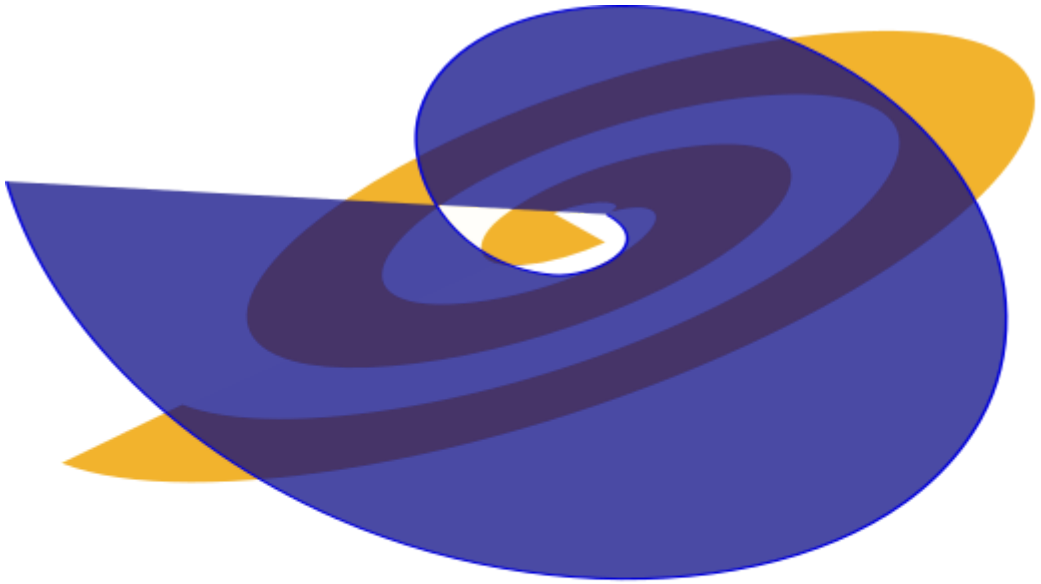
**

Cloud Transformation Security Reference Architecture



Verviam

Simple Secure Identity Management

Author: Nya Alison Murray

January 2018

Abstract

This paper is intended as a guide to current security practice for organizations moving part or all of their information technology to a hybrid cloud deployment, where virtual infrastructure is hosted in a combination of on-premises data centers and with public and private Cloud Service Providers.

A best practice security reference architecture contains specifications for the following key deliverables.

1. Identity and Access Management
   Manage identity and access for your cloud administrators, application developers and application users.
2. Infrastructure Security
   Handles network security, secure connectivity and secure compute infrastructure.
3. Application Security
   Address application vulnerabilities and the effectiveness of application security measures.
4. Data Security
   Discover, Categorize, Protect Data & Information Assets including protection of data at rest and in transit
5. Secure Dev-Ops
   Securely acquire, develop, deploy and maintain cloud services, applications and infrastructure
6. Security Monitoring and Intelligence
   Provide visibility into cloud infrastructure, data and applications in real time and manage security incidents
7. Security Governance, Risk and Compliance
   Maintain policy, audit and compliance measures, meeting corporate policies, solutionspecific regulations and governing laws

Key Compliance Requirements

It is essential to determine which regulations, standards and frameworks apply to a given jurisdiction, particularly for important directives such as PCI-DSS and the EU GDPR which have strong penalties for non-compliance.

Audit Trails

Generally, audit trails and logs of system activities are particularly useful to demonstrate compliance, as currently there are no foolproof methods for information security.

1. Application monitoring

   a. Application security events
   b. Payment transaction events
   c. Application logs

2. Activity auditing and compliance monitoring
   a. Audit logs
   b. Analytics and monitoring reports
3. System intrusion detection
   a. Network security events
   b. Infrastructure logs
   c. Operating System logs
4. 4. Forensics – proactive monitoring based on known threats and vulnerabilities in view of system design

PSD2

PSD2 provides rules for payment security and customer authentication, concentrating on protecting consumers when paying on the internet. For payment security, the directive states that "all payment service providers, including banks, payment institutions or third-party providers (TPPs), will need to prove that they have certain security measures in place ensuring safe and secure payments. The payment service provider will have to carry out an assessment of the operational and security risks at stake and the measures taken on a yearly basis." PSD2 also acknowledges the need for authentication mechanisms to match the context of the payment transaction. Specifically, Article 98.3 specifies exemptions for strong customer authentication in certain scenarios including: low value payments, outgoing payments to trusted beneficiaries, low-risk transactions based on a transaction risk analysis. Under PSD2, PSPs will be obliged to apply Strong Customer Authentication (SCA) when a payer initiates an electronic payment transaction. The EC defines SCA as a process that "validates the identity of the user of a payment service or of the payment transaction". SCA is based on the use of two or more elements:
1. Knowledge – something only the user knows, e.g. a password or a PIN
2. Possession - something only the user possesses, e.g. a card or an authentication code (One-Time-Password or OTP) generating device
3. Inherence – something the user is e.g. biometric authenticator such as fingerprint, voice or eye-print.

Key Aspects of Cloud Security

The following are the key aspects that should be included when designing a secure cloud solution:

1. **Manage identity and access**: Consistent way to manage identities and access for platform and application users.
2. **Protect infrastructure, data, and application**: Ensure tenant isolation and protection for compute, data and networking. Safeguard against application and network threats, exploits, and vulnerabilities. Provide secure connectivity to data at the enterprise and protect sensitive data both in transit and at rest on the cloud.
3. **Security monitoring and intelligence**: Gain visibility into virtual infrastructures by collecting and analyzing logs in real-time across the various cloud components and cloud services. 4.

Optimize cloud security operations: Optimizing the processes, methods, and tools for running your security operations is key to keeping the overall cost low. Consistently assess security practices, plans, and designs and evolve them in a timely manner to stay ahead of threats.

Table 1: Core Application Security Considerations

Areas	Description	Application Security Considerations
Governance	Provide the right guidance and oversight to ensure the right controls are in place to protect application services	Policies Principles
Protection	Implement processes and technologies to maintain the confidentiality, integrity, and availability of application services	Identity and Access Management Cryptography Web Application Firewall
Monitoring	Monitor application services to detect unauthorized activity or breach	Traceability Web Application Scanning
Response	**Provide effective and efficient response Security Incident to security incidents and mitigate risk management**	Security Incident Response

Governance

Well-defined policies and principles are required to drive effective application and software security. As referenced in the OMG publication "Security in Cloud Computing: 10 Steps to Ensure Success Practical Guide", clearly defined security policies are essential to ensure applications enable business rather than introduce additional risk. The ISO27002 code of practice for information security provides a comprehensive framework for security policy development across all technology areas.

The following guidelines are provided by cloud reference architecture from publications from the Object Management Group Cloud Resources

1. Ensure effective governance, risk and compliance processes exist
2. Audit operational and business processes
3. Manage people, roles and identities
4. Ensure proper protection of data and information

5. Enforce privacy policies
6. Assess the security provisions for cloud applications
7. Ensure cloud networks and connections are secure
8. Evaluate security controls on physical infrastructure and facilities
9. Manage security terms in the cloud service agreement
10. Understand the security requirements of the exit process

Evaluation of Infrastructure Services

The cloud service provider should be able to give assurances of sufficient physical security, such as by an independent SOC 2 report, ISO 27001 certification, or similar means.

In cases where there is on-premises infrastructure, it is up to the implementer to have good physical security practices. For example, doors with anti-tailgating measures, video surveillance, slab-to-slab barriers, and password protected consoles are common controls. Detailed implementation guidance may be found in ISO 27002 section 11, PCI DSS 3.2 requirement 9, and other standards.

Network and Infrastructure Security

Network Security Groups and ACL Firewalls

Network Security Groups and Access Control Lists act as simple firewalls, controlling access to servers based on source IP addresses. A fundamental challenge with securing access is the requirement to provide secure user access with changing IP addresses, depending on location.
All server instances that are assigned to a cloud firewall group will inherit this set of rules, which allows network access to specific ports, and coarse-grained access to VPCs, VNETs and subnets

One of the major problems is complexity in terms of user access policies. Changes to policies can ripple through with unintended access changes. While there are tools to scan for the effects of policy changes, this is an inefficient approach.

Both role based and attribute based access control have benefits, and a hybrid approach is required to cover secure remote access by devices, users and applications.

The real challenge is that as deployments become more complex, and BYOD is now standard practice, while telematics from a wide variety of IOT devices are becoming ubiquitous, a true Zero Trust approach is required.

While this term has been widely used, the original context was to ensure isolation at the network connectivity level. A true zero trust deployment requires an approach such as the Cloud Security Alliance Software Defined Perimeter where an identity attribute check at the first packet of a network connection is the only really effective allow/deny checkpoint for the current multi-cloud hybrid environment.

Perimeter Security for Virtual Machines

In a cloud virtual network, setting up the perimeter security for the configuration is a core prerequisite to a secure cloud.

The following steps are essential determinations for optimizing cloud security

1. Security groups/ACLs, virtual appliances/gateways at the perimeter
2. Outbound traffic control/monitoring rules
3. Inbound traffic control/monitoring rules
4. Network connectivity with private, cloud site-to-site connections, VPNs and internet gateways
5. Hybrid set up, gateways and connections
6. Design subnetwork configuration, internal and external connections
7. Design subnets for API services and gateways including TLS termination and connections
8. Design web, application server and PaaS and SaaS subnet configuration and connections

Containers

Containers have gained popularity along with the deployment of microservice architectures. Applications isolated using containers share access to the operating system kernel. An understanding of container threats and available security controls is needed to manage risk associated with container deployments. IT organizations must begin with understanding the basics of container isolation: namespaces, control groups (cgroups), and network configuration. Namespaces define the virtual boundary for processes executed within containers while cgroups define the resources that can be consumed by the container (prevent resource starvation). Network bridging within container environments increase risk; proper configuration is needed to reduce exposure of container network traffic. After attending to the basics of secure container configuration, the administrator must focus on addressing specific threats, such as kernel exploits, container escapes, and cross container attacks. Identifying the right Mandatory Access Control (MAC) tool to apply security policies to containers (e.g., SMACK), performing kernel hardening, and limiting application calls to the kernel (Seccomp, “SECure COMPuting”) is essential to addressing specific threats. The NCC Group white paper Understanding and Hardening Linux Containers provides insight into container security.

Edge of Network Security

Edge-of-network content delivery services provide security services for protection from attacks including DDoS, SQL injection, cross-site scripting and other common threats that can bring applications down or compromise sensitive data. These platform delivers “always-on” protection authenticating valid traffic at the network edge. This capability can absorb even the most intensive attack in the hundreds of Gbps with relative ease.

Bastion Hosts and Jump Boxes

Bastion Hosts and Jump Boxes are servers that are designed to give users in a less-secure zone access to servers or services running in a more secure zone, either from within a software defined network, for from an external connection. The connectivity may be secured by a private connection or a VPN. Access is controlled by user authentication and can certainly help control access to cloud resources. They are designed for occasional access, such as by system admins, and provide all-or-nothing network access to all servers on the network, and may be a single point of failure if the authentication is breached.

TLS and Perfect Forward Secrecy

While TLS 1.3 provides advances in performance, some encryption improvements, such as Perfect Forward Secrecy (ephemeral shared secrets) full handshake signature, downgrade protection, and improved Diffie Hellman Elliptical Curve implementation, there are vulnerabilities. If an agent is deployed at TLS termination, it can decrypt packets based on shared secret in real time.

Certificate Pinning

Pinning is the process of associating a host with their *expected* X509 certificate or public key. Once a certificate or public key is known or seen for a host, the certificate or public key is associated or 'pinned' to the host. If more than one certificate or public key is acceptable, identity must match one of the elements in the pinset.

A host or service's certificate or public key can be added to an application at development time, or it can be added upon first encountering the certificate or public key. The former - adding at development time - is preferred since *preloading* the certificate or public key *out of band* usually means the attacker cannot taint the pin. If the certificate or public key is added upon first encounter key continuity can fail if the attacker has a privileged position during the first encounter. There are known vulnerabilities in some code libraries.

Data Security

Data Classification Considerations

Data exists in one of three basic states, Data at Rest, Data in Process, Data in Transit. All three states require unique technical solutions for data classification, but the applied principles of data classification should be the same for each. Data that is classified as confidential needs to stay confidential when at rest, in process, and in transit.

Data can also be either structured or unstructured. Typical classification processes for the structured

data found in databases and spreadsheets are less complex and time-consuming to manage than those for unstructured data such as documents, source code, and email.

For data that is managed in public clouds, the following table is of responsibility is a guide to the level of complexity in managing data custodianship.

Data Privacy

Some example categories of private data include:

1. Personally Identifiable Information (PII), such as name, address, phone number, email, etc.
2. Technically Identifiable Personal Information, such as geolocation data, device IDs, usage based identifiers, static IP address, etc. ...when linked to an individual
3. Employment Related Identifiable Information, such as job history and performance review information
4. Personality Related Identifiable Information, such as personality insights or sentiment analysis
5. Sensitive Personally Identifiable Information (SPI), such as government ID, racial/ethnic origins, marital status, sexual orientation, trade union memberships, political views, etc. Financial Information (PCI DSS, FFIEC, etc.), such as credit card, bank account, financial holdings, salary information, etc.
6. Healthcare Information (PHI, HIPAA, etc.) such as patient record, health insurance details, diagnostic or treatment information, genetic information, etc.
7. Law Enforcement Information, such as Security clearances, criminal history, background check information, etc.

Data Activity Monitoring

Data Protection can be accomplished by all or some of the following activities: Privacy is becoming a greater factor, especially with EU GDPR rules i.e. IP addresses regarded as private items - should privacy be incorporated into every application and related service? Some would suggest that given the open scrutiny of encryption algorithms, privacy should have the same treatment

1. At rest encryption, in process and in transit
2. Hardware storage management with varying storage types files/objects/databases/block
3. Key management
4. Data services (such as an SQL DB as a service) with encryption
5. Data Integrity (e.g. Hash values) - certificate management
6. Data classification
7. Data activity monitoring
8. PII handling

Identity, Authorization and Authentication

For web application access control, the OpenID protocol provides for authentication of the identity of the user requesting resources. OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the EndUser in an interoperable and REST-like manner.

The OAuth protocol provides for authorization of the user's access to protected resources. However, it was developed to provide cross-domain access control, and the elaborate handshakes and code exchanges can be intercepted readily.

OAuth 2.0 Authorization Grant for Native Mobile Apps

Historically, one of the weaknesses of OAuth was a poor end-user experience on mobile devices. To help smooth the user experience, it was common for native OAuth clients to leverage a "web-view" component when sending the user to the authorization server's authorization endpoint (interacting with the front channel).

A web-view is a system component that allows applications to display web content within the UI of an application. The web-view acts as an embedded user-agent, separate from the system browser. Unfortunately, the web-view has a long history of security vulnerabilities and concerns that come with it. Most notably, the client applications can inspect the contents of the web-view component, and would therefore be able to eavesdrop on the end-user credentials when they authenticated to the authorization server.

Since a major focus of OAuth is keeping the user's credentials out of the hands of the client applications entirely, this is counterproductive. The usability of the web-view component is far from ideal. Since it's embedded inside the application itself, the web-view doesn't have access to the system browser's cookies, memory, or session information. Accordingly, the web-view doesn't have access to any existing authentication sessions, forcing users to sign in multiple times.

One thing that native OAuth clients can do is to make HTTP requests exclusively through external user-agents. A great advantage of using a system browser is that it lets the resource owner see the URI address bar, which acts as a great anti-phishing defense. It also helps train users to put their credentials only into trusted websites and not into any application that asks for them.

In recent mobile operating systems, a third option has been added that combines the best of both approaches. In this mode, a special web-view style component is made available to the application developer. This component can be embedded within the application just like a traditional web-view. However, this new component shares the same security model as the system browser itself, allowing single sign-on style user experiences. Furthermore, it is not inspectable by the host application, leading to greater security separation on par with using an external system browser.

The OAuth 2.0 grant that mobile apps utilize in order to access an API, is the Authorization Code Grant using Proof Key for Code Exchange (PKCE). The best current practice for authorizing users in native apps is to perform the OAuth authorization request in an external user-agent (typically the browser),

rather than an embedded user-agent (such as one implemented with web-views). This is a view of native app authorization from the IETF.

In typical web-view based implementations of embedded user-agents, the host application can: log every keystroke entered in the form to capture usernames and passwords; automatically submit forms and bypass user-consent; copy session cookies and use them to perform authenticated actions as the user. Even with system browsers there are mobile device vulnerabilities.

JSON Web Tokens (JWTs)

JSON Web Token is an Internet standard for creating JSON-based access tokens that assert authorization claims as required by business processes.

Token information is Base64 encoded, and can be signed with using an agreed algorithm to one-way hash the payload. This allows for validation of the token contents. There are vulnerabilities, and token security is vastly improved by using a public/private key pair signature and/or encryption.

Internet Connections

Use of TLS (https), OAuth/OpenID and JWTs is no guarantee of security across the public internet, with the current level of sophisticated hacking.

The best form of internet security is to encrypt data on the device, send it encrypted over TCP and TLS and store it in encrypted form, securing symmetric and asymmetric public/private keys in secure purpose designed data and key stores.

Mobile devices are particularly insecure at the OS level, and the following practices are important:

1. Use specific content types.
2. Validate input
3. Don’t allow caching on devices
4. Authenticate all API requests
5. Ensure parsers are up to date

For data sent over any network, particularly the public internet, the best form of data privacy and data protection is to use device security, and data encrypted in the browser, sent encrypted, stored encrypted, returned encrypted, and decrypted in the browser only by the data owner on provision of multi-factor authentication. And even this level of security is vulnerable with weak authentication, such as passwords and biometrics stored on the device, even in data security stores, because of cached data and browser leakage.

API Management Security

It is important to determine APIM security policy and security integration, including user management, rate limiting, and attack protection.

Enforcing agreements on API use and security

An API key is often only the first element of an APIM provider's tracking of API use. API management products enforce the usage parameters that API providers and API users agree upon in a variety of ways, including the use of TLS, secure tokens, and digital signatures for added security; the use of authentication and API authorization for service and data access, and quotas and rate limits on how many API calls an API user can make. API management solutions use an API gateway - in most cases, one embedded in the solution - to enforce some level of security and access control.

For API management, B2B and P2P scenarios increase the need to manage partner organizations, which may have many individual developers. Security federation for API requests, from end users, devices and applications, and more complex services with higher needs for security and integrity, with a broader range of messaging styles, means that many layers of security is more effective.

Integration with business partners may require some custom solutioning to ensure third party data protection.

API Management capabilities can be augmented with

1. Rich authentication and authorization capabilities including support for encrypted tokens as well as OAuth/ OpenID, SAML 2.0, and JWT support
2. Integration with leading identity and access management providers
3. PKI and key distribution capabilities including comprehensive HSM integration 4. Use of existing enterprise security systems so that users can manage access rights for their own data
5. Message security with sophisticated encryption and signature capabilities
6. DDoS prevention and detection measures.

Secure DevOps

DevOps extends the application software development lifecycle (SDLC) by taking the approach that increasing automation of promoting applications to production can mitigate risks due to disconnects between developers, quality assurance (QA), testers, and operations. Native cloud deployments and DevOps together provide a rapid response mechanism for application changes.

Embedding security in a DevOps operational framework takes advantage of the increased agility and standardization of application deployment. Secure DevOps can be viewed as an extension of application security that is designed to utilize cloud infrastructure, platform virtualization and automation. While scripted deployment of code, configuration and toolsets provides faster and more consistent software lifecycles, all the principles of application security are even more relevant, particularly determination of what must be manually tested to avoid unintentional results from increasing automation.

From an automation point of view, security tests can be categorized as follows:

1. **Functional Security Tests.**
   These are essentially the same as automated acceptance tests, but targeted at verifying that security features such as authentication and logout, work as expected. They can mostly be automated using existing acceptance testing browser automation tools

2. **Specific non-functional tests against known weaknesses.**
   Includes testing known weaknesses and misconfigurations such as lack of the HttpOnly flag on session cookies, or use of known weak SSL suites and ciphers. These are particularly well suited for automation because the weaknesses are known up front. This approach can serve as the security specification before building the application and environment. Some work has already been done in extracting these types of tests into security test automation frameworks. Because these test non-functional aspects of the application, they need access to the HTTP layer which browser automation tools do not provide. So testing these requires a hybrid approach: Browser automation together with a proxy server to inspect and inject requests.

3. **Security scanning of the application and infrastructure.**
   Even manually driven penetration tests usually kick off with an automated scan using vulnerability scanning tools such as an infrastructure scanner to test an IP address and all exposed ports for known weaknesses, web scanning components that will test HTTP services OWASP ZAP Open Source is focused on the web tier as application scanners in that they inspect and test at the HTTP layer by injecting attack data into parameters and evaluating the application's response. They can provide in-depth security scanning if they're used correctly. But if they're simply used to spider the application and run an automated test then there's a good chance that they won't find or test all the available content. To successfully automate application scanning, one should ensure that all the content to be scanned is navigated and populated in the scanning tool, before starting to spider and scan the application.

4. **Security testing application logic**
   Automated tools can only go so far in detecting security flaws. To identify flaws in the logic of the application requires a human brain, to understand the context, and apply human logic such as an attacker may use. Once tests are defined, they can be recorded as automated tests and become a part of the security regression tests.

Development Practices for Threat Prevention

SQL Injection Prevention

1. Use of Prepared Statements (Parameterized Queries)
2. Use of Stored Procedures
3. Escaping all User Supplied Input
4. Principle of Least Privilege
5. White List Input Validation

Defensive Coding Practices for Web Applications

The OWASP Top 9 coding security flaws are listed in this section the most critical security flaws to find during a code review. These coding practices may require extension to address container micro

services security.

1. Input Validation
2. Source Code Design
3. Information leakage and improper error handling
4. Direct object reference
5. Resource usage
6. API usage
7. Best practices violation
8. Weak Session Management
9. Using HTTP GET query strings

Application Security Testing

OWASP provides a good starting point for web application penetration testing. There are many open source penetration testing frameworks.

Application analysis by developer teams covering logic, data flow and platform is very useful.

Peer review of application code installation, configuration and deployment is required prior to code deployment automation.

Automated Security Scanners for DevOps pipelines are very useful, as is the integration of testing frameworks.

Cultural Change

Automated and manual scans are really useful, however the main defense has to be a culture of proactive security awareness by developers, both in reviewing code, and in sharing information about best practice. This is the most effective method of prevention of vulnerabilities introduced during development and deployment, particularly in view of the short time cycles associated with CI/CD.

Security Monitoring, Logging, SIEM, Endpoint Prevention and Detection

These technologies are proliferating, and most are developing machine learning and AI capabilities from collected data to identify potential and actual attacks. AI models use multiple linear regression analysis of past events as training for detection and prevention models. As collected data accelerates, major refinements will become available over the coming decade. Operational logs have to be carefully analyzed for insight capabilities for both streaming and time series data.

Useful References

PCI DSS

Organization payment standards compliance quick reference guide

https://www.pcisecuritystandards.org/documents/PCI%20SSC%20Quick%20Reference%20Guide.pdf

GDPR

https://www.microsoft.com/en-us/trustcenter/Privacy/GDPR

https://gdpr-info.eu/

Cryptography

Overview of Cryptographic methods as at 26.02.2017

http://www.garykessler.net/library/crypto.html

JWT tokens vulnerability https://auth0.com/blog/critical-vulnerabilities-in-jsonweb-token-libraries/

JWT tokens with RSA https://connect2id.com/products/nimbus-josejwt/examples/jwt-with-rsa signature

Crypto libraries for various purposes Microsoft kernel validated

http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401val2017.htm

TLS/SSL

TLS 1.3 https://ietf.org/blog/tls13/

OWASP TLS overview https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

Application Security

OWASP ASVS

https://www.owasp.org/

Data De-identification (Masking, Pseudonymisation, Tokenization)

http://csrc.nist.gov/publications/drafts/800-188/sp800_188_draft2.pdf