Editors: Lori M. Kaufman, lori.kaufman@ieee.org
Bruce Potter, bpotter@pontetec.com

Monitoring Cloud Computing by Layer, Part 2

Jonathan Spring
Software Engineering Institute

In part 1, I briefly introduced cloud computing and a model of it that has seven layers (facility, network, hardware, OS, middleware, application, and the user). Each cloud computing deployment must have these layers, but different deployment types give control of them to different parties. Here, I cover controls that could be implemented in the middleware, application, and user layers to monitor and audit information assurance.

This information is relevant to whichever entity controls the layer; however, the customer is ultimately responsible for ensuring compliance with his or her standards. Often, this must be accomplished through contracts and service-level agreements (SLA). Such agreements must include strong auditing and monitoring powers for the customer. As I discussed in part 1, no technical controls can absolutely prevent fraud because the owner of the information is divorced from the owner of the hardware and facilities. As I discuss here, there's also plenty to monitor and audit in the higher layers. Customers should require in their agreements that the provider demonstrates all the controls it supplies.

Middleware

Middleware is a broad topic that can range from virtualization management tools, to data format conversion, to allowing applications to abstract away from the cloud architecture on which the middleware runs.[1] Some middleware also claims to perform security functions across heterogeneous cloud architectures, such as enforcing role-based access control in a distributed fashion.[2] Middleware is a significant potential weak point in a cloud provider's or customer's information assurance capabilities. In many ways, it's novel in cloud computing, and as such is the most immature layer.

Regardless of the middleware's function, there are various safeguards to implement and pitfalls to avoid. The middleware is the natural place to monitor and secure communication between various system components because it mediates between the applications and the OS. So, the provider should be able to demonstrate that all its middleware will accept and transmit only encrypted data. If middleware will be part of the trusted code base provided to the customer, the provider should protect it against malicious manipulation just as strongly as the OS. If an attacker replaced or modified middleware functions, it would be as damaging as modifying the OS, if not more so. This is because middleware tends to have rights to access, manipulate, and distribute cloud-wide data, besides specialized functions such as managing access controls.

Assurance concerns fall in three categories. The first is software architecture flaws, which might result from human misconfiguration of resources or policies. Such flaws might be indicated by abuse of the middleware's normal functions, but anomalies generally don't correlate well with security violations. Intelligent monitoring of middleware functions is important but difficult. Maintaining an architecture that's simple or formulaic enough to prevent erroneous settings is important.

The second category is coding vulnerabilities, which are often due to a few relatively common software defects. Middleware should uphold rigorous coding standards. If the provider won't make its source code available for inspection, it should at least have a neutral third party inspect all code for compliance with secure coding standards (see www.cert.org/secure-coding) and make that report available to customers. Furthermore, during the software's development, the developers should test it for vulnerabilities using fuzzing tools in addition to normal testing procedures. Fuzzing tools try to supply erroneous inputs to the program in a controlled environment and document any crashes and errors so that the developer can fix them before deployment.[3,4] For the purposes of auditing the cloud provider, if its source code isn't publicly available, then the devel-

COPUBLISHED BY THE IEEE COMPUTER AND RELIABILITY SOCIETIES
1540-7993/11/$26.00 © 2011 IEEE

opment process's details should be. Documentation should include those details and a patch management plan for when vulnerabilities are inevitably discovered.

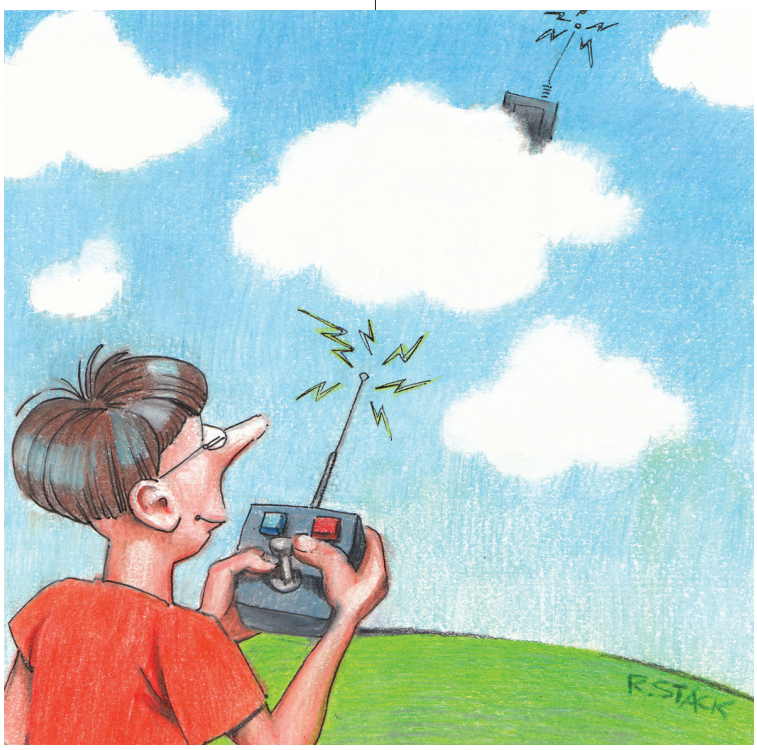
The third category is security services. Middleware claiming to actually provide security services, rather than just provide its own services securely, should be approached with particular caution. These services include monitoring, access control, data validation, and other functions. Obviously, the customer must supremely trust the software that's doing the monitoring or mediating access to the information. Providers often outsource the provision of middleware, which further complicates accountability and responsibility for errors and the interaction of trusted developers and stewards of various blocks of code. The more distributed the process is, the harder it is to audit the code and its authors for compliance with myriad regulations. A customer can't hope to closely watch all the watchmen but might expect to know at least who they are or how they got there.

Some security-minded middleware aims to manage versioning, backup, cost tracking, building, migration, and other similar functions.[5] A customer might also want to monitor service operations, usage metrics, service balancing, middleware communication patterns, user access, and data access operations. Some companies specialize in developing security and monitoring middleware (for example, WSO2; http://wso2.com/products), and large cloud providers might develop proprietary custom solutions. Whatever the solution is, the provider should disclose what and how it monitors so that the customer can make informed decisions. Although accomplishing this monitoring such that humans can make useful decisions based on it is challenging owing to the volume of data, such monitoring is feasible and advisable.

Applications

For software as a service (SaaS), the cloud layers I previously discussed are merely the vehicle with which to provide the application. The application is the forward-facing aspect of the SaaS provider and so will expose the most code to potentially malicious users. These factors make it critical that the development team has abided by the standards for secure coding and secure software development discussed in the section "Middleware." Also, the application and its foundation shouldn't rely on any kind of "security through obscurity." Customers should prefer applications in which the source code and business logic can be and have been carefully examined by neutral third parties for potential flaws. These development practices are also the goal for any outward-facing customer-developed application hosted in a platform as a service (PaaS) or infrastructure as a service (IaaS) deployment.

Web-based applications must operate in an insecure environment: Web browsers. So, applications should try to sufficiently monitor behavior to detect violations. Cloud environments should take precautions beyond Web security best practices because they have higher-than-average value as targets.[6] The ability to widely deploy stricter security policies is one potential advantage of cloud computing. For example, the provider should take care in deploying SSL and other digital certificates; with the distributed responsibility in cloud computing, deploying wildcard SSL certificates for services hosted in a cloud becomes riskier. The Domain Name System Security Extensions (DNSSEC) also are important for authenticating domains and controlling the authentication of domain names in the customer space. The provider should use DNSSEC to demonstrate legitimate domains so that the customer can authenticate the resources. This DNS traffic is another source to monitor for anomalies.



Both SSL and DNSSEC are examples of public-key cryptography that can be leveraged to help verify services provided through the cloud. The private keys for these or any service should never be put into the cloud; they should remain safely and solely in customer possession. If either the customer or the provider receives a request to alter public-key-based credentials, the change should be authenticated out of band. Being careful and persistent with the deployment of these and other cryptographic protocols will help prevent tampering with sensitive or critical data.

Middleware claiming to actually provide security services, rather than just provide its own services securely, should be approached with particular caution. These services include monitoring, access control, data validation, and other functions.

Applications should continue to uphold standard best practices, such as sanitizing all inputs. For monitoring in a traditional environment, a host-based security system can be set to monitor for abnormal behavior in the resident applications. However, monitoring specific violations per application in the cloud is more difficult because one instance of an application serves multiple tenants simultaneously and doesn't reside on a dedicated host. Because the whole application will likely sit in memory to accomplish the multitenant nature of cloud computing, an application compromised in memory could continue uncorrected indefinitely. Recovering memory dumps (as I noted in the section "OS") at intervals and analyzing the content can also serve to monitor applications. The same corruption detection concerns apply, unless the applications are refreshed from a known good copy from read-only media regularly. Both options are probably wise, because it's easier to automate refreshing an application image. If the system image is loaded from read/write media, then this solution isn't reliable because the application's disk version could also be compromised.

Host-based application security is analogous to some functions that antivirus software provides in a more traditional host-protection model. However, antivirus behavior, which operates a blacklist to identify forbidden actions, differs fundamentally from white lists, which identify permissible actions. White lists are far more secure than blacklists but lead to a much more restricted environment and are difficult to build and maintain. However, compared to traditional host-based installations, cloud computing can be a relatively homogeneous and specialized environment; this lends itself to a white-listing model. Any application-monitoring solution should apply a white list, rather than a blacklist, for the applications that are permitted to run.

Distributed network-monitoring solutions come in several flavors, and several of the functions monitor essentially application-level problems. There are many applications that you can install internally to monitor the state of the network and its hosts. Some of these, such as the open source tool Nagios (www.nagios.org), allow for the various hosts on the network to report many kinds of statistics to the collection point. Several companies have also developed and are selling distributed-computing or cloud-computing monitoring solutions, such as

- distributed availability and performance monitoring (for example, Alertra; www.alertra.com) and
- security and change management software (for example, AccelOps; www.accelops.net/product/data-center-monitoring.php).

For a list of available systems and a comparison of features, see http://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems.

With a customer-controlled application layer, the only change from a traditional computing model is that because the monitoring will also be virtualized, it allows for a more fine-grained cost-benefit analysis of monitoring different attributes. This is due to the cloud architecture's inherent scalability and flexibility. For SaaS, the provider might develop custom monitoring solutions; however, it should be able to describe those monitoring and remediation strategies in detail, as Google has.[7]

Users

If the cloud just provides services for distributing public information, such as a webpage or video, the user's consumption has little security impact. However, if the users are members of the customer organization, they're integral to the security policy. In general, personnel policies for cloud-based resources are no different from the user perspective. Indeed, that's the goal—to deliver services to users that are identical to or better than their local services more consistently and with broader access.

Access patterns can be monitored for malicious behavior. For example, Google Apps monitors login behavior such as the time and IP address, makes this information available to the user, and

notifies the user of aberrant behavior. This idea could be extended to make digests of such alerts available to IT managers about the accounts for which their organization is responsible.

In addition, the customer might need to make an addendum proscribing access to sensitive data in public areas. As always, an ignorant, authorized user can demolish many security considerations in a few clicks because Web browsers and applications have many vulnerabilities to exploit. Programs and services exist to aid user education, such as PhishGuru (www.wombatsecurity.com/phishguru), although they aren't cloud specific. Nevertheless, user education remains extremely important.

Monitoring cloud computing is complex; although outsourcing the decision might appear to simplify the problem, it might raise as many questions as it answers. It introduces yet another party into the circle of organizations that must be trusted with the data. It's another relationship that the organization must monitor, and although small organizations might have sufficiently limited resources to justify that choice, medium and large organizations should decide for themselves what aspects of their cloud environment to monitor. Outsourcing doesn't identify what precisely to monitor, which is the important question. Any SLA that a customer enters into with a cloud provider should clearly demarcate which of the described monitoring and auditing responsibilities are to be undertaken by which party and what reports will be made with what frequency and specificity. Framing the question in regard to the cloud's layers facilitates a more manageable answer. □

Acknowledgments

The information and views in this article don't necessarily reflect the view or opinion of the Carnegie Mellon University Software Engineering Institute.

References

1. A. Ranabahu and E.M. Maximilien, "A Best Practice Model for Cloud Middleware Systems," *Proc. Best Practices in Cloud Computing: Designing for the Cloud Workshop*, ACM Press, 2009, pp. 41–51; http://knoesis.wright.edu/library/download/cloud-oopsla-09.pdf.
2. S.N. Foley et al., "A Framework for Heterogeneous Middleware Security," *Proc. 18th Int'l Parallel and Distributed Processing Symp.* (IPDPS 04), Workshop 1, IEEE CS Press, 2004.
3. W. Dorman, "CERT Basic Fuzzing Framework," blog, 26 May 2010; www.cert.org/blogs/vuls/2010/05/cert_basic_fuzzing_framework.html.
4. "Zzuf—Multi-purpose Fuzzer," CaCa Labs, 2011; http://caca.zoy.org/wiki/zzuf.
5. E.M. Maximilien et al., "IBM Altocumulus: A Cross-Cloud Middleware and Platform," *Proc. 24th ACM SIGPLAN Conf. Companion on Object-Oriented Programming Systems Languages and Applications* (OOPSLA 09), ACM Press, 2009, pp. 805–806.
6. L.M. Kaufman, "Data Security in the World of Cloud Computing," *IEEE Security & Privacy*, vol. 7, no. 4, 2009, pp. 61–64.
7. "Security Whitepaper: Google Apps Messaging and Collaboration Products," white paper, Google, 2010; www.google.com/a/help/intl/en/admins/pdf/ds_gsa_apps_whitepaper_0207.pdf.

Jonathan Spring *is a member of the technical staff in the Software Engineering Institute's CERT program. Contact him at jspring@sei.cmu.edu.*