PRIVATE DATA IN THE CLOUD.
DOI: 10.5220/0003895800050012
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 5-12
ISBN: 978-989-8565-05-1

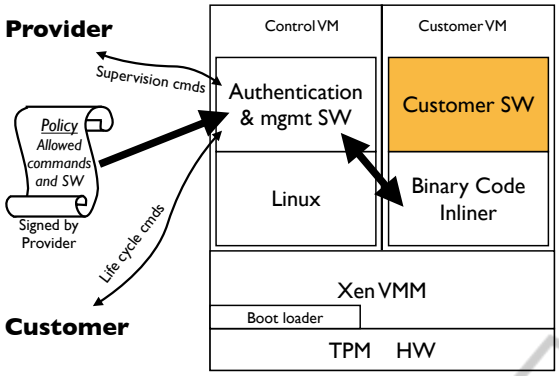

Figure 1: Computer resource.

2 ARCHITECTURE

Our goal is to have a runtime system for clouds where the provider can check for e.g. viruses/botnets, unlicenced software, etc., while at the same time verifiably show to a remote client that the provider has no posibility to extract undue information from the client's virtual machine.

2.1 Resources

The cloud infrastructure is composed of computational resources, a resource management system, and a PKI with certifying authorities. The computational resources are the computers, networks and storage that the provider is offering to clients.

Figure 1 shows a computer in the cloud that is running a virtualization layer, management software, a machine code inliner and the client's software.

The provider configures the computers by setting them up for trusted boot, installs the necessary software on them (the "trusted software stack"), and registers their availability to the resource management system. See step 1 and 2 in figure 2. The software that is installed on the computers is responsible for only permitting authorized commands to be executed. In particular, it must prevent the provider from gaining root access or full control over the machine. Trusted third parties may be required to audit the trusted software for bugs and to make sure that the machines are guarded against physically attacks or modifications.

2.2 Software Probes

Instead of root access or physical access, the cloud administrators are provided with a software tool that enables them to install software probes into the client's running program. The probes may check for network attacks made by the client, for undue hardware access, or perform software license enforcement by detecting execution of licensed code. Since they are in software, probes can be arbitrarily complex, but to prevent probes from leaking information to the provider, only safe probes authorized by the client must be installed. This is assured by the trusted management software.

A probe is monitoring the CPU state and one or several memory cells or data structures inside the client's VM, in the kernel or in the application programs. (Constructing probes that look into application data structures will require access to symbol tables or debug information.) When the probe detects an action, it can update counters or other data structures in a protected memory area which is not adressable by the client software. An appropriate probe may thus detect execution patterns that the provider has requested to scan for. How the provider gets the information from the probe is described in section 2.4.

Probes are built into the code at the time that it is Jut-In-Time (JIT, immediately before execution) translated from its original form to the code to be actually executed. For each machine instruction in the original code a corresponding function that emulates the instruction on a virtual CPU exists. To create new machine code for a sequence of instructions in the original program, a sequence of function calls are created in the LLVM (Lattner and Adve, 2004) intermedate representation (IR). These calls are then linked (at run time) with in-memory IR code for the functions, inlined, constant folded, and optimized, before new machine code is emitted. A cache can be used to avoid redoing this for loops.

Concretely, probes are implemented as modifications/extensions to the functions that emulate machine instructions. For instance, they may count the number of times a certain memory address is accessed by the ADD instruction. While this check at first will appear at ever place the ADD instruction occurs, the optimization step will in many cases be able to remove this check based on constant information known at run-time.

2.3 Procurement

A client wishing to run a private computation on a cloud resource contacts the resource management system to get access to a free resource. See steps 3 and 4 in figure 2. In the negotiation with the resource management system the client can negotiate an acceptable set of probes that may be installed on the machine, but if no agreement can be made, no re-

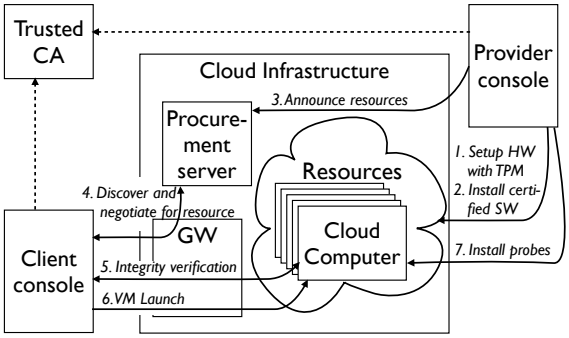

Figure 2: Startup, procurement and launch.

source can be allocated. Once details about payment have been cleared, the resource management system issues a resource definition file containing a policy description.

2.4 Resource and Policy Description File

The resource description contains a policy (see figure 1) consists of a list of commands that may be executed, and the identity of the one who may trigger them. For instance the client "C1" may execute the commands "run" and "destroy" to start and remove his virtual machine. The provider "cloudP" may execute the command "installVirusProbe", "removeVirusProbe", and "getVirusProbeReading". The meaning of these commands are also defined by the policy file, as the commands are defined in program files whose checksums are stored in the policy file.

The resource description file also contains the checksums that the trusted software stack will generate in the Trusted Platform Module (TPM), the public key of the TPM on the physical computer, and the network address of the resource's Launch Manager.

The file is signed by the provider, and the signature is verified by the resource computer during the VM Launch step.

2.5 VM Launch

To start using the resource, the client contacts the Launch Manager and asks via a challenge response protocol for proof that the machine has a genuine TPM on a certified platform. (See step 5 in figure 2.) The Launch Manager asks the TPM chip to provide the necessary signed certificates and forwards them to the client who verifies them. (Essentially, a TPM chip contains a protected private key that the software cannot access. The chip maker has supplied a certificate stating that a chip with the associated public key was made in their factory. The client asks (via the Launch Manager) the TPM to sign a partly random string. The response is checked to see that signatures and certificates match.)

Once satisfied that the platform is genuine, the next step for the client is to verify that the correct trusted software stack is installed. For that, the client requests a signed dump of the checksum registers in the TPM. The TPM checksum registers are updated during system startup, from BIOS (on machine initialization code) start and on, and can only be physically reset by rebooting the machine. Since all the trusted software checksums all trusted software it loads in, the checksum registers will be in a specific state, depending on which software that has been loaded.

Once the client has verified that the checksum registers are in the state described in the resource file (and potentially checked with a third party that this software stack is indeed to be trusted), it can upload the private data and code to the cloud, knowing that control of the machine can only be done via the narrow command interface defined in the policy description. (See step 6 in figure 2.)

2.6 Probe Inlining

The software probes are inlined into the client's machine code with a Binary Code Inliner that runs underneath the client's program, essentially as a JIT compiling machine emulator. (See step 7 in figure 2.)

The main reason why the probes are JITed in rather than inserted as jumps is to enable complex probes to be mixed into the code everywhere without causing excessive branching, since that is expensive on modern architectures with deep pipelines, prefetching, and branch prediction. Another reason is that the JITed probes can be reduced in size by JIT optimization using local knowledge of the machine code.

The Inliner disassembles machine instructions up until a branch and generates new code that emulates the instructions in another area of the memory. The emulation code may be extended with probe code to do more things than the normal code, such as maintain counters, track state changes or information flows in the program, detect the occurrence of certain data in the memory, etc.

The Inliner optimizes the code to remove constant checks, inline function calls, reuse partial computation values, and merge multiple updates to the same memory location into one. Thus, JITed monitoring code that does a lot of the same checks all the time can be greatly reduced in size compared to running the full monitoring code each time.

The JITed code is stored into a code cache, and once an already JITed entry point is called again, the cached code is used. Since loopy code will achieve high cache hit rate, the expensive JIT step will be amortized. The additional cost of probe processing cannot be amortized since it is new functionality that is added to the system. The probing cost should thus be considered as a cost for protection against a omnipotent but rogue system administrator.

Turning probes on and off amounts to using different sets of emulation code and using separate code caches or clearing the code cache when probes are toggled.

3 IMPLEMENTATION

3.1 Base System

Our current implementation is running on a dual hexcore Intel Xeon based Ericsson GEP3 board. The GEP3 has a TPM soldered onto the board (and not replaceable TPM module). It has a special BIOS that checksums the BIOS, firmware and boot loader before it passes control to the boot loader.

The boot loader is a version of Trusted Grub (Trusted Grub, 2012) which is configured to checksum and load a Xen VMM (Virtual Machine Monitor, a software layer that provides virtualization functionality and isolation), a Linux kernel and a ramdisk, before it launches Xen. Xen then launches the Linux kernel in a virtual domain, the Control VM.

The Linux kernel creates checksums of all the binaries and config files that are loaded into the system and keeps the checksums in a loaded files list that can later be checked by a client. Since varying load order would change and aggregated checksum, the checksums are not stored in the TPM checksum registers. Instead, the client has to check the kernel's list to make sure that no unauthorized program has been launched before.

3.2 Launch Manager

After the boot process has finished, the Launch Manager is started. It is written in Java and uses the IAIK java Trusted Software Stack (IAIK java Trusted Software Stack, 2012) to communicate with the TPM chip on the board. The Launch Manager is run as a user level process and has no system level access. Access to the TPM device and Xen commands are provided via group permissions and setuid:ed scripts that double check the parameters and access rights before execution.

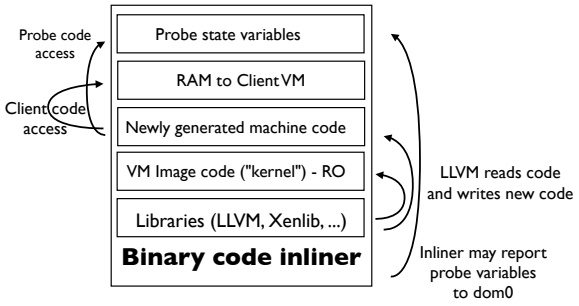

Figure 3: Memory regions inside the client's Xen domain.

All communication with the Launch Manager is made over TLS (Transport Layer Security communication protocol) and uses certificates to establish identities of all parties. The launch manager receives the resource description file from the client and starts to accept and execute the commands that are listed in the resource definition file, provided that all file checksums are correct. The commands to manipulate the virtual machines are executed with Java's `Runtime.exec(cmd)`.

During the launch phase, the Launch Manager sends back a *bind key* to the client, which is a public key inseparably tied (or bound) to the machine's TPM, which the client uses to encrypt its secret data. The encrypted data can then only be decrypted by the very TPM on which the bind key was created.

The Launch Manager receives an encrypted tar file containing the VM image and other necessary files from the client which it decrypts, unpacks and uses to create the virtual machine. The client's code is not set to run directly into another Xen domain. Instead the Launch manager creates a Customer VM domain with a Binary Code Inliner kernel, and then sends over the client's code and data to the Inliner to execute, together with the software probe files to be installed.

3.3 Binary Code Inliner

The Binary Code Inliner is based on the Mini-OS Xen software. It is a miniature kernel that can be used for driver domains. The Binary Code Inliner contains the LLVM libraries necessary for disassembling, building, optimizing and JIT compiling native machine code.

The Inliner communicates with the Launch Manager over the Xenbus message passing interface, and via a shared memory page for fast transfer of files between the domains.

The Inliner uses different memory regions for the client's VM's RAM and the memory in which the probes store their state information. (See figure 3.)

Since the pure emulation code never will access the probes memory area, there is no machine instruction the client can use to access the probe memory. But probe code that is mixed in with the emulation code can access the probe memory as normal memory, without needing changing the memory context or using traps. Therefore the JIT compilation can produce very lightweight and ubiquitous software probes.

The Inliner is written in C++ and C to interface with LLVM and Mini-OS and is about 2MB in size, mostly due to the large libraries that it includes. It currently only supports a subset of the full X86 instruction set, and more sophisticated optimization such as optimization across multiple code blocks is not yet implemented.

4 SECURITY ANALYSIS

4.1 Locking Out the Provider with TC

A rogue system admin may try to eaves-drop and dump data in the clients' virtual machines to sell, use, or to keep "for a rainy day". Trusted Computing (TC) with hardware chip (TPM) is used to convince the client that no eaves-dropping software is installed. Trusted boot lets the client verify that the system was started up in a known state, but it is the loaded trusted software that correctly must prevent remote login and other undue access, as Trusted Computing (TC) cannot protect against software bugs in that software.

Some attacks can be detected by TC, i.e. if an intrusion has triggered loading of an unexpected binary it will be listed in the kernel's loaded files list, but in general TPM really relies on having a locked down trusted software stack. In the described system the trusted software stack consists of the BIOS, firmware, boot loader, Xen, the Linux kernel, the OS configuration files, and the Launch Manager and its runtime system (Java). The need for trust is reduced by layering. The trust in the Launch Manager does not have to be complete, as it is partly isolated with OS techniques. The Launch Manager runs as a low privileged user process that is not allowed to arbitrarily access VM memory or other `xm` or `xenstore` commands. In the current implementation the Launch Manager is responsible for decrypting the client's VM image. This task should be moved into the kernel, to remove the ability for a subverted Launch Manager or other process in the Control VM to leak decrypted uploaded client data.

The Launch Manager is written in Java, but may also execute commands with `Runtime.exec(cmd)`. The commands that it executes are taken from a policy file that was signed by the provider, but the client has full access to what the commands do and their binary code. Because their checksums are also in the policy file the management commands cannot be modified neither by the client nor by the provider. To prevent the provider from putting malicious commands into the policy file, the client needs an external auditor to vet the code. The cost of the vetting will spur the adoption of commonly used and well understood software and policies, or increase demand for formal computer-checked security proofs.

TC does not guard against physical attacks except through obscurity and complexity, and in theory TC can be circumvented by tricking the TPM to be reset by physically manipulating its input pins. After a reset, arbitrary values can be loaded into the checksum registers, hiding the presence of malicious software. The data in RAM capsules and hard disk is not encrypted by the TPM. To prevent the hardware from being physically extracted from the machine the machines must be guarded, and policies must be enforced to for instance never let a single unmonitored system admin manipulate the machines. Since the TPM does not encrypt the data in memory or the computation, it does not protect against forced access to the hardware (e.g. due to a warrant).

TC is vulnerable to TPM chipmakers that issue false TPM certificates or put in backdoors in the chip to extract the private keys, as it enables man-in-the-middle attacks on the supposedly authenticated communication between the client and the TPM.

4.2 Locking Out the Client

The system relies on standard techniques for locking out the client from attack on infrastructure and other clients: the clients' software is executed in low privileged virtual machines whose access to hardware goes via a virtual device layer that enforces limits on network access, I/O and memory capacity consumption. The standard techniques are not sufficient for detecting and blocking botnets and other malicious software that may thrive in the cloud. A cloud client who is hacked may be made to run software that causes great damage on others. To detect such activity, providers usually have the ability to look deep into their clients' processes and VMs. Unfortunately, the same ability also opens up to data-stealing by rogue system admins. Instead of giving direct and full access to the admins, the cloud architecture provides a means for them to inject pre-defined software probes that probe for characteristic patterns or activities of botnets, viruses, etc., and act or alert only when something is detected.

Malicious software in the client may try to circumvent the probes by overwriting the memory area where they store state information. They are unable to do so because each instruction in converted into new code that cannot explicitly address the probes' private memory area.

An attacker may try to disassemble the code to detect the probes, but the machine instructions that the client will see are read from a different part of the memory than the actual machine code, because the memory is virtualized. A client disassembling the instruction at its PC register will disassemble its virtual memory, because the PC (and the other registers) is virtualized and points into the original code, not the newly generated code with probes.

The layered design is helpful if there are bugs in the Binary Code Inliner and LLVM. In the worst case, an attacker may construct an instruction sequence that tricks LLVM's JIT compiler to produce JITed code that breaks out of the Inliner. The risk is minimal, but if that happens, the program is still isolated in the Customer VM, which is an unprivileged Xen domain, and it is therefore prevented from attacking the Launch Manager in the Control VM.

5 RELATED WORK AND CONTRIBUTIONS

5.1 Related Work

Terra (Garfinkel et al., 2003) describes a trusted virtual machine monitor that uses Trusted Computing (Trusted Computing Group, 2012) to prevent the machine owner (provider in our terminology) from accessing the contents of a virtual machine to protect its confidentiality. Trusted Computing for cloud computing is first suggested as a possible solution by (Santos et al., 2009), which so far only contains early work on protocols for launching and migrating virtual machines protected by Trusted Computing. TVDc (Berger et al., 2008) let the client check the integrity of a Xen dom0 and the user domains via virtual TPMs, but the TVDc provider is neither locked out nor given an alternative means for controlling the client.

Machine code to machine code JIT translation has been used to speed up execution in the Dynamo system (Bala et al., 2000). It is not used to insert additional monitoring code into the generated code. PIN (Reddi et al., 2004) and DynamoRIO (Bruening, 2004) are tools that let a user write probes that are dynamically injected into an application level program, not a full OS. The probes are compiled before injection, and no further optimization is done at runtime. Their APIs operate on instruction level and basic block level.

LLVM (Lattner and Adve, 2004) is a compiler framework that has an intermediate code representation that enables programmatical modification, optimization and JIT compilation at runtime. It is used in the Binary Code Inliner to produce optimized code for each translated basic block on the fly.

Xen (Barham et al., 2003) uses hardware based protection to isolate virtual machines and separate memory areas. Xen can only monitor a few things, such as what goes in and out via the virtual devices, memory and CPU utilization rate.

PINOS (Bungale and Luk, 2007) instruments full OS by running PIN inside a Xen VM. Its uses the same low-level API as PIN. XenAccess (Payne et al., 2007) is a tool that permits looking/sampling into the memory of a running virtual machine by following entries and pointers in the symbol tables and data structures in the VMs memory. It does not enforce limits, and it does not monitor memory or registers continuously, but only when an external monitoring program is explicitly started from outside of the VM.

Van Dijk, et al. (Van Dijk and Juels, 2010) argue that software alone, such as fully homomorphic encryption or secure multiparty computation is not enough to provide privacy-preserving cloud computing. They point out Trusted Computing's weakness with respect to hardware attacks. Parno points out the problem of getting the TPM key to the client (Parno, 2008). We argue that hardware attacks may be countered with (human) guards, and that a public key infrastructure can prove to a user that the keys are genuine.

While our system uses policies to inline specialized machine code into the client's binary, it has still been compared to iRODS (Wan et al., 2009), a policy based system for controling cloud clients' access to data. While iRODS enforces policies at specific enforement points, our system may insert control policies anywhere into the client's binary. It may also excercise control over the execution, not just over access to data. While the iRODS system is more mature, our system is yet but a (versatile) tool for implementing and executing fine grained policies. Which those policies will be, what they will measure and how to implement them concretely still remains to be found out.

While some recent surveys of multi-tenant clouds (Rodero-Merino et al., 2012) still ignore the risk of a rogue provider, Vaquero et. al. (Vaquero et al., 2011) list what they consider the main threats to IaaS clouds,

including malicious insiders, the risk for data loss and leakage, and account or service hijacking. They survey current work addressing these threats, i.e. (Constandache et al., 2008; Descher et al., 2009; Baldwin et al., 2009). With respect to the threat of insiders, the listed solutions only address the task of locking out the provider. The main purpose of our solution is to provide an environment that restores some power to the provider by offering her a versatile tool to, in a controlled way, inspect the client at run time.

5.2 Contribution

Our contribution is to develop a cloud architecture and proof-of-concept prototype implementation that (under the Trusted Computing assumptions) can prove to the client that no eaves-dropping software is or can be installed on the resource computer.

The cloud administrators' lost supervisory powers are partially compensated with a generic tool (probe inlining) in which invasive checks and enforcement of the client's VM for detection and thwarting of dangerous activities can be implemented. The plan is that probes for botnets, viruses, and cloud infrastructure attacks will be implemented in this framework, but specific probes have not been the focus of our current work. Probes are installed on a fine-grained per-instruction level meaning they are always on, and the probes cannot be accessed or circumvented in any way by the client's software.

The idea to only let cloud system admins install software probes into the clients' VMs instead of having full access to a machine is new. Our approach for doing that, JIT machine code translation is not new, but our approach of translating it into an intermediary compilation format that enables mixing in high level probing functions, and to optimize entire basic blocks rather than individual instructions, is to our knowledge not found elsewhere.

We also have contributed a TPM based architecture in which policies that determine the extent of probing first are negotiated between provider and client, and later provably enforced by the cloud software.

We have presented a security analysis to highlight the strengths and limitations of the security provided by this cloud architecture.

The current work consists of the architecture for the runtime system for the probes, and does not yet provide any specific language for declaratively defining probes. Defining concrete probes for actual threats remains an issue for future work.

ACKNOWLEDGEMENTS

We wish to thank Christian Gehrmann at SICS and Andrs Mhes and Rolf Blom at Ericsson for thoughtful comments and critique. Special thanks to Andrs for providing, configuring and thoroughly locking down the GEP3 board and Xen. We also wish to thank the anonymous reviewers for their helpful comments.

REFERENCES

Bala, V., Duesterwald, E., and Banerjia, S. (2000). Dynamo: a transparent dynamic optimization system. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, PLDI '00, pages 1–12, New York, NY, USA. ACM. http://doi.acm.org/10.1145/349299.349303.

Baldwin, A., Dalton, C., Shiu, S., Kostienko, K., and Rajpoot, Q. (2009). Providing secure services for a virtual infrastructure. *SIGOPS Oper. Syst. Rev.*, 43:44–51. http://doi.acm.org/10.1145/1496909.1496919.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37:164–177. http://doi.acm.org/10.1145/1165389.945462.

Berger, S., Cáceres, R., Pendarakis, D., Sailer, R., Valdez, E., Perez, R., Schildhauer, W., and Srinivasan, D. (2008). TVDc: managing security in the trusted virtual datacenter. *SIGOPS Oper. Syst. Rev.*, 42:40–47. http://dx.doi.org/10.1145/1341312.1341321.

Bruening, D. L. (2004). *Efficient, transparent, and comprehensive runtime code manipulation*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.7639.

Bungale, P. P. and Luk, C.-K. (2007). PinOS: A programmable framework for whole-system dynamic instrumentation. In *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, pages 137–147, New York, NY, USA. ACM. http://doi.acm.org/10.1145/1254810.1254830.

Christodorescu, M., Sailer, R., Schales, D. L., Sgandurra, D., and Zamboni, D. (2009). Cloud security is not (just) virtualization security: a short paper. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 97–102, New York, NY, USA. ACM. http://doi.acm.org/10.1145/1655008.1655022.

Constandache, I., Yumerefendi, A., and Chase, J. (2008). Secure control of portable images in a virtual computing utility. In *Proceedings of the 1st ACM workshop on Virtual machine security*, VMSec '08, pages 1–8, New York, NY, USA. ACM. http://doi.acm.org/10.1145/1456482.1456484.

Descher, M., Masser, P., Feilhauer, T., Tjoa, A. M., and Huemer, D. (2009). Retaining data control to the

client in infrastructure clouds. *Availability, Reliability and Security, International Conference on*, 0:9–16. http://doi.ieeecomputersociety.org/10.1109/ARES.2009.78.

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., and Boneh, D. (2003). Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 193–206, New York, NY, USA. ACM. http://doi.acm.org/10.1145/945445.945464.

IAIK java Trusted Software Stack (2012). http://trustedjava.sourceforge.net.

Kuttikrishnan, D. (2011). Cloud Computing: Slow Adoption Rates, Current Obstacles. http://www.datamation.com/cloud-computing/cloud-computing-slow-adoption-rates-current-obstacles.html.

Lattner, C. and Adve, V. (2004). LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–, Washington, DC, USA. IEEE Computer Society. http://llvm.org/pubs/2003-09-30-LifelongOptimizationTR.pdf.

Parno, B. (2008). Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd Conference on Hot Topics in Security*, pages 9:1–9:6, Berkeley, CA, USA. USENIX Association. http://www.usenix.org/event/hotsec08/tech/full_papers/parno/parno.pdf.

Payne, B. D., Carbone, M., and Lee, W. (2007). Secure and Flexible Monitoring of Virtual Machines. *Computer Security Applications Conference, Annual*, 0:385–397. http://doi.ieeecomputersociety.org/10.1109/ACSAC.2007.10.

Reddi, V. J., Settle, A., Connors, D. A., and Cohn, R. S. (2004). PIN: A Binary Instrumentation Tool for Computer Architecture Research and Education. In *Proceedings of the 2004 workshop on Computer Architecture Education: held in conjunction with the 31st International Symposium on Computer Architecture*, WCAE '04, New York, NY, USA. ACM. http://doi.acm.org/10.1145/1275571.1275600.

Rodero-Merino, L., Vaquero, L. M., Caron, E., Muresan, A., and Desprez, F. (2012). Building safe paas clouds: A survey on security in multitenant software platforms. *Computers & Security*, 31(1):96 – 108. http://dx.doi.org/10.1016/j.cose.2011.10.006.

Santos, N., Gummadi, K. P., and Rodrigues, R. (2009). Towards Trusted Cloud Computing. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, Berkeley, CA, USA. USENIX Association. http://portal.acm.org/citation.cfm?id=1855533.1855536.

Trusted Computing Group (2012). http://www.trustedcomputinggroup.org.

Trusted Grub (2012). http://projects.sirrix.com/trac/trustedgrub.

Van Dijk, M. and Juels, A. (2010). On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of the 5th USENIX conference on Hot topics in security*, HotSec'10, pages 1–8, Berkeley, CA, USA. USENIX Association. http://www.usenix.org/events/hotsec10/tech/full_papers/vanDijk.pdf.

Vaquero, L. M., Rodero-Merino, L., and Morn, D. (2011). Locking the sky: a survey on IaaS cloud security. *Computing*, 91:93–118. http://dx.doi.org/10.1007/s00607-010-0140-x.

Wan, M., Moore, R., and Rajasekar, A. (2009). Integration of cloud storage with data grids. *Computing*. https://www.irods.org/pubs/DICE_icvci3_mainpaper_pub-0910.pdf.