Trusted Clouds

Mike Burmester

Florida State University, Department of Computer Science
Tallahassee, FL 32306, USA
{burmester}@cs.fsu.edu

‘*When you have completed 95% of a journey then you are halfway there*’ Japanese proverb

Abstract. A new paradigm for network applications emerged in the 1990s as the centralized mainframe computer model evolved into a PC client/server based model. This captured a broader scope including business, commerce and finance. Recent Cloud computing and Big Data deployments suggest that we have now come full circle with centrally managed trust infrastructures supporting an even broader application base for any-time, any-where, synchronized access to data and services. This extends the flexibility/manageability of the client/server paradigm and allows for ubiquitous lightweight service endpoints such as notebooks, tablets or smart phones that do not need to store sensitive data (other than cryptographic keys in “sealed storage”).

Even though it may take some time before we understand the full extent of the Cloud paradigm, some features have already emerged and can be analyzed and studied. For example for backward compatibility, legacy practices will be maintained. In particular, cloud deployment models will comprise several technologies including public, private and hybrid. Also, past practices strongly support open virtualization, so clouds can be customized and tailored to specific security settings. Finally, the emerging paradigm will clearly be impacted by social media technologies and the Internet of Things, suggesting that social behavior, profiling and causal reasoning will play a major role.

In this report we analyze the cloud paradigm from a security point of view. Our goal is to show that for critical applications, not only is the new paradigm more flexible, but it is also technically easier to secure. Finally, the Cloud has a dark side, at least from a security point of view. We shall discuss some of its more obnoxious features.

1 Introduction

Cloud computing is an evolving paradigm. It is a technology that supplies on demand computing services as a utility, with price elasticity, continuity of service, quick scaling and reliability. NIST defines it as [1]: “A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.*, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”.

Below we briefly overview the cloud models, the provided services and the service agreements (for more details see [2]).

Models. There are four general types of cloud deployment models: public clouds, that supply infrastructure and computational resources to the general public over the Internet, and are owned and operated by cloud providers; community clouds that are owned and operated by several organization, but have common regulatory and security policies; and hybrid clouds.

Services. There are three basic cloud computing services on demand: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). The cloud client typically chooses the operating system and development environment to host the services. Service agreements specify the terms and conditions for access to cloud services For most cloud technologies security provisions that go beyond the basic infrastructure services are carried out by the cloud client.

Service agreements. These specify the terms and conditions for using cloud services, which includes the expected level of service—the Service Level Agreement (SLA), and the compensation if that level is not reached, licensing details as well as security and privacy.

Related Work. Most of the cloud computing publications in the literature focus on the "computing as a utility" business service (*e.g.,* [3–7]). There are only a few publications that address security issues in the Cloud (*e.g.,* [2, 8, 9]), and these use a fragmented approach.

2 Cloud architectures for secure applications

The Cloud is a client-driven access control infrastructure that manages computing services. A cloud Monitor mediates between clients and service providers with access granted based on service agreements that establish a trust-bond between clients and providers.

2.1 Cloud Monitors

A cloud Monitor can be modelled by a trust-graph $G = (V, E)$. G is a directed labeled graph with nodes $X, Y, Z, \dots$, the clients, the cloud providers and the cloud services. There are two types of edges: (a) edges $X \xrightarrow{\tau_{xy}} Y$ that link clients X to providers Y with labels τ_{xy} that contain: an SLA, terms of use, privacy/security policies and the compensation in the event the provider fails to deliver at the specified level or violates agreed policies; (b) edges $Y \xrightarrow{\tau_{yz}} Z$ that link providers Y to services Z with labels τ_{yz} that contain: the agreement regarding the particular service Z, privacy/security policies and the compensation if Z is not delivered at the specified level or agreed policies are violated.

Labels of the type τ_{xy} capture the confidence the client has in the provider regarding specific services as well as the risks involved (a function of the criticality of the service and the agreed compensation). Labels of the type τ_{yz} capture the confidence that the client has in the specific service Z.

Trust is not transitive: $X \xrightarrow{\tau_{xy}} Y$ and $Y \xrightarrow{\tau_{yz}} Z$ implies $X \xrightarrow{\tau_{xz}} Z$ only when the service agreement of τ_{yz} is specified in the description of τ_{xy} as a required service. In this case we say that τ_{xy} dominates τ_{yz} and write $\tau_{xy} \succeq \tau_{yz}$. Access to a service Z provided by Y is granted to client X if $\tau_{xy} \succeq \tau_{yz}$. For private clouds, trust labels can be much simpler reducing to, for example, public key certificates, or symmetric keys.

The trust-graph G is dynamic with edges added (deleted) in real-time corresponding to new service requests (completions) or new services becoming available (withdrawn).

2.2 Access control models

Access control models are trust infrastructures that manage the resources of computer or network systems. Bell-LaPadula [10] was the first proposed access control model. It enforces need-to-know (confidentiality) policies. Other models followed. Biba [11] enforces integrity policies; Clark-Wilson [13] enforces separation of duties (integrity) policies; and Role Based Access Control (RBAC) [12] enforces authorization policies. In these models, clients and resources are assigned labels selected from a linearly ordered set (or lattice) and access is based on domination. For example, in Bell-LaPadula a client with “secret” label (clearance) can access all resources with label (classification) “secret” or less.

These models all focus on managing data resources, *not* computing services that are functions of data. For cloud deployments we have to design new access control models and specify new policies for the secure management of computing services. In this report we propose to use the trust levels τ in Section 2.1 as labels, appropriately modified to capture dynamic management and policies that enforce need-to-know and separation-of-duties.

For cloud deployments need-to-know refers to the trust τ_{xz} required for accessing a particular computing resource Z: it should be no more than strictly necessary. For example, a client and provider need not share any secret encryption keys if private channels are not necessary for the service. If secret keys are needed (*e.g.,* for integrity) then these must be session keys. If long term keys have to be shared then these must be public keys.

Separation of duties refers to the trust τ_{yz} between a provider Y and the computing service Z: it should be no more than strictly necessary. In particular the provider should not be able to access data or secret keys of the client. For example, for an Infrastructure-as-a-Service application, the cloud provider should not get access to any data or secret keys that the IaaS shares with the cloud client.

2.3 Trusted Computing

The Trusted Computing Group [14] has published specifications for architectures and interfaces for several computing implementations. Platforms based on these are expected to meet the functional and reliability requirements of computer systems and provide increased assurance of trust. The Trusted Platform Module (TPM) [15] and the Trusted Network Connect (TNC) [16] are two such architectures.

The TPM is a Trusted Computing (TC) architecture that binds data to platform configurations of hardware systems to enhance software security. It has two basic capabilities: remote attestation and sealed storage, and is supported by a range of cryptographic primitives. TPMs employ trusted engines, called *roots of trust*, to establish trust in the expected behavior of the system. Trust is based on an integrity protected boot process in which executable code and associated configuration data are measured before execution—this requires that a hash of the BIOS code is stored in a Platform Configuration Register (PCR).

For remote attestation the TPM uses an attestation identity key to assert the state of the current software environment to a third party—by signing PCR values. Sealed storage is used to protect cryptographic keys. To encrypt/decrypt/authenticate, keys are released conditional on the current software state (using current PCR values). The TPMs must be physically protected from tampering. This includes binding the TPM to physical parts of the platform (*e.g.* the motherboard).

The TNC is a TC architecture for trusted network applications. What distinguishes TNC from other interoperability architectures is the requirement that the OS configuration of the client and server is checked prior to a communication channel being established. A trusted link between a client and server is established only if: (*i*) the identity of the client and server is trusted. A Public Key Infrastructure is used to establish trust-links between a Root Authority and the TPMs of the client/server; (*ii*) the client has real-time access to the server; (*iii*) the client and server are authenticated. A root of trust on the TPM of both parties is invoked to release the required keys to execute a handshake protocol [16]; (*iv*) the integrity of communicated data, and if necessary the confidentiality, is enforced by the TPM.

The TC paradigm has been studied extensively, with TPM- and TNC-compliant systems implemented in several configurations. There are some concerns regarding implementations, which mainly involve poor design: the TC paradigm relies heavily on strict compliance to policies, procedure and hardware design, and unless these are being adhered to, there is no protection. Other concerns involve "Big Brother" privacy issues. However for critical applications or DoD type networks, implementation issues and privacy leakage can be addressed.

2.4 A threat model and security framework for TC-compliant systems

The TPM prevents compromised components of a TC-compliant system from executing. As a result, if we exclude run-time (execution) threats, malicious

(Byzantine) threats are reduced to DoS threats that can be addressed with redundancy.

There are two kinds of faults that may affect a TC-compliant computer system: natural (this includes accidents) and adversarial (intentional/malicious/ insider). Natural faults can be predicted, in the sense that an upper bound on the probability of such faults can be estimated. Redundancy can then be used to reduce this probability to below an acceptable threshold. Malicious DoS faults cannot be predicted. However they are overt and, because of the TPM and TNC integrity verification, must be physical (*e.g.*, involve tampering the TPM chip). So there is a cost involved. One way to thwart them is to make the cost high enough to prevent them.

There are several security models that use economics and risk analysis based on redundancy [17] that are appropriate for threat models with overt faults. These assume a bound on adversarial resources and an architecture with sufficient redundancy to make such DoS attacks prohibitively expensive.

2.5 The good, the bad and the ugly

The good. The TPM protects system components from behaving in an unexpected way. In particular, prior to the execution of any trusted program an integrity check of its state (against a stored PCR configuration) is required. Consequently if the program is compromised it will not be executed by the OS.

The bad. The TPM allows only trusted code to execute. Therefore the integrity of trusted code (which includes the OS) is a fundamental requirement in order to ensure trust in the computing infrastructure. The system software must be well designed, with no security holes backdoors or vulnerabilities that could be exploited by an adversary. An exploit in the OS may allow the adversary to bypass the protection offered by the TPM. There are several reasons why the design of software programs may be faulty. A major reason is the complexity of the execution environment (the OS and CPU hardware). Another is poor software development practices.

The ugly. "*Security is not necessarily composable*". Proof-carrying code is not closed with respect to composability unless the proofs are composable (Universal Composability [18]). An interesting example involving routing protocols is discussed in [19], where it is shown that a routing protocol that is secure in isolation is not secure when executed concurrently with itself. Consequently the TPM provides integrity guarantees only at load-time, not run-time.

An exploit of the OS may make it possible for the adversary to change the execution flow of a trusted program. There are several run-time attacks [20] that use metamorphic malware such as the self-camouflaging Frankenstein [21] or more generally, return oriented programming (ROP) [22]. For these the adversary must be able to control the execution flow on the stack, and there are ways to prevent this [23].

However as pointed out earlier, even if there are no exploits, concurrent execution of trusted code (that is not Universally Composable) may lead to untrusted behavior.

2.6 An Architecture for Trusted Clouds

The basic components of a Cloud are: the clients, the providers, the computing services and the cloud Monitor. For a Trusted Cloud we propose an architecture with:

- A private cloud deployment and trusted service providers.
- A trusted cloud Monitor (Section 2.1).
- An access control model for computing services that supports need to know and separation of duties policies (Section 2.2).
- TC-compliant computing services (Section 2.3).
- Lightweight TC-compliant client service endpoints.

The private cloud deployment is intended to secure computing services for critical infrastructures and DoD type networks. This deployment should not be used in hybrid mode, or concurrently with other clouds. All service providers should be trusted to adhere to service agreements as well as the security policies. The cloud Monitor should enforce need-to-know and separation of duties policies. All computing services should be TC-compliant. This prevents execution of untrusted code, while guaranteeing adherence to the service agreements. The requirement for lightweight TC-compliant service endpoints is based on the fact that the Trusted Cloud is the most appropriate place to store sensitive data (from physical and cyber threats). Ideally a lightweight (such as TinyOS [27]) operating system should be used—all the code that is needed can be run on a PaaS.

Assuming that all the components of the Cloud are trusted, and that all execution code is trusted, the only remaining (load-time) threats are DoS attacks, which because of the redundancy in the Cloud are not a concern. For applications in which real-time access is critical, services may have to be prioritized. The system must have enough redundancy to guarantee that all critical computing services are executed in real-time.

3 The Dark Side of the Cloud

Well defined architectures that are based on trusted system behavior in the presence of an adversary will be secure unless the trust is breached. By assuming that all service providers are trusted, and that any computing service will not deviate from its expected behavior (because of the TC controls), we make certain that the only remaining threats are those that bypass the trust mechanisms. From our discussion in Section 2.5 (the ugly case) it is clear that concurrent execution of trusted codes may lead to untrusted (run-time) system behavior. To mitigate such threats, any successful approach will have to: (a) limit the openings for exploitation on platform software and, (b) employ methods to detect run-time compromise.

(a) To achieve a smaller attack surface, client endpoint devices must abide by constraints on functionality and resource usage, operating with a structured, well-defined, enumerated set of duties. Clearly the presence of software flaws is related to the complexity and size of software.
(b) There is substantial existing work on techniques for dynamically detecting code faults (dynamic integrity monitoring and taint analysis) [24–26]. Although most solutions carry a significant computational overhead, critical applications can justifiably be expected to bear the computational resources. However, the time required to address such threats may be an issue.

Acknowledgments. This material is based upon work supported by the National Science Foundation Grant No. 1027217.

References

1. P. Mell and T. Grance, "The NIST Definition of Cloud Computing", Publication 800-145, 2011. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`
2. W. Janson and T. Grance, "Guidelines on Security and Privacy in Public Cloud Computing", NIST Publication 800-144, 2011. `http://csrc.nist.gov/publications/nistpubs/800-144/SP800-144.pdf`
3. S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Comp. Syst.*, vol. 29, no. 4, pp. 1012–1023, 2013.
4. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
5. T. Velte, A. Velte, and R. Elsenpeter, *Cloud Computing, A Practical Approach*, New York, NY, USA: McGraw-Hill, Inc., 2010.
6. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comp. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.
7. A.-M. K. Pathan, J. Broberg, and R. Buyya, "Maximizing utility for content delivery clouds," in *WISE*, ser. Lecture Notes in Computer Science, G. Vossen, D. D. E. Long, and J. X. Yu, Eds., vol. 5802. Springer, 2009, pp. 13–28.
8. R. H. K. Yanpei Chen, Vern Paxson, "Whats New About Cloud Computing Security? CS Division, EECS Dept. UC Berkeley, TR UCB/EECS-2010-5."
9. B. R. Kandukuri, R. P. V., and A. Rakshit, "Cloud security issues", in *IEEE SCC*. IEEE Computer Society, 2009, pp. 517–520.
10. E. D. Bell and J. L. La Padula. Secure computer system: Unified exposition and Multics interpretation. Bedford, MA, 1976.
11. K. J. Biba. Integrity considerations for secure computer systems. MITRE Corp., Tech. Rep., 1977.
12. R. S. Sandhu, E. J. Coyne, and C. E. Feinstein, H. L.and Youman Role-Based Access Control Models. *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

13. D.D. Clark and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. newblock *Proc. IEEE Symp. on Research in Security and Privacy* (SP'87), May 1987, Oakland, CA. IEEE Press, pp. 184–193.
14. Trusted Computing Group (TCG), "http://www.trustedcomputinggroup.org/."
15. TCG, "TPM Structures, Level 2, Version 1.2, Revision 116, Communication networks and systems for power utility automation", *http://www. trustedcomputing-group. org/resources/tpm_main_specification*, March 2011.
16. ——, "Trusted Network Connect Architecture for Interoperability; Specification 1.3; Revision 6," April 2008.
17. M. Lelarge and J. Bolot, "Economic incentives to increase security in the internet: The case for insurance," *INFOCOM 2009*, pp. 1494–1502, 2009.
18. R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS.* IEEE Computer Society, 2001, pp. 136–145.
19. M. Burmester and B. de Medeiros, "On the security of route discovery in manets", *IEEE Trans. Mob. Comput.*, vol. 8, no. 9, pp. 1180–1188, 2009.
20. A. Baratloo, N. Singh, and T. Tsai, "Transparent run-time defense against stack smashing attacks," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '00, Berkeley, CA, USA: USENIX Association, 2000, pp. 21–21.
21. V. Mohan and K. W. Hamlen, "Frankenstein: Stitching malware from benign binaries," in *WOOT*, E. Bursztein and T. Dullien, Eds. USENIX Association, 2012, pp. 77–84.
22. R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-Oriented Programming: Systems, Languages, and Applications," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 2:1–2:34, 2012.
23. Davi, Lucas and Sadeghi, Ahmad-Reza and Winandy, Marcel, "Ropdefender: a detection tool to defend against return-oriented programming attacks," in *Proc. 6th ACM Symposium on Information, Computer and Communications Security* (ASI-ACCS '11), New York, NY, USA: ACM, 2011, pp. 40–51.
24. Walter Chang, Brandon Streiff, and Calvin Lin. Efficient and extensible security enforcement using dynamic data flow analysis. In *ACM Conference on Computer and Communications Security*, pages 39–50, 2008.
25. W. Cheng, Qin Zhao, Bei Yu, and S. Hiroshige. Tainttrace: Efficient flow tracing with dynamic binary rewriting. In *11th IEEE Symposium on Computers and Communications, 2006. ISCC '06*, pp. 749–754, 2006.
26. Feng Qin, Cheng Wang, Zhenmin Li, Ho seop Kim, Yuanyuan Zhou, and Youfeng Wu. Lift: A low-overhead practical information flow tracking system for detecting security attacks. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 135 –148, dec. 2006.
27. TinyOS, http://www.tinyos.net/