SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

CSA cloud security alliance SM

Practices for Secure Development of Cloud Applications

PRIMARY AUTHORS:
Bryan Sullivan, Microsoft
Said Tabet, EMC
Edward Bonver, Symantec
Judith Furlong, EMC
Steve Orrin, Intel
Peleus Uhley, Adobe Systems, Inc.

DECEMBER 5, 2013

Table of Contents

I. Foreword

The emergence of cloud computing as a new paradigm brings with it a lot of challenges and issues that require close attention by the industry. While information technology continually goes through changes, cloud computing is one of the most significant, bringing new innovation and with it new requirements. For cloud computing to reach its true potential, all parties involved – both consumers and providers – will need new ways of thinking about security needs and related standards. Cloud consumers have legitimate concerns about security risk related to cloud computing, including those related to the 'outsourcing' of the security of their information systems. Cloud service providers are working to provide solutions and ways to mitigate such risks. A cohesive and common approach is only just beginning to emerge. Research being conducted by the Cloud Security Alliance (CSA), the National Institute for Standards and Technology (NIST), and standards development organizations such as ISO, ITU-T and others, is helping to bridge the gaps.

At the same time, the recognition that security must be addressed throughout the software development lifecycle has grown alongside industry efforts to advance the current state of software security. The Software Assurance Forum for Excellence in Code (SAFECode) is one industry-led organization working to identify and promote best practices for secure software development. In the five years since SAFECode released its flagship work identifying fundamental secure development practices, the process of building secure software has continued to evolve alongside innovations and advancements in the information and communications technology industry. One key question is how the emergence and maturation of cloud computing has impacted the security development lifecycles of leading technology providers.

To address this question, SAFECode and CSA partnered to determine whether additional software security guidance was needed to address unique threats to the cloud computing, and if so, to identify specific security practices in the context of identified threats. This report represents the product of that collaboration and is intended to help readers better understand and implement best practices for secure cloud software development.

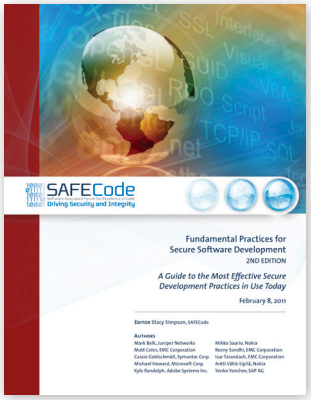
This paper should not be considered a replacement for SAFECode's "Fundamental Practices for Secure Software Development" document, but rather as a supplement to it that provides additional secure development recommendations in the context of critical threats specific to cloud computing.

Methodology and Scope

SAFECode and CSA created a working group with practical experience in both cloud and software security. The team reviewed existing secure software development practices and secure design considerations as outlined in the SAFECode publication "Fundamental Practices for Secure Software Development 2nd Edition" in the context of CSA cloud security guidance, including its paper, "The Notorious Nine: Cloud Computing Top Threats in 2013."

While the working group's efforts confirmed that each practice identified by SAFECode as fundamental to software security applied equally to cloud software, it did identify additional practices that should be adopted by those developing software for the cloud given the unique threats faced in that domain. As such, this paper should not be considered a replacement for SAFECode's "Fundamental Practices for Secure Software Development" document, but rather as a supplement to it that provides additional secure development recommendations in the context of critical threats specific to cloud computing.

The working group focused its efforts on analyzing security considerations for Platform as a Service (PaaS) model of cloud computing. This seemed like the most logical starting point given that PaaS is often the intersection between the Infrastructure as a Service (IaaS) and Software as a Service (SaaS) models. The Platform runs on an infrastructure that needs



Now in its second edition, SAFECode's flagship paper, Fundamental Practices for Secure Software Development: A Guide to the Most Effective Secure Development Practices in Use Today, provides a foundational set of secure development practices that should be applied during the design, development and testing activities in the software development lifecycle. The practices identified and explained in the report have been shown to be effective in improving software security in real-world implementations by SAFECode members even across diverse development environments. The full report can be downloaded at no cost by visiting www.safecode.org.

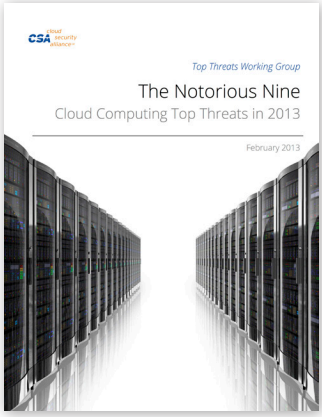
to provide security protections and security services to the PaaS. Additionally, SaaS and application developers will rely on the PaaS to provide them with security features, APIs and services. Given this relationship, much of the guidance provided in this paper is relevant to the application or services layers, even though the primary target of work is PaaS security.

Just as in traditional software development practices, it is crucial to consider security at each layer and at each step of the development lifecycle. Accordingly, the target audience for this document includes not only software developers, software and security architects and designers, but also managers and IT leads who need to be aware of both the risks and mitigations this document describes.

To aid others in adopting and using these practices effectively, this paper describes each identified security practice in the context of the unique attributes of cloud computing and the associated threats as identified by CSA. We have mapped our recommended practices to specific threats in order to provide a more detailed illustration of the security issues these practices aim to resolve and a starting point for those wishing to learn more. Each section offers specific action items for development and security teams, as well as useful references that provide additional implementation guidance.

II. Threats to Cloud Computing

In order to develop secure software, it is crucial for development teams to model potential threats to their applications and to implement mitigations to those threats. Developing for a PaaS cloud environment exposes certain unique risks that development teams need to understand and address. In this section, we will describe some of these risks to provide a clearer picture of the security issues our recommendations aim to help mitigate. While many of these threats extend to both the Infrastructure and Software layers, we've focused this section on those most relevant to the Platform layer.



The Notorious Nine: Cloud Computing Top Threats in 2013 aims to provide organizations with up-to-date, expert-informed understanding of cloud security threats in order to make educated risk-management decisions regarding cloud adoption strategies.

While there are many risks associated with the cloud in general, this report focuses on threats specifically related to the shared, on-demand nature of cloud computing. It serves as an up-to-date threat identification guide that will help cloud users and providers make informed decisions about risk mitigation within a cloud strategy.

Data Breaches

Readers may be familiar with the attack technique known as "SQL injection," in which attackers can manipulate SQL queries, for example in order to extract the credit card numbers of all customers in a database[1]. As damaging as SQL injection attacks can be against web applications, their consequences in multitenant cloud environments can be even worse. If multiple applications share the same underlying database system, a flaw in any one of the clients' applications could potentially put all of the other customers' data at risk. Broadening the credit card theft example, in a cloud environment an attacker might be able to extract the credit card numbers of all the customers whose data happens to be located in an underlying database that is shared among multiple clients of the same cloud provider.

In another, more complex example, researchers have demonstrated that they could extract private cryptographic keys from co-hosted virtual machines using a side-channel attack (http://arstechnica.com/security/2012/11/crypto-keys-stolen-from-virtual-machine/). Additionally, current trends in malware and attack techniques have malicious actors targeting lower levels of the stack such as hypervisors, virtual infrastructure and BIOS. In co-hosted environments, compromised virtual infrastructure and BIOS's pose a serious threat to the data and applications running on co-tenant cloud PaaS infrastructure.

Data Leakage and Data Loss

Data needs to be protected in transit and at rest. Protecting data in transit might mean securing the communication channel between physical systems (e.g., server-to-server or server-to-client), physical resources on any given machine (e.g., interprocess communications), virtual machines, virtual network interfaces, etc. When it comes to virtual machines, the development team needs to be aware of additional threats, such as VM escape (vulnerabilities in the hypervisor that let attackers take control over the VM's underlying infrastructure), VM hopping (when vulnerabilities in the hypervisor let attackers gain access from one virtual machine to another one that they do not own), and insecure VM migration (where, for example, an attacker can force the VM to migrate to an untrusted host that they have control over, or can access data illegally during the migration process).

When data is kept in the cloud, the system needs to be designed, implemented, and deployed so that it can withstand attacks on various levels in the multi-tier architecture. One of the most obvious examples is if an attacker breaks in and modifies or deletes data (i.e.,

1 If you are unfamiliar with SQL injection, please refer to the SAFECode publication "Fundamental Practices for Secure Software Development 2nd Edition"; particularly the sections "Avoid String Concatenation for Dynamic SQL Statements" and "Validate Input and Output to Mitigate Common Vulnerabilities."

attack on the integrity of data). In this case, is it possible to trace what happened during the attack, roll back malicious actions and/or restore from a backup? If restore operations need to be performed, which entity is responsible for maintaining the backups (e.g., is it the cloud provider) and can the integrity of backups be guaranteed? Going beyond tracing the attack, were the logs maintained in accordance with privacy protection laws, which differ from country to country, and depending on where and how the application is deployed? Which country's privacy laws would apply in this case?

Using encryption in order to protect confidentiality of data seems like a logical choice; however, encryption presents its own set of challenges, such as key management, correct use of crypto algorithms and libraries, and again, country-specific laws. Here, new questions arise: at which layer(s) should encryption be performed? Who is in charge of the keys? How can data be accessed when a key is lost? What if a key gets compromised?

Some of the responsibilities may lie on the provider's side, some on the operational side, while others lie on the development team's side. For example, a provider might be in charge of storing VM snapshots. Are those snapshots stored securely (is proper authentication and authorization in place)? Is data confidentiality preserved throughout the VM's lifecycle?

Insecure Interfaces and APIs

Quite frequently as part of cloud services, a cloud provider exposes a set of interfaces or APIs that allows their customers to perform various operations (e.g., monitoring, provisioning, management, etc.) Much of the security of the system depends on whether those APIs are designed properly (e.g., access control, authentication, etc.), and if they are used correctly by customers. In this case correct usage can be expressed as following the provider's instructions and by following best practices associated with using such APIs in distributed/cloud environments. Special care needs to be taken when using such APIs, especially given the fact that there might be multiple layers of APIs, as third-parties are known to extend APIs and build new services on top of them.

Denial of Service

In cloud environments, denial of service attacks might occur at various layers. Examples include attacks on the cloud provider's systems (virtual machines, memory, disk space, network, database tier, etc.).

Generally speaking, the more layers that are introduced in an architecture, the more attack surface is also introduced. This is particularly true for denial of service attacks compared to other attacks such as data theft or elevation of privilege. For example, it might be very difficult for an attacker to exfiltrate data from a database component that resides many layers deep in the architecture. However, a denial of service attack against any single layer – such as forcing a backend NoSQL database into an infinite loop by sending a malformed query, or causing an XML configuration file parser to hang by sending it a "billion laughs attack" exponential entity expansion – may be sufficient to take down the entire system.

It is important to leverage resources, capacity and defenses that might be offered by cloud providers. Examples include load balancing, traffic management, active failover, scaling, backup, etc.

References

- The Notorious Nine: Cloud Computing Top Threats in 2013; Cloud Security Alliance; February 2013
  https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf
- Journal of Internet Services and Applications; An analysis of security issues for cloud computing; Keiko Hashizume, David G Rosado, Eduardo Fernández-Medina and Eduardo B Fernandez; 2013
  http://www.jisajournal.com/content/pdf/1869-0238-4-5.pdf
- Ars Technica; Virtual machine used to steal crypto keys from other VM on same server; Dan Goodin; Nov. 6, 2012
  http://arstechnica.com/security/2012/11/crypto-keys-stolen-from-virtual-machine/
- Fundamental Practices for Secure Software Development 2ND EDITION: A Guide to the Most Effective Secure Development Practices in Use Today; SAFECode; February 8, 2011
  www.safecode.org/publications/SAFECode_Dev_Practices0211.pdf
- Cloud Security: Attacks and Current Defenses (by Gehana Booth, Andrew Soknacki, and Anil Somayaji) from the 8th Annual Symposium on Information Assurance (ASIA'13)
  http://www.albany.edu/iasymposium/proceedings/2013/16-BoothSoknackiSomayaji.pdf

III. Design Issues

Multitenancy

Description

One of the key distinguishing features of cloud computing is multitenancy. Multitenancy allows multiple consumers or *tenants* to maintain a presence in a cloud service provider's environment, but in a manner where the computations and data of one tenant are isolated from and inaccessible to another tenant. In a multitenant environment:

- No tenant should be able to identify or determine the existence of other tenants;
- No tenant should have access to the data of another tenant;
- No tenant should be able to perform an operation that affects the operation of or denies service to another tenant;
- Configuration for each tenant should be independent from that of another tenant;
- Auditing and tracing should be provided on a per tenant basis; and
- Provisioning and decommissioning of tenants should be done in a manner that enforces tenant segregation.

Multitenancy has architectural and design impacts on the cloud service provider's environment and on the applications that run in multitenant environments.

From a cloud service provider perspective, the fundamental design principle for multitenancy is "logically separate, but physically shared." To tenants, the cloud service provider's environment must appear to provide a logical separation among tenants, but in fact the underlying infrastructure used by the cloud service provider is typically physically shared among tenants. In order to offer secure service to its tenants, the cloud service provider needs to employ physical, technical and procedural means to secure the underlying infrastructure from attackers. The provider will also rely upon specific software and hardware components within the infrastructure to provide multitenant capabilities such as the processing and segregation of tenant information. Other components may exist within the provider's infrastructure, but not be trusted to provide tenant segregation on their own. For example, the infrastructure may include shared storage, but the data from multiple tenants stored within it may be encrypted (a key per tenant) prior to being written to this shared storage.

Multitenancy considerations also influence how the cloud service provider provisions tenants into their environment and the type of service interfaces that they provide to their tenants. Most cloud service providers offer a range of services to their customers, each with its own associated policies and Service Level Agreements (SLAs). It may not be technically feasible or cost effective to provide a full range of multitenant offerings on a single infrastructure. Instead, the provider may choose to implement a separate infrastructure for each distinct service and deploy tenants with similar policy and SLA requirements on the same infrastructure.

The type of interface provided to a tenant may also vary based on the cloud service provider offering. The interfaces may expose different views of the provider's (PaaS) stack to a tenant application developer. Some may be at a high level of abstraction such as an API, while others may be at a lower level such as an object or file. The provider needs to ensure that it maintains and presents logical tenant separation at all the interfaces it chooses to expose. A similar rule applies to any common services (e.g., security, management and reporting)

that the cloud service provider offers to its customers. These common services need to be designed and deployed in a way that ensures that the tenant segregation is maintained.

While it is certainly possible to deploy a single tenant application into a multitenant environment by deploying multiple instances of the application (one per tenant), it is more interesting to develop applications which are mulititenant-aware and capable of leveraging the benefits of such an environment. However, developers of multitenant-aware applications have additional considerations to keep in mind. These considerations are explored in more detail in the following database design example.

In an example of multitenancy, multiple organizations can share a common application without having access to each other's data. Multitenancy can allow vast improvements to an application's scalability, but it is critical for the application to ensure that no tenant can accidentally or intentionally gain access to another tenant's data. Two companies may be competitors yet still have their data stored side-by-side on the same multitenant database; if one of them were able to read the other's data (or worse yet, write over it), trust in the system and in cloud computing itself would be undermined.

The easiest way to implement database multitenancy is in a "shared schema" approach, where you simply add a "TenantID" column to each data table:

TenantID	ProductName	QuarterlySales
5991	Laptop Model 3200	297500
6012	Tablet Model 3Z	99300
5991	Phone Model 3700	154800
…	…	…

Unfortunately, while this is the easiest approach to implement, and the most scalable, it is also the most difficult to secure. Developers would have to ensure that each and every data access call, whether via stored procedure, parameterized query or ad-hoc query, would include a filter for the appropriate TenantID clause. Even a single instance where the TenantID clause is missed could create a data disclosure or tampering vulnerability. A potential mitigation would be to create a data view filtered by the current user's tenant ID and to always query against that data view rather than directly against the table; however, this only reduces the problem rather than solving it, as the developer now has to always remember to query the view rather than the table. Setting appropriate database permissions (such as removing direct access to the tables) can help solve this problem too, but overall a shared schema approach is fragile and is not recommended.

A better approach is a "separate schema" design, in which the provider provisions each tenant with its own set of tables that store only that tenant's data. This design sounds like it would be just as difficult to secure – that developers would now have to remember to add clauses for table names instead of column names – but many database engines support the concept of named schemas that make this alternative transparent to the developer. Continuing the earlier example, there may be completely separate "Tenant5991.ProductSales" and "Tenant6012.ProductSales" tables in the database, but the application only needs to set the default schema for a user at the time the user account is created. The application can then simply query against a default "ProductSales" table. Again, this makes the multitenancy transparent to the tenant developers, and better still, helps to ensure that they can neither accidentally nor intentionally access other tenants' data.

Action Items

- For providers: Model all of your application's interfaces in threat models. Ensure that multitenancy threats such as information disclosure and privilege elevation are modeled for each of these interfaces, and ensure that the threats are mitigated in the application code and/or configuration settings.
- For providers: Use a "separate schema" database design when building multitenant applications instead of adding a "TenantID" column to each table.
- When developing applications that leverage a cloud service provider's (PaaS) services, be aware of and take advantage of how the interfaces into the PaaS stack provide tenant segregation.

CSA Reference

CSA Top Threat 1: Data Breaches

CSA Top Threat 2: Data Loss

References

Books, Articles and Reports:

- Multitenant Data Architecture; http://msdn.microsoft.com/en-us/library/aa479086.aspx

Trusted Compute Pools

Description

Trusted Compute Pools (TCP), also referred to as trusted pools, are either physical or logical groupings of compute resources/systems in a data center that share a security posture. These systems provide measured verification of the boot and runtime infrastructure, typically the BIOS and VMM/Hypervisor or OS using protocols and methods as described in the TCG (Trusted Computing Group) standards[1] and the NIST Interagency Report 7904[2] for measured launch and trust verification. The measurements are stored in a trusted location on the system (usually a TPM) and verification occurs when an agent, service or application requests the trust quote from the TPM. The measurements can then be validated via a verification service or system that verifies the measurements against known good values as well as validity of the signature of the quote. The use and propagation of the "Platform Trust" attribute(s) by datacenter/cloud management and security systems allows for the establishment of Trusted Compute Pools that can be used to assign workloads/applications to trusted systems as well as to audit of the trust for a given system. TCP is used to aggregate trusted systems and segregate them from untrusted resources, which results in the separation of higher-value, sensitive workloads from commodity applications/workloads.

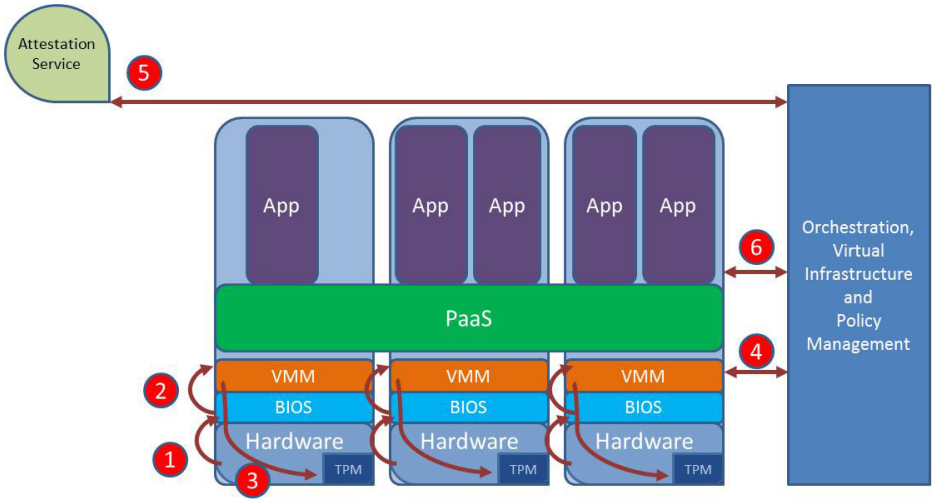


Figure 1. Trusted Compute Pools

In the above diagram (Figure 1), the measured launch sequence and storage of the trust values is illustrated in steps 1-3. In steps 4 and 5, the Virtual infrastructure (and/or orchestration system or policy management system) verifies the trust measurements with the Attestation Service. Finally, in step 6, VMs and Applications can be provisioned/migrated based on policies and trust.

A PaaS environment can expose trust verification APIs to the applications that run on it as illustrated in Figure 2 below.

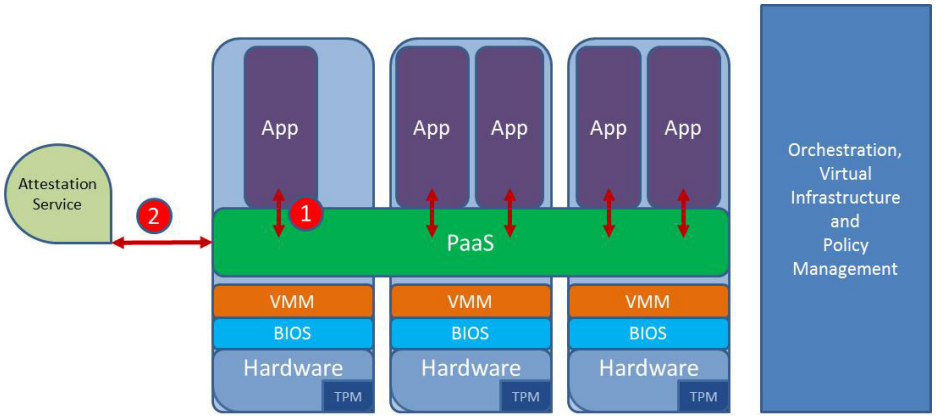


Figure 2. Trusted Compute Pools and PaaS Applications

The PaaS APIs exposed should follow the TCG specifications for TPM quoting or leverage a local agent to abstract them. The verification service APIs can either be provided as part of the platform, the virtualization/cloud infrastructure or as separate Representational State Transfer (REST) APIs. There are examples available from both the Virtualization vendors and the OpenStack: Open Attestation SDK project[3].

An example of a JavaScript Object Notation (JSON) based request to an attestation service and its response is shown in Figure 3[4].

Request	Response
POST AttestationService/resources/PollHosts Host: Attestation.ras.com:8443 Context-Type: application/json Accept: application/json Auth_blob: authenticationBlob	HTTP/1.1 200 OK Server: BaseHTTP/0.3 Python/2.7.1+ Date: Mon, 15 Oct 2012 22:36:58 CST Context-Type: application/json

{
“hosts”: [
“host1”,
“host2”
]
}
	{
	"hosts":[
	{"host_name":" host1","trust_lvl":"trusted",
	"vtime":"2012-10-15T22:36:58.836+08:00"},
	{"host_name":" host2","trust_lvl":"trusted",
	"vtime":"2012-10-15T22:36:58.836+08:00"}
	]
	}

Figure 3. Attestation Service Request Example

Action Items

- Ensure that your platform for developing cloud applications provides trust measurement capabilities and the APIs and Services necessary for your applications to both request and verify the measurements of the infrastructure they are running on.
- Verify the trust measurements either as part of the initialization of your application or as a separate function prior to launching the application.
- Audit the trust of the environments your applications run on using attestation services and/or capabilities such as the OpenAttestation (OAT) SDK, products that leverage OAT or native attestation features from your infrastructure provider.

CSA Reference

CSA Top Threat 8: Insufficient Due Diligence

CSA Top Threat 9: Shared Technology Vulnerabilities

References

Books, Articles and Reports:

[1] TCG Standards and Specifications: http://www.trustedcomputinggroup.org/

[2] NIST Interagency Report 7904: Trusted Geolocation in the Cloud: Proof of Concept Implementation (Draft) http://csrc.nist.gov/publications/drafts/ir7904/draft_nistir_7904.pdf

[3] OpenStack: Open Attestation SDK https://wiki.openstack.org/wiki/OpenAttestation

[4] API Example from OpenStack OpenAttestation SDK: https://github.com/OpenAttestation/OpenAttestation/blob/59cbf7853db3f50268e8983706cf628e10bab937/docs/Overview.pdf

Tokenization of Sensitive Data

Description

Sensitive Data

Data is considered sensitive if it should not be made known to the general public and should only be made known within a specific context. Data may be deemed sensitive by an organization, an industry or a government and each of these entities would specify the requirements for how to handle and protect the sensitive data. Examples of sensitive data include:

- Intellectual Property (IP) and other business sensitive data (e.g., customer lists)
- Payment Card Industry (PCI) data such as a credit card number
- Personally Identifiable Information (PII) such as a U.S. Social Security Number (SSNs), and
- Protected Health Information (PHI) such as medical records

Depending upon the type of sensitive data and the requirements for protecting that data, it may not be possible to generate or store the data within cloud environments. For example, an organization may not wish to generate and store intellectual property in a cloud environment not under its control. There may also be government or regional requirements that may prevent sensitive data from being stored outside the boundaries of the country of origin and would restrict the data to be generated and stored within a cloud environment that is operating in that country of origin.

The most common recommendation is to protect sensitive data as close as possible to its point of origin, which is normally within the application where the data is generated. But it is possible to protect data at other points, such as when data is transmitted outside of an organization's boundary, or when it is written out to storage. In many cases, sensitive data may be protected at multiple points and using multiple methods as it is transmitted and stored within an IT environment. This is particularly true of cloud computing environments where all or part of the environment is not under the control of the originator/owner of the sensitive data.

Encryption is often the mechanism used to protect sensitive data and is discussed in more detail within the "Data Encryption and Key Management" section of this paper. But other approaches to the protection of sensitive data include removing the sensitive data from systems where it does not need to exist or disassociating the data from the context or the identity that makes it sensitive. Tokenization is an example of removing sensitive data; where the sensitive data is replaced with a token or alias for that data. Data Masking is an example of disassociating the data from the context or the identity that makes it sensitive. Both approaches are discussed in more detail in this section.

Tokenization

Tokenization was initially introduced by the Payment Card Industry as a means to protect credit card information, but tokenization is now used to protect all types of sensitive data.

In an example tokenization solution, the sensitive data (e.g. credit card number) is encrypted when it is created or captured and transmitted to a secure central system. The system generates a substitute value for the sensitive data, known at the *token*, that is used as a replacement for the sensitive data in the environment in which it is processed and stored. Tokens may be single-use or multiple-use and which is chosen is based on the nature of the sensitive data being replaced and whether the sensitive data needs to be tracked as it is passed in the environment.

The sensitive data is secured within the central system which can be protected with multiple layers of protection and appropriate redundancy for disaster recovery and business continuity. Access to the sensitive data can be limited to applications within the environment that need to process the actual sensitive data. In that case the token can be exchanged at the central system for the corresponding sensitive data.

Multiple deployment options exist for tokenization solutions. Tokenization may be offered as an on-premise solution such as one deployed within a merchant environment or it may be offered as a service (SaaS) by a provider such as a payment processor.

Advantages of the tokenization approach include:

- Reduction in the risk of exposure of the sensitive data, since it is resident in fewer places.
- Systems that don't contain the actual sensitive data may not need to comply with data protection requirements defined by organizational, industry and government regulations.
- The impact to applications to support tokenization is often much less than that of other data protection methods (e.g. encryption), since the token size and format can be adjusted to fit the existing data fields of the sensitive data being replaced.

Token Generation

Tokens must be generated using a method where if someone is in possession of only the token it is computationally infeasible to recover the sensitive data that the token represents. Methods used to generate tokens include the use of cryptographic algorithms or one-way irreversible functions (e.g., hash functions). Token generation implementations are normally configurable allowing for the token size and format to be adjusted to meet the data format and schema requirements of the application into which the token will be substituted for the sensitive data.

Token Mapping

To access the sensitive data associated with a token, a mapping between a token and the sensitive data needs to be maintained. This mapping table is normally stored in the same system where the sensitive data is securely stored to ensure that appropriate protections are also applied to the mapping information itself. When the sensitive data need to be retrieved, the token is provided and used as an index into the mapping table to find the link to the associated sensitive data.

Data Masking

Data masking is an approach that disassociates data from the context or the identity that makes the data sensitive. In contrast with tokenization, which replaces entire pieces of sensitive data, data masking techniques involve replacing or obfuscating portions of a data set.

Data masking is an approach that is often used in pre-production test system or in systems used to debug a software application where one needs to work with a representative data set, but does not need to have access to actual sensitive data. This approach allows the test and debug systems to be exempt from sensitive data protection requirements.

Action Items

- When designing a cloud application, determine if the application needs to process sensitive data and if so, identify any organizational, government, or industry regulations that pertain to that type of sensitive data and assess their impact on the application design.

- Whenever possible, organizations should minimize the applications and or systems that need to process and store sensitive data. Consider implementing tokenization to reduce or eliminate the amount of sensitive data that needs to be processed and or stored in cloud environments.
- Use data masking in development, continuing engineering and pre-production environment to exempt these environments from data protection requirements.

CSA Reference

CSA Top Threat 1: Data Breaches

CSA Top Threat 2: Data Loss

CSA Top Threat 6: Malicious Insiders

References

- Payment Card Industry Data Security Standard (PCI DSS), Version 2.0, October 2010 https://www.pcisecuritystandards.org/documents/pci_dss_v2.pdf
- Payment Card Industry Data Security Standard (PCI DSS), Version 2.0, Information Supplement: PCI DSS Tokenization Guidelines, August 2011 https://www.pcisecuritystandards.org/documents/Tokenization_Guidelines_Info_Supplement.pdf
- Tokenization: What's Next After PCI?, 2012, EMC Corporation http://www.emc.com/collateral/white-papers/h11918-wp-tokenization-rsa-dpm.pdf
- Visa Best Practices for Tokenization Version 1.0, 14 July 2010 http://usa.visa.com/download/merchants/tokenization_best_practices.pdf

Data Encryption and Key Management

Description

Data Encryption

Encryption is the most pervasive means of protecting sensitive data and may be applied in multiple scenarios. Encryption applied as data is moved within an environment is referred to as *data-in-motion* (or *data-in-transit* or *data-in-flight*), while encryption applied to the data resident within an application or infrastructure component is referred to as *data-at-rest*. Data-in-motion and data-at-rest solutions may be implemented in a number of ways.

Data-in-motion is normally provided via secure communication protocols such as TLS/SSL or IPSec. Within cloud environments secure communications are often implemented as part of the infrastructure, protecting data as it as it moves between infrastructure components, and as it passes between the enterprise and the cloud provider and vice versa.

Data-at-rest solutions are more varied since data may be represented in many forms depending upon how it is being created, processed and stored within an environment. Within an application, data may be kept in a database or in a file, so encryption schemes that encrypt data within a database or which encrypt files may be employed. In another approach, data may be encrypted as it is written to storage. This encryption may occur at the data leaves the host, within a switch within the network, or the entire disk within the storage array may be encrypted. Given the diversity of the data-at-rest implementations, these solutions may be available as part of the infrastructure or as part of an application running in the cloud. It may even be possible that multiple data-at-rest solutions may be applied within a single environment.

Key Management

Once encryption capabilities are introduced into an application or environment, it is important for both providers and tenants to ensure that the associated cryptographic key materials are properly generated, managed and stored. This is particularly the case for data-at-rest encryption solutions where the loss of a data encryption key results in data loss since the encrypted data can no longer be decrypted. Similarly the compromise of a data encryption key can result in the exposure of the sensitive encrypted data. To address these concerns, appropriate key management solutions to support data-at-rest or data-in-motion implementations should be employed. Key management solutions also ensure that the appropriate policies are applied to the cryptographic keys including how they are generated, how often the keys need to be changed and how to handle the compromise of a key.

For secure communication protocols such as TLS/SSL and IPSec, which are often used to provide data-in-motion protection, key management is in part built into the protocol. In these protocols, the data encryption key (a symmetric key) used to protect a specific communication channel is negotiated when the channel is first established. However, these negotiations rely upon the presence of asymmetric key materials and associated public key certificates (i.e., X.509 public key certificates) installed at each end-point of the communication channel. These asymmetric keys and certificates are generated and managed via a Public Key Infrastructure (PKI). A PKI could be operated by the enterprise, the cloud service provider or a trusted third-party.

In many cloud environments, all three types of PKIs may co-exist. For example, the cloud service provider may use its own PKI to manage the keys and certificates used to secure communications between infrastructure components. Communications from the enterprise to the cloud could be secured using credentials issued by the enterprise and the cloud service provider, where the appropriate trust relationships are put in place between the enterprise and the service provider PKIs. Alternatively credentials issued by a trusted third-party PKI could be used to secure the communication channel between the enterprise and the cloud where embedded trusted Root Certification Authority (CA) credentials within the enterprise and the service provider end points are leveraged.

Data-at-rest solutions are often based on symmetric cryptography and hence need to leverage symmetric key management techniques. In these solutions, the key use used encrypt the data is referred to a data encryption key (DEK) and this DEK is protected (or wrapped) using a key encryption key (KEK). Symmetric key management solutions provide management of both DEKs and KEKs. Key management functionality could be located within the application or infrastructure component where the encryption is implemented or the functionality could be split between the application (which incorporates a key client) and a central key server/repository. The advantages of a centralized key server are that policy controls can be implemented in one place; multiple encryption capable applications can be supported; the same key material can be shared between applications; and the server can be designed for redundancy and disaster recovery. Given these advantages, it is likely that centralized key management solutions will be employed within cloud environments.

The location of the central key server may vary based upon the type of data-at-rest solution and the nature of the cloud environment. An application which bridges the enterprise and cloud may leverage the key server located within the enterprise, while a storage disk encryption solution running inside the cloud service provider's data center may leverage a key server in the same data center. It is also possible for a service provider to operate a key management service in the cloud to support other cloud offerings. The emergence of interoperable key management protocols such as the OASIS Key Management

Interoperability Protocol (KMIP) make it easier for cloud developers to build solutions which can work in any of these cloud deployment scenarios.

Encryption and Key Management Considerations

Introduction of encryption and key management solutions into the cloud environment brings with it additional regulations, compliance requirements and technical design considerations of which developers of cloud solutions should be aware. These considerations include the following:

- **Import/Export Regulations:** Ensure that any technical solution incorporating encryption and key management functionality complies with the import and export regulations for the countries in which the solution is developed and in which the solution is operated.
- **Compliance with Government or Organizational Key Management Requirements:** Ensure that the encryption and key management solution accommodates for any organizational or government requirements for the back-up, archive or escrow of encryption keys necessary to enable recovery of encrypted data.
- **Use of Strong Cryptographic Algorithms:** Choose algorithms, algorithm modes and key lengths that are considered strong (not currently vulnerable to known cryptographic attacks). Whenever possible, design solutions which enable algorithms, modes and key lengths to be configurable. This will allow the solution to dynamically disable algorithms which are no longer considered strong or to accommodate for specific algorithm requirements specified by a government or an industry. Please refer to the "Eliminate Weak Cryptography" section of the SAFECode document "Fundamental Practices for Secure Software Development" for more detailed information on which cryptographic algorithms, modes and key lengths are considered secure.
- **Use of FIPS 140-2 or Common Criteria Validated Cryptographic Modules:** Some customers or market verticals may require that encryption and key management solutions leverage cryptographic modules (software or hardware) that have been FIPS 140-2 validated or Common Criteria certified.
- **Leverage Key Management Standards and Protocols:** Whenever possible, implement key management standards and protocols (e.g., X.509, RFC5280, OASIS KMIP) to ensure maximum interoperability in cloud environments where products from multi-vendors are likely to be employed.
- **Interaction of Encryption with Other Solution Features:** Encryption of data-in-motion or data-at-rest may prevent other solution functionality from being implemented or may require that certain functionality be implemented before or after encryption is applied. For example, it would not be possible to search encrypted data for key words or it may be necessary to apply compression or de-duplication techniques before encrypting the data.
- **Choosing the Appropriate Encryption Solution:** Different types of cryptographic algorithms (e.g., block cipher vs. streaming) or algorithm modes may be more suited for use in a particular solution. Developers may trade-off choices based on criteria such as performance or the need to keep the size of the encrypted data the same as that of unencrypted data.

Action Items

- When developing an application for the cloud, determine if cryptographic and key management capabilities need to be directly implemented in the application or if the application can leverage cryptographic and key management capabilities provided by the PaaS environment.

- In order to protect and have access to encrypted data, make sure that appropriate key management capabilities are integrated into your application to ensure continued access to data encryption keys, particularly as data moves across cloud boundaries (enterprise to could, public to private cloud, etc.).
- Developers should be aware of and act upon the encryption and key management considerations outlined in this section.

CSA Reference

CSA Top Threat 1: Data Breaches

CSA Top Threat 2: Data Loss

References

- Common Criteria
  http://www.commoncriteriaportal.org/
- D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, RFC5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008.
  http://www.ietf.org/rfc/rfc5280.txt
- E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, Recommendations for Key Management - Part 1: General (Revision 3. NIST Special Publication 800-57 part 1, July 2012.
  http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf
- International Telecommunication Union (ITU)–T, X.509: Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks, August 2005.
  http://www.itu.int/rec/T-REC-X.509-200508-S/en
- National Institute of Standards and Technology (NIST), Federal Information Processing Standards Publication (FIPS) 140-2, Security Requirements for Cryptographic Modules, May 25, 2001.
  http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
- Organization for the Advancement of Structured Information Standards (OASIS), Key Management Interoperability Protocol Specification, Version 1.1, 24 January 2013, OASIS Standard
  http://docs.oasis-open.org/kmip/spec/v1.1/kmip-spec-v1.1.doc

Authentication and Identity Management

Description

A thorough discussion of the topics of authentication and identity management would extend well beyond the scope of this paper; however, given that proper authentication and identity management are critical to secure cloud application development, we are providing some high level secure authentication and identity management design principles. If you would like to explore these topics in more detail, please refer to the documents listed in the References section below.

Authentication

It is important to consider authentication methods when designing cloud applications, both from the point of view of the cloud application as an authentication consumer and as an authentication provider. As an authentication consumer, the application may need

authenticate itself to the PaaS in order to access interfaces and services provided by the PaaS. As an authentication provider, the application may need to authenticate the users of the application itself. Because the authentication methods are likely to be different for both the consumer and provider scenarios, it is likely that a single cloud application will need to support multiple authentication methods.

For authentication to the PaaS service (i.e., the cloud application as an authentication consumer), PaaS services commonly support public key certificate-based authentication and directory-based authentication methods such as LDAP and Active Directory. The target deployment environment for the application tends to drive the choice of authentication mechanism: applications deployed in private or hybrid clouds tend to use enterprise-focused authentication mechanisms such as Microsoft Windows domain or directory-based authentication (i.e. LDAP and Active Directory); while applications deployed in public clouds tend to use consumer-focused authentication mechanisms such as OATH and OpenID.

Identity Management

Cloud application developers should also consider the type and location of the identity sources that their application and users will use with each of the authentication mechanisms that their application supports. For authentication to the PaaS provider, the PaaS provider will typically manage and host the identity source (i.e. directory or PKI-based identity). The identity sources used for authentication to the cloud application vary with the target deployment environment in the same way that the authentication mechanism itself does: applications deployed in private or hybrid clouds tend to use identity sources managed by and hosted within the enterprise (such as Windows domain servers); while applications deployed in public clouds tend to use identity sources managed and hosted by a cloud service provider.

Single Sign-On (SSO) and Identity Federation

There is great interest, particularly from enterprise users, for applications to provide seamless access for their users regardless of whether the application is deployed in a private, public or hybrid cloud environment. As an alternative to the authentication mechanisms we've discussed so far, in which users directly authenticate themselves to the cloud application, the application developer may choose to use single sign-on (SSO) and/or federated identity mechanisms instead. With these mechanisms, a user only needs to authenticate himself/herself once, within his or her own enterprise environment, and from then on can access the cloud application without having to re-enter his or her credentials. SSO and federated identity mechanisms work by establishing a "behind-the-scenes" trust relationship between the enterprise and the cloud service provider so that SSO tokens, such as Security Assertion Markup Language (SAML) tokens, can be passed to and validated by the cloud application.

Action Items

- Cloud application developers should implement the authentication methods and credentials required for accessing PaaS interfaces and services.
- Cloud application developers need to determine the type of cloud environment (private, hybrid or public) in which their application will run and ensure that they implement appropriate authentication methods for those environments.
- When developing cloud applications to be used by enterprise users, developers should consider supporting SSO solutions such as SAML.

CSA Reference

CSA Top Threat 3: Account or Service Traffic Hijacking

CSA Top Threat 4: Insecure Interfaces and APIs

CSA Top Threat 6: Malicious Insiders

References

- Initiative for Open AuTHentication (OATH), *OATH Reference Architecture, Release 2.0, 2007*. http://www.openauthentication.org/webfm_send/1
- OpenID Foundation, *What is OpenID?* http://openid.net/get-an-openid/what-is-openid/
- Organization for the Advancement of Structured Information Standards (OASIS), *Security Assertion Markup Language (SAML),* Version 2.0, March 2005, OASIS Standard http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip

IV. Implementation Issues

Shared-Domain Issues

Description

Several cloud providers offer domains that developers can use to store user content, or for staging and testing of their cloud applications. Some examples of such domains include:

- Windows Azure cloudapp.net and azurewebsites.net
- Amazon s3.amazonaws.com
- Google googleusercontent.com

While such domains are undoubtedly convenient, they can also present surprising security issues because of nuances of the same-origin policy. The same-origin policy is a mechanism built into browsers that prevents a page running client-side script (such as JavaScript) from reading data outside that page's same "origin". Browsers define two pages as belonging to the same origin when they share the same protocol (HTTP or HTTPS), port (normally port 80 for HTTP and 443 for HTTPS) and domain name (such as www.safecode.org). Furthermore, the domain names between the two pages must be exactly the same; "www.safecode.org" is considered to be a different origin from "safecode.org" or "public.safecode.org".

From a security point of view, the same-origin policy is a good thing: a cloud developer building an application on the domain jacksbuckets.cloudsite.cxx wouldn't necessarily want his competitor at jillspails.cloudsite.cxx to be able to write client-side script code that could read pages on the jacksbuckets site. Without the same-origin policy, pages on jillspails could open frames to jacksbuckets to see if the user had been making orders to both businesses (and maybe to offer the user a special discount to undercut Jack's prices if so).

However, suppose that Jack and Jill settle their differences and decide to enter into a business partnership together. Now they do want their sites' client-side script to be able to access each other's pages, but the same-origin policy prevents it. To bypass the restrictions of the same-origin policy, they both set their applications' JavaScript *document.domain* property to "cloudsite.cxx". Now client-side script on jacksbuckets.cloudsite.cxx can read pages on jillspails.cloudsite.cxx, and vice versa. However, now any other site hosted on cloudsite.cxx can also lower its *document.domain* property to just "cloudsite.cxx" and read both Jack's and Jill's customers' data.

Similar shared-domain access problems exist for other areas:

- HTTP cookies: Normally cookies are scoped to the fully qualified domain name of the page where it was created (i.e., "jacksbuckets.cloudsite.cxx"). However, developers can choose to widen the scope of the cookie domain (i.e., ".cloudsite.cxx"). Just as in the *document.domain* case, this lets applications share cookies between domains, but opens up the potential for malicious third parties to both intercept and tamper with cookie values.
- HTML5 local storage mechanisms: All browsers' implementations of the HTML5 localStorage object default to scoping access to the complete fully qualified domain. However, some browsers allow pages to widen the scope. Again, this lets applications share data, but creates potential security vulnerabilities that allow attackers to read and write that data.

Action Items

- Ensure that your cloud applications are using custom domains whenever the cloud provider's architecture allows you to do so. For example, Azure users should register custom domains for the production versions of their applications, and should only use cloudapp.net as a test or staging area.
- Review your source code for any references to shared domains.
- As a defense-in-depth measure, review your source code for any loosening of domain restrictions, such as the JavaScript document.domain property, the HTTP cookie domain property or HTML5 localStorage scope.

CSA Reference

CSA Top Threat 1: Data Breaches

CSA Top Threat 9: Shared Technology Vulnerabilities

References

Books, Articles and Reports:

- Web Application Security, A Beginner's Guide; Chapter 5, Browser Security Principles: The Same-Origin Policy; Sullivan, Liu; McGraw-Hill
- Security Best Practices for Developing Windows Azure Applications; Marshall, Howard, Bugher, Harden; http://www.microsoft.com/en-us/download/details.aspx?id=7253

Securing APIs

Description

APIs are the front door into any application and it is critical that they are properly secured. In many ways, API security for cloud applications is similar to API security for web applications hosted in data centers. Traditional application layer security risks, such as the OWASP Top 10[1], are still present when deploying your application to the cloud. This section will highlight a few minor differences from securing traditionally hosted applications and the risks that are common among most cloud applications.

As with any coding project, it is important to have a secure software development lifecycle to ensure that there are security controls for each phase of development. If you are following an Agile development methodology for building your cloud application, you may want to refer to the SAFECode publication "Practical Security Stories and Security Tasks for Agile Development Environments"[2] and/or to the Microsoft SDL for Agile[3] documentation.

The first opportunity for securing a web application API is the design phase of the software development process. Many web APIs are designed to only be used by other cloud services and they are not intended to be exposed to the general public. While a set of services may logically be deemed "back end" since they are not providing the end-user UI, often times these "back end" cloud servers are still exposing their APIs to the Internet in order to receive requests from the "front end" cloud services. In these instances, an organization should consider options for restricting access to the API such that only approved clients can reach it. One option is to leverage mutually authenticated SSL to ensure that only approved clients can interface with the API. Depending on your infrastructure, you may also be able to use firewall rules to limit access to the known client's IPs.

Organizations often look to single sign on (SSO) solutions for allowing the customer to use multiple individual cloud services. SSO solutions such as OAuth can be tricky to implement

correctly and it is critical that these methods are reviewed in the design phase to ensure that they are applied correctly. In addition, many authentication methods require the use of SSL. Review your cloud provider's SSL capabilities and ensure that your SSL solution can scale within your cloud deployment.

Lastly, cloud applications often have a multi-tiered architecture with front end web servers delegating tasks to workers deployed on a different network of machines. As a company grows, there may be multiple front end services passing data back to those workers. Organizations should review how data validation will be scaled in these situations. Front end services are typically responsible for authentication, authorization and a minimum of validation. As new services are created that pass data to the back end, it will be important to ensure that each service enforces those controls consistently. On the other hand, if the back end workers are intended to handle services such as file uploads, then it may make sense to have the workers perform certain types of validation such as anti-virus scanning. As a defense in depth measure, workers should perform their own validation if they are exposed to the Internet. Careful planning and threat modeling will help ensure that validation is handled at the correct point in the work flow and consistently applied as you grow your service.

For the testing phase, some organizations will want to deploy continuous scanning against their production environment to determine any real world risks as soon as possible. Be aware that there are risks associated with testing the production cloud environment. Organizations should review their hosting provider's policies to determine whether the hosting provider will need to be notified. Consider whether the testing will impact other customers in multi-tenant environments. Also, load balancing within the cloud may affect certain types of tests and they may need to be performed in a standalone environment. For instance, if the test attempts use a vulnerability to upload a malicious file, then the researcher may not be guaranteed to reach the same server to test for the existence of the file in the second step.

API tests should cover all of the traditional web application/service vulnerabilities such as those found within the OWASP Top 10. Organizations must ensure that their testing tools are appropriate for their environment. For instance, many cloud services are based on NoSQL databases and some testing tools do not yet test for NoSQL injection vulnerabilities. In addition, many testing tools now have the ability to be deployed in the cloud for large scale scanning and localized testing from the same network. Organizations should explore whether that is preferred for their testing needs.

The security testing phase is also an opportunity to validate the organization's centralized logging and monitoring capabilities. If an attack during the penetration test was successful, validate whether a log of the attack was captured in the centralized logging system. There are a multitude of approaches for detecting attacks in the cloud. Some organizations choose to implement web application firewalls (WAFs), such as mod_security. Other organizations have chosen to search for attacks based on the errors generated by their web server and databases[4]. If the services provide web pages, organizations can leverage content security policies (CSP) with reporting enabled. Host based intrusion detection can help identify web attacks that target the OS. Regardless of your monitoring implementation, your penetration test can be used to measure your breach detection capabilities. Once the log is captured, analyze whether it would be possible to structure an alert based on that log to detect future successful attacks.

Deploying cloud services focuses on the ability to scale. Therefore, the security development lifecycle for those services should also be able to handle scalable deployments. By carefully restricting access to APIs, an organization can reduce the threats to their data. Ensuring that regular testing is performed with the appropriate tools will ensure that the designs and code

meet the organization's security goals. Further, testing can help to validate the monitoring capabilities of the services.

Action Items

- Use your design phase to carefully plan how the components of your cloud service will interact. Determine if the APIs can be restricted so that only trusted hosts can call them. Ensure that inter-service communication is securely authenticated.
- Ensure that the tools that you are using are appropriate for your APIs and can target the deployed technologies.
- Use testing to validate your security monitoring and alerting capabilities. Ensure that any successfully exploited vulnerability was logged and appropriate alerting occurred.

CSA Reference

CSA Top Threat 1: Data Breaches

CSA Top Threat 9: Shared Technology Vulnerabilities

References

[1] OWASP Top Ten Project; https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] Practical Security Stories and Security Tasks for Agile Development Environments; http://www.safecode.org/publications/SAFECode_Agile_Dev_Security0712.pdf

[3] SDL for Agile; Microsoft; http://www.microsoft.com/security/sdl/discover/sdlagile.aspx

[4] Effective Approaches to Web Application Security; Zane Lackey; http://vimeo.com/54107692

V. Summary of Design and Implementation Action Items

Multitenancy

- For providers: Model all of your application's interfaces in threat models. Ensure that multitenancy threats such as information disclosure and privilege elevation are modeled for each of these interfaces, and ensure that the threats are mitigated in the application code and/or configuration settings.
- For providers: Use a "separate schema" database design when building multitenant applications instead of adding a "TenantID" column to each table.
- When developing applications that leverage a cloud service provider's (PaaS) services, be aware of and take advantage of how the interfaces into the PaaS stack provide tenant segregation.

Tokenization of Sensitive Data

- When designing a cloud application determine if the application needs to processing sensitive data and if so, identify any organizational, government, or industry regulations that pertain to that type of sensitive data and assess their impact on the application design.
- Whenever possible, organizations should minimize the applications and or systems that need to process and store sensitive data. Consider implementing tokenization to reduce or eliminate the amount of sensitive data that needs to be processed and or stored in cloud environments.
- Use data masking in development, continuing engineering and pre-production environment to exempt these environments from data protection requirements.

Trusted Compute Pools

- Ensure that your platform for developing cloud applications provides trust measurement capabilities and the APIs and Services necessary for your applications to both request and verify the measurements of the infrastructure they are running on.
- Verify the trust measurements as either part of the initialization of your application or as a separate function prior to launching the application.
- Audit the trust of the environments your applications run on using attestation services and/or capabilities such as the OpenAttestation (OAT) SDK, products that leverage OAT or native attestation features from your infrastructure provider.

Data Encryption and Key Management

- When developing an application for the cloud, determine if cryptographic and key management capabilities need to be directly implemented in the application or if the application can leverage cryptographic and key management capabilities provided by the PaaS environment.
- In order to protect and have access to encrypted data, make sure that appropriate key management capabilities are integrated into your application to ensure continued access to data encryption keys, particularly as data moves across cloud boundaries (enterprise to cloud, public to private cloud, etc.).
- Developers should be aware of and act upon the encryption and key management considerations outlined in this document.

Authentication and Identity Management

- Cloud application developers should implement the authentication methods and credentials required for accessing PaaS interfaces and services.
- Cloud application developers need to determine the type of cloud environment (private, hybrid or public) in which their application will run and ensure that they implement appropriate authentication methods for those environments.
- When developing cloud applications to be used by enterprise users, developers should consider supporting SSO solutions such as SAML.

Shared-Domain Issues

- Ensure that your cloud applications are using custom domains whenever the cloud provider's architecture allows you to do so. For example, Azure users should register custom domains for the production versions of their applications, and should only use cloudapp.net as a test or staging area.
- Review your source code for any references to shared domains.
- As a defense-in-depth measure, review your source code for any loosening of domain restrictions, such as the JavaScript *document.domain* property, the HTTP cookie domain property or HTML5 localStorage scope.

Securing APIs

- Use your design phase to carefully plan how the components of your cloud service will interact. Determine if the APIs can be restricted so that only trusted hosts can call them. Ensure that inter-service communication is securely authenticated.
- Ensure that the tools that you are using are appropriate for your APIs and can target the deployed technologies.
- Use testing to validate your security monitoring and alerting capabilities. Ensure that any successfully exploited vulnerability was logged and appropriate alerting occurred.

VI. Moving Industry Forward

The emergence and maturation of cloud computing has already provided significant advantages to technology users of all kinds, and many believe we have only just begun to explore the possibilities. But with new advancements often come new security challenges. The path to secure cloud computing cannot be viewed as a simple checklist of practices, but rather a process that must evolve alongside new use cases and advancements in our understanding of IT and software security.

It is our hope that our work as outlined this paper reflects that approach by bringing together the unique considerations of those developing the software that powers cloud computing with the latest guidance on secure development practices. While we aimed to provide actionable advice that can – and should – be tailored by individual enterprises to meet their needs, we believe that these practices are best viewed as part of a larger software security process that will continue to evolve alongside advancements in cloud computing.

To this end, we not only encourage broad industry adoption of these practices, but also continuing collaboration among professionals and experts across the industry to ensure that software security processes and practices continue to support the security needs of those building and implementing cloud computing solutions.

Supporting the Cloud Security Ecosystem

The Cloud Security Alliance has a number of working groups dealing with topics such as mobile, big data, privacy and security service level agreements. To support secure adoption of cloud computing, there are a number of areas we believe are important to consider. These include interoperability and portability, security innovation in the cloud, Privacy level agreements (PLA), Security Service Level Agreements (SecSLA), and the promotion of open standards. New guidance and best practices will emerge to complement the work done in this paper and hopefully enable and support cloud providers and especially cloud customers in aligning their internal policies and security architectures with agreed upon cloud security policies where applicable, and to help them evaluate the level of service offered by the cloud service providers.

Acknowledgements

Brad Arkin, Adobe Systems, Inc.

Eric Baize, EMC

David Doughty, Intel

Steven B, Lipner, Microsoft

Dr. Frances Paulisch, Siemens AG

Stacy Simpson, SAFECode

About SAFECode

The Software Assurance Forum for Excellence in Code (SAFECode) is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. SAFECode is a global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services. Its members include Adobe, CA Technologies, EMC Corporation, Intel Corporation, Microsoft Corp., SAP AG, Siemens AG, and Symantec Corp.

About the Cloud Security Alliance

The Cloud Security Alliance (CSA) is a not-for-profit organization with a mission to promote the use of best practices for providing security assurance within Cloud Computing, and to provide education on the uses of Cloud Computing to help secure all other forms of computing. The Cloud Security Alliance is led by a broad coalition of industry practitioners, corporations, associations and other key stakeholders.