CertiKOS: A Certified Kernel for Secure Cloud Computing

Liang Gu, Alexander Vaynberg, Bryan Ford, Zhong Shao, David Costanzo
Yale University
{liang.gu, alexander.vaynberg, bryan.ford, zhong.shao, david.costanzo}@yale.edu

Abstract

Though attractive as a model for elastic on-demand service, cloud computing solutions based on existing hypervisors cannot guarantee that the provider will service a user's requests correctly, and will not leak sensitive information to unauthorized parties. We introduce **CertiKOS** (**Certi**fied **K**it **O**perating **S**ystem), a hypervisor architecture that leverages formal certification to ensure correctness and counter information leakage in cloud computing. CertiKOS isolates guest applications not only from each other but from provider-controlled resource management mechanisms. The kernel's API gives untrusted, provider-supplied management software control over allocation and delegation of resources such as memory and I/O devices, but prohibits management code from accessing a guest's memory or other resources while in use, or from interfering with a guest's execution except through clean resource revocation. CertiKOS represents an effort to apply recent advances in certified software design to a ground-up design of a modular and evolvable certified kernel. Through machine-checkable proof certificates and runtime monitoring, CertiKOS aims to offer users the assurance of correct and leak-free execution of their cloud services.

Categories and Subject Descriptors D.4.6 [*Operating Systems*]: Security and Protection; D.2.4 [*Software Engineering*]: Software/Program Verification

General Terms Design, Security, Verification

1. Introduction

Cloud computing offers a popular model for providers to deploy computing infrastructure and applications on-demand. Running sensitive applications or storing private information in the cloud also poses serious security concerns, however [6]. The risk of information leakage hampers adoption of the cloud model. In addition to external threats such as attacks from other customers on a multi-tenant platform [19], cloud servers also face internal attackers. A malicious operator with access to the provider's management software, or malware targeting hypervisor vulnerabilities [26, 7], can take control of the hypervisor to inspect or manipulate hosted guest computations. Thus, current cloud architectures require users to place high and perhaps unwarranted levels of trust in the operators and hypervisor software responsible for hosting their services.

APSys'11 July 11–12, 2011, Shanghai, China

We propose a proof-of-concept certified kernel, CertiKOS, as a new trusted foundation for cloud computing. CertiKOS isolates guest services not only from each other, but also from the provider's resource management facilities. In current hypervisor architectures, the management facility is a trusted operating system controlled by the provider—e.g., Xen's Dom0—which has complete access to the private information of any guest. Thus a compromised or a malicious provider can violate both the correctness and privacy of a guest computation. CertiKOS treats the provider's resource management facility as just another client, however. The management facility has the special privileges necessary to set resource management policies and delegate resources, but not to access those delegated resources while in use by clients. To guarantee that CertiKOS correctly implements this isolation, CertiKOS itself will be certified using modern formal tools to satisfy both correctness and information security properties. By using a machine-checkable formal proof to certify a kernel's correctness, together with standard TPM hardware attesting that the provider's system is actually running a formally certified version of CertiKOS, the provider's clients can gain high confidence of the correctness and security of their computations and information in the cloud.

CertiKOS represents a minimalistic design driven by the resource requirements of applications in a cloud environment. The current version of CertiKOS handles the delegation only of CPU cores, RAM, disk, and network I/O. CertiKOS offers privacy guarantees for each of these types of delegated resources. Through hardware-based virtualization [3, 1], CertiKOS also supports transparent isolation and resource management, supporting legacy OSes as clients.

The CertiKOS design leverages recent advances in certified software [23]. We organize CertiKOS as a set of certified modules, applying domain-specific logics and OCAP [10] to link separately certified modules into a complete certified kernel. As CertiKOS handles only resource delegation and isolation, it can be made small and thus feasible for certification. CertiKOS is composed of basic modules such as process management, memory management, network and disk I/O, and isolation and delegation components.

This paper makes the following contributions. First, we present a new architecture for countering information leakage in a cloud computing environment, particularly for defending against malicious service providers. Second, by separating resource delegation from resource management, CertiKOS treats the resource management facility as an untrusted client applications, preventing management software from accessing client spaces. Third, CertiKOS handles the main types of resources present in cloud environments, and can be easily extended to support flexible resource usage polices and provider business models. Fourth, CertiKOS introduces the first kernel architecture designed from the ground up for modular certification, Most importantly, CertiKOS enables end users to share the benefits of cloud computing while gaining confidence that their information will not be leaked or tampered with.

2. Motivation and Related Work

Companies are increasingly turning toward cloud providers, such as Amazon EC2, Salesforce CRM, and Rackspace, to host their information, applications, and IT services. In the current architecture, cloud service providers and their data center operators have full control over the machines hosting customers' information and applications, and can access customers' transactions and data at will. Service providers offer informal assurances that they will protect the integrity and privacy of customers' information and computations, but customers have limited means of ensuring that providers actually offer the claimed levels of security. Even assuming the provider and data center operators are normally trustworthy, external attackers may mount attacks from other hosted computations [19]. Worse, internal attackers or malware targeting hypervisor vulnerabilities [26, 7] can take advantage of the provider's management facility to access client information or tamper with guest computations. The cloud computing model would clearly benefit from a mechanism to protect clients' information and computations both from each other and from the management facility, while preserving the advantages of elastic, on-demand resource provisioning that make cloud environments attractive.

Related Work The security of hypervisors and virtual machine monitors is a well-researched topic [21, 25, 4], but traditional hypervisors cannot protect clients from cloud service providers. A hardware-based Trusted Platform Module (TPM) [24] can attest that a machine is running a particular trusted cloud kernel or hypervisor [12, 22], but the size and complexity of modern hypervisors makes it difficult to guarantee and verify their correctness.

Exokernel [9] supports user-level management of physical resources, but is not concerned with information security, and leaks extensive resource management/allocation information to all user space code. NoHype [14] extends the CPU architecture, removing the hypervisor layer to protect a guest OS, but is not applicable to current commodity processors.

Information Flow Control (IFC) enables the explicit labeling of information and controls its propagation. Asbestos [8] and HiStar [28] are operating systems employing IFC to enforce precise information security policies. These kernels are not formally certified, however, and must simply be trusted to enforce information flow control correctly. Additionally, these operating systems do not address threats such as timing side-channels, which represent increasing risks in cloud environments [19].

Singularity [13] implements its base kernel in a typesafe language, but its semantic correctness remains merely an assumption. The seL4 project [15] demonstrated that it is possible to verify a nontrivially-sized microkernel. However, the kernel's certification is limited: e.g., the virtual memory manager is left uncertified. With recent advances in certified software [23, 10], it has become practical to develop a fully-certified OS kernel. CertiKOS is designed to leverage and showcase these certification techniques.

3. The Kernel Design

Figure 1 illustrates the architecture of CertiKOS. The design currently handles only the most common types of resources in cloud environments: CPU cores for computation, RAM and disk storage, and network controllers for communication. CertiKOS implements a low-level layer of abstraction over physical resources. Both the provider's resource management facilities and cloud customers' software execute atop this kernel, which protects guests from each other and from the provider's management software. The rest of this section highlights relevant design details and rationale.

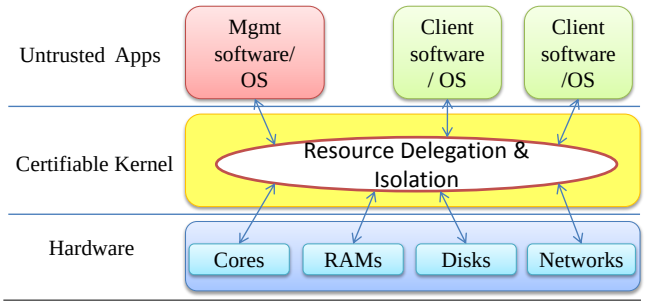

Figure 1. Overview of CertiKOS

3.1 Trust Model

We leave physical attacks out of scope, assuming that hardware is trusted and physically well-protected, a reasonable assumption in data centers designed for security. End users establish trust in CertiKOS via two mechanisms: by verifying the static correctness of the CertiKOS kernel, and by verifying its correct deployment at runtime. We will apply recent advances in certified software to produce a formal, machine-checkable proof of the correctness of the kernel's resource delegation and isolation properties. By distributing these formal proofs with CertiKOS, end users can confirm its security without trusting the kernel developer. The user relies on a standard hardware-based root of trust, such as a TPM [24], to attest that the cloud hardware is running the formally certified kernel. We assume that the provider's management software and all guest software are untrusted, and may be compromised or malicious.

3.2 Design Goals and Rationale

The main design principle underlying CertiKOS is to separate high-level resource management software from low-level resource delegation and protection mechanisms. CertiKOS separates the traditional resource management mechanisms implemented by existing hypervisors, such as Xen's Dom0, into two trust levels: a certified low-level layer implementing only minimal resource delegation and protection mechanisms, and a higher-level resource management layer, which is trusted in conventional hypervisors but untrusted in CertiKOS. The certified kernel's sole purpose is to implement the resource delegation policies specified by the untrusted, server-provided resource management software, while protecting the computation integrity and information security of cloud guests from each other and from the provider's management software.

Since management and guest software are equally untrusted by the kernel, the kernel implements isolation mechanisms ensuring that each untrusted domain has access only to its own resources. To implement this isolation, CertiKOS maintains ownership records for all hardware resources. When an end user requests cloud resources, the provider's management software allocates the resources requested by the client, then uses CertiKOS's resource management interface to delegate the resources to the client's execution. This resource management interface, accessible only by the provider-controlled management software, allows the provider to update the CertiKOS's resource ownership records by reassigning ownership of specific resources to the client. For example, when the management software starts a hosted client, it might use the CertiKOS management interface to assign a particular set of CPU cores, a particular physical memory range, a particular disk partition, and a particular network interface controller (NIC) to the client. This management interface is designed—and certified—to prevent information leaks. The provider's management software can later revoke memory or disk storage it assigned the client, for example, but the kernel

guarantees that the reassigned storage is appropriately "scrubbed" before being returned to provider control.

3.3 Resource Abstraction and Delegation

Cloud users primarily require three types of resources: computation, storage (RAM and disk), and networking. CertiKOS provides interfaces to the provider's resource management layer to assign and delegate access to these resources, but once assigned, the kernel exposes the assigned resources directly to the current owner, with minimal virtualization or abstraction in the kernel.

The CertiKOS kernel implements no dynamic resource scheduling, such as multiplexing a single CPU core among multiple guests, as current VMs often do. Instead, it relies on the provider's untrusted resource management code to perform any necessary resource allocation and scheduling at relatively coarse granularity. This design principle relies on the fact that modern server hardware usually provides the relevant resources—memory, CPU cores, and real or hardware-virtualized NICs—in sufficient number to make spatial partitioning feasible. We expect CertiKOS's focus on spatial rather than temporal partitioning also to increase the architecture's resistance to side-channel attacks in the cloud [19].

CPU Cores: CertiKOS allocates CPU resources spatially, at core granularity. In the current design, CertiKOS simply assigns one guest to each core, though future extensions could allocate CPUs more flexibly. CertiKOS provides interfaces for management software to allocate CPU cores to clients and revoke them.

RAM and Disks: CertiKOS abstracts both RAM and disks as memory space for untrusted management software and guests. The kernel includes no file system, leaving this functionality to untrusted guest software. CertiKOS exposes only interfaces supporting delegation of and protected access to memory and disk storage.

Networking: CertiKOS exposes interfaces allowing a provider's management software to give clients access to the provider's shared network infrastructure. The current design assumes that server hardware provides either enough physical NICs, and/or hardware-virtualized NICs, to dedicate at least one NIC to each client without software multiplexing. CertiKOS may be enhanced in the future to provide its own dynamic multiplexing of network interfaces, but this is not a high priority since hardware-based NIC virtualization is becoming increasingly common and inexpensive.

3.4 Resource Isolation

The provider's management software uses the trusted kernel's delegation interfaces to allocate and revoke resources according to the provider's policy. The trusted kernel in turn updates its ownership records to restrict each untrusted domain's resource access appropriately using standard protection and virtualization techniques. At runtime, CertiKOS uses its ownership records to check the permission on all explicit access requests, and to configure hardware-based protection mechanisms such as MMU and IOMMU hardware. In this way CertiKOS enforces resource isolation among applications and prevents information leakage.

Figure 2 illustrates the conceptual isolation model. A client's resources must be allocated in advance (step 1), otherwise the kernel denies the client's access request at runtime (step 5). When the client attempts to access its own resource, it invokes the kernel (step 2), which intercepts the request and checks the appropriate ownership record (step 3). If the client is the current owner of the requested resource, the kernel services the access request (step 4).

The kernel imposes access permission checks even on access requests by the provider's management software (step 6). Although

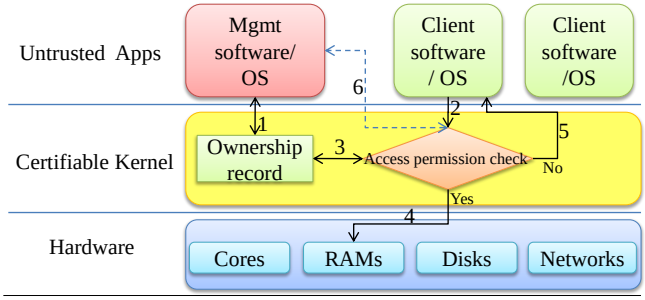

Figure 2. Permission check for access requests
1. Resource allocation/revocation; 2. Access request from a client; 3. Query the ownership record; 4. Authorize the request; 5. Deny the unauthorized request and return a fault message; 6. Access request from the management software.

the management software can allocate and revoke cloud hardware resources, it is not allowed to access resources owned by clients.

CertiKOS intercepts access requests based on resource types. In the current version of CertiKOS, CPU cores are handled by the kernel's minimal CPU multiplexer, and are allocated at a granularity of entire cores. Clients may choose to implement their own finer-grained scheduling atop CPU cores they own.

CertiKOS employs memory management hardware to partition and isolate guests' access to physical memory. In the current version, CertiKOS uses the AMD Secure Virtual Machine [3] facilities, setting up hardware page tables to represent memory ownership and relying on the hardware to translate guest virtual addresses to host physical addresses. Guests manage and access their assigned physical memory in the same way as on a physical platform, enabling clients to run traditional guest operating systems. CertiKOS uses IOMMU (Input/Output Memory Management Unit) hardware to protect DMA-based access requests from guests via physical or hardware-virtualized I/O devices such as NICs.

3.5 Resource Management and Usage

CertiKOS isolates management and guest software into strongly isolated runtime environments. The kernel gives the provider's management software the special privileges required to allocate and revoke resources, in order to implement the provider's high-level business model. Clients in turn manage their own assigned resources using their own mechanisms, such as those in a conventional guest OS. The current CertiKOS design focuses on managing and protecting a single multicore machine, although we expect to extend it in the future to offer secure clustering across data center networks.

A client may request and be assigned resources in several ways:

- Explicitly, using IPC to contact the management software and requesting resources. It is up to the client and the management to agree on the structure of the message. The kernel is not involved, and does not offer privacy guarantees to inter-domain messages. A client might request additional memory this way, for example.
- Explicitly, by including the resources in its initial loading instructions. When the management software loads the client, the client's object format (which is up to the provider and the client to agree on) may specify resources that the process requires. The management then checks that the specified resources observe its policy, and instructs the kernel to allow the client access to them.
- Implicitly. The client program may try to access a resource, initially causing a fault caught by the kernel. The kernel notifies the management program of the fault event, scrubbed of any

private client data. The management interprets the fault, assigns the requested resources to the client if appropriate by asking the kernel to update the ownership records, then restarts the client with the additional resources.

Resource Revocation Given unrestricted ability to revoke arbitrary subsets of a client's resources at any time, compromised management software might attempt to interfere with a client's correct execution, and/or attempt to learn information about a guest through side-channel attacks. Management software might selectively revoke assigned resources at a provider-chosen moment in time, for example, and monitor how this revocation affects visible guest behavior in an attempt to "probe" what a guest is doing based on its resource usage.

To guard against such side-channel attacks, CertiKOS gives untrusted management software only one *non-consensual* means of revoking guest resources: by destroying the entire guest and revoking all its resources at once, in "all-or-nothing" fashion. Such an event essentially represents a provider-induced virtual node failure, which the kernel guarantees will operate in a fail-stop fashion without leaking any information about the former guest's operation.

If the provider wishes to revoke resources from guests at finer granularity, it can communicate explicitly with its guests to *request* that they give up certain resources gracefully. The guest saves any state the resources may have held (for memory or disk), and notifies the kernel when it is ready to relinquish the resources, which in turn passes resource ownership back to the provider. If the guest fails to respond in a reasonable time, according to a policy agreed between provider and client, the provider can always escalate to an "all-or-nothing" revocation without the guest's consent.

4. Certifying the Kernel

To verify CertiKOS, we will take advantage of its minimalistic design and several recent advances in software certification [23].

First, we will not certify what we do not have to. Much of the complexity in cloud virtualization lies in resource management algorithms. CertiKOS delegates this functionality to the untrusted management layer, implementing only permission checking in the kernel. Thus, we can omit sophisticated policy management and scheduling tasks from the kernel without sacrificing functionality, leaving less kernel code to certify formally.

Second, we are designing the kernel ground-up for certification, instead of trying to certify existing software. Instead of disentangling complex existing kernels, therefore, we are developing a kernel specifically to be easy to reason about. In particular, the kernel is designed to be highly modular, with strong and readily formalizable invariants defining module boundaries. Although conventional wisdom may suggest that such strong modularity risks incurring high performance overhead, our formal layering and modular reasoning techniques are capable of expressing modules whose formal boundaries do not actually impose additional code at each layer or interface, which may mitigate this performance risk. Further, even if the resulting certifying kernel achieves far from optimal performance, CertiKOS's focus on leveraging hardware protection and virtualization mechanisms, and spacial rather than temporal resource assignment, leaves the certified kernel responsible mostly for occasional setup, teardown, and guest fault handling, with hardware handling most of the performance-critical resource access paths.

We will employ domain-specific logics to certify different CertiKOS modules through the use of OCAP [10], a framework designed for this purpose. This approach allows us to integrate several frameworks that have proven effective for certifying programs involving dynamically allocated memory [27], interrupts [11], non-interfering concurrent threads [11], and self-modifying code [5].

Another technique we will employ allows certified linking of components verified at different abstraction levels. Since only the bootloader loads code dynamically, for example, we need use a self-modifying code framework only for bootloader verification. The rest of the kernel can use a fixed code heap abstraction. We can then link these modules together across their different levels of abstraction, such that the entire resulting code is certified.

We are developing an approach for merging Decentralized Information Flow Control (DIFC) [17] with ideas from Separation Logic [18]. We will develop a logic in which we can specify and prove the correctness of information flow policies. We will apply this logic to CertiKOS to prove that the kernel properly enforces isolation. This guarantees that our kernel cannot, for example, accidentally leak data from one client's memory to another's.

We will use a Flume-like label model [16] to track the origin of all data. In this model, memory locations are "tainted" with tags representing clients. If the kernel reads data from client A, that data is tainted with tag A. If the kernel then attempts to write this data into client B's memory, which is tainted with tag B, our logic catches this bug, causing certification to fail. Of course, there are situations in which data must be passed across domains, such as IPC between client and provider software. As in any useful DIFC system, our logic will allow controlled *declassification* of data in appropriate situations to handle such cases explicitly. Our logic will make all declassification explicit, allowing for formal certification of high-level declassification policies. For example, we might prove that the kernel never leaks data from one client to another, *except for* IPC messages. It then becomes the client's responsibility to avoid leaking information via IPC. For more discussion of the role and necessity of declassification in DIFC, see [20].

5. Implementation

The CertiKOS design presented here is just a first step in a larger project to develop a fully-certified, practical OS kernel. As the first stage of this project, we wished to implement a lightweight and modular kernel with minimal functionality to facilitate certification. Our minimal kernel design reflects the goal of separating out and surgically confronting critical cloud security problems.

We are now building a prototype based on PIOS[2], by further modularizing the kernel and minimizing global interdependencies. Figure 3 illustrates this conceptual simplification by comparing the dependency graphs of the old and new kernels.

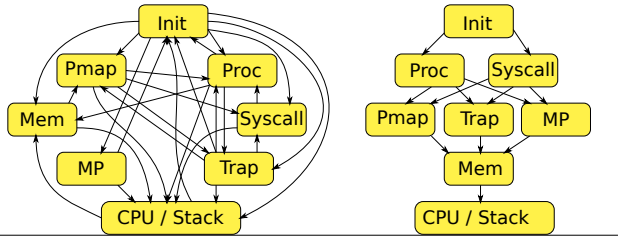

Figure 3. Invariant dependencies of PIOS vs. our prototype

We believe that a massive redesign of core code of the kernel will be crucial for verification and reasoning about isolation properties.

Our prototype allows us to manage the client's use of CPU cores, limit its use of memory, and guarantee that management and client code cannot access each other's memory. Continuing development is leading in two main directions. First, we will improve and generalize the features of our kernel. Our permission system is currently

ad hoc, and does not readily extend to new resources. Second, we will more rigorously separate the kernel components, facilitating the formal specification and verification of each component.

Atop the kernel, we are building an untrusted management application as an example. This application serves as a demo showing how a cloud provider might use our kernel. Our aim for this demo is to illustrate the kernel's ability to allow the management application to safely run the clients' applications, and manage their access to resources, while formally guaranteeing security and isolation.

6. Conclusion

This paper presents CertiKOS, a certified kernel for secure cloud computing. CertiKOS handles resource delegation and isolation, and protects clients from each other and from the provider. The current version targets a single platform; we will later investigate cross-machine clustering and migration.

Acknowledgment We thank anonymous referees for their suggestions and comments on an earlier version of this paper. This research is based on work supported in part by the DARPA CRASH grant FA8750-10-2-0254 and NSF grants CNS-1017206, CNS-0915888, CNS-0910670, and CCF-0811665. Any opinions, findings, and conclusions contained in this document are those of the authors and do not reflect the views of these agencies.

References

[1] Intel virtualization technology (VT). `http://www.intel.com/technology/virtualization/technology.htm`.

[2] Parallel instructional operating system. `http://zoo.cs.yale.edu/classes/cs422/pios`.

[3] AMD. AMD64 virtualization codenamed "Pacifica" technology — secure virtual machine architecture reference manual. Tech. Rep. Publication Number 33047, Revision 3.01, AMD, May 2005.

[4] AZAB, A. M., NING, P., WANG, Z., JIANG, X., ZHANG, X., AND SKALSKY, N. C. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *17th ACM Conference on Computer and Communications Security, 2010*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds., pp. 38–49.

[5] CAI, H., SHAO, Z., AND VAYNBERG, A. Certified self-modifying code. In *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '07, pp. 66–77.

[6] CIRCLEID. Survey: Cloud Computing No Hype, But Fear of Security and Control Slowing Adoption. `http://www.circleid.com/posts/20090226_cloud_computing_hype_security/`.

[7] CVE. CVE-2008-2100: VMware Buffer Overflows in VIX API Let Local Users Execute Arbitrary Code in Host OS. `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2100`.

[8] EFSTATHOPOULOS, P., KROHN, M., VANDEBOGART, S., FREY, C., ZIEGLER, D., KOHLER, E., MAZIÈRES, D., KAASHOEK, F., AND MORRIS, R. Labels and event processes in the asbestos operating system. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP'07)* (Brighton, UK, Oct. 2005), ACM, pp. 17–30.

[9] ENGLER, D. R., KAASHOEK, M. F., AND O'TOOLE, JR., J. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pp. 251–266.

[10] FENG, X. *An Open Framework for Certified System Software*. Ph.D. thesis, Department of Computer Science, Yale University, 2007.

[11] FENG, X., SHAO, Z., DONG, Y., AND GUO, Y. Certifying low-level programs with hardware interrupts and preemptive threads. In *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '08, pp. 170–182.

[12] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pp. 193–206.

[13] HUNT, G. C., LARUS, J. R., ABADI, M., AIKEN, M., BARHAM, P., FAHNDRICH, M., HAWBLITZEL, C., HODSON, O., LEVI, S., MURPHY, N., STEENSGAARD, B., TARDITI, D., WOBBER, T., AND ZILL, B. An overview of the Singularity project. Tech. Rep. MSR-TR-2005-135, Microsoft Research, Redmond, WA, USA, Oct. 2005.

[14] KELLER, E., SZEFER, J., REXFORD, J., AND LEE, R. B. Nohype: Virtualized cloud infrastructure without the virtualization. In *Proc. 37th International Symposium on Computer Architecture (37th ISCA'10)* (Saint-Malo, France, June 2010), ACM SIGARCH, pp. 350–361.

[15] KLEIN, G., ELPHINSTONE, K., HEISER, G., ANDRONICK, J., COCK, D., DERRIN, P., ELKADUWE, D., ENGELHARDT, K., KOLANSKI, R., NORRISH, M., SEWELL, T., TUCH, H., AND WINWOOD, S. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd Symposium on Operating Systems Principles (22nd SOSP'09), Operating Systems Review (OSR)* (Big Sky, MT, Oct. 2009), ACM SIGOPS, pp. 207–220.

[16] KROHN, M. N., YIP, A., BRODSKY, M. Z., CLIFFER, N., KAASHOEK, M. F., KOHLER, E., AND MORRIS, R. Information flow control for standard os abstractions. In *SOSP* (2007), pp. 321–334.

[17] MYERS, A. C., AND LISKOV, B. A decentralized model for information flow control. In *SOSP* (1997), pp. 129–142.

[18] REYNOLDS, J. C. Separation logic: A logic for shared mutable data structures. In *LICS* (2002), pp. 55–74.

[19] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pp. 199–212.

[20] SABELFELD, A., AND SANDS, D. Declassification: Dimensions and principles. *Journal of Computer Security 17*, 5 (2009), 517–548.

[21] SAILER, R., JAEGER, T., VALDEZ, E., CACERES, R., PEREZ, R., BERGER, S., GRIFFIN, J. L., AND DOORN, L. V. Building a mac-based security architecture for the xen open-source hypervisor. In *Proceedings of the 21st Annual Computer Security Applications Conference* (2005), pp. 276–285.

[22] SANTOS, N., GUMMADI, K. P., AND RODRIGUES, R. Towards trusted cloud computing. In *Proceedings of the conference on Hot topics in cloud computing*, HotCloud'09, pp. 3–3.

[23] SHAO, Z. Certified software. *Commun. ACM 53* (December 2010), 56–66.

[24] TRUSTED COMPUTING GROUP. TPM main specification. `http://www.trustedcomputinggroup.org`, Feb. 2005.

[25] WANG, Z., AND JIANG, X. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (2010), SP '10, pp. 380–395.

[26] WOJTCZUK, R. Subverting the Xen hypervisor. In *Blackhat* (2008). `http://www.invisiblethingslab.com/resources/bh08/part1.pdf`.

[27] YU, D., HAMID, N. A., AND SHAO, Z. Building certified libraries for pcc: dynamic storage allocation. *Sci. Comput. Program. 50* (March 2004), 101–127.

[28] ZELDOVICH, N., BOYD-WICKIZER, S., KOHLER, E., AND MAZIÈRES, D. Making information flow explicit in HiStar. In *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation* (Nov. 2006).