Application Security Test Strategy and Plan

Verviam Security Program

Table of Contents

INTRODUCTION

As cloud computing becomes pervasive, and organizations are increasingly turning to hybrid cloud services from multiple clouds, there is a growing adoption of containers as the basis for microservices to connect capabilities across the business, apply logic to API services, and provide additional functionality to existing commercial systems. Verviam has adopted the following approach to security.

1. Security encompasses the end-to-end deployment environment, covering
    a. network connectivity and interconnectivity including public internet
    b. infrastructure perimeter includes edge-of-network services best practice as provided by Cloud Service Providers (CSPs)
    c. data security including encrypting PII data, security data in process, in transit and at rest
    d. API management security best practice as per CSPs
    e. Application workload security e.g. VMs and containers
2. Security starts with the Zero Trust network paradigm, as per the Cloud Security Alliance Software Defined Perimeter Architecture Guide.
3. A secure approach includes agile testing, development, network and operations statistics mapped to project dashboards, such as Jira, is applied to application development and operations monitoring.
4. While automated application layer scanning is taken very seriously, particularly OWASP guidelines, Verviam examines scan results in the context of the entire deployment, as security vulnerabilities are now to be found in every scripting and code library current versions, as well as past deprecated versions, so that defence in depth is the first consideration of security posture. Most application layer vulnerabilities disappear with a Zero Trust network deployment.

Verviam Identity and Network Services has been security tested for the following capabilities

1. Web browser application interfaces
2. Internet connectivity to API gateway services
3. API management services
4. Container platform services
5. Application microservices
6. Identity Management, Authentication and Authorization
7. Network connectivity
8. Hosting infrastructure perimeter security
9. Edge-of-network security

Application testing is particularly important and provides the last chance to check security compliance before production, however it is just one small part that must be taken in context with the industry standard Defence in Depth approach. The following diagram provides the current security context.

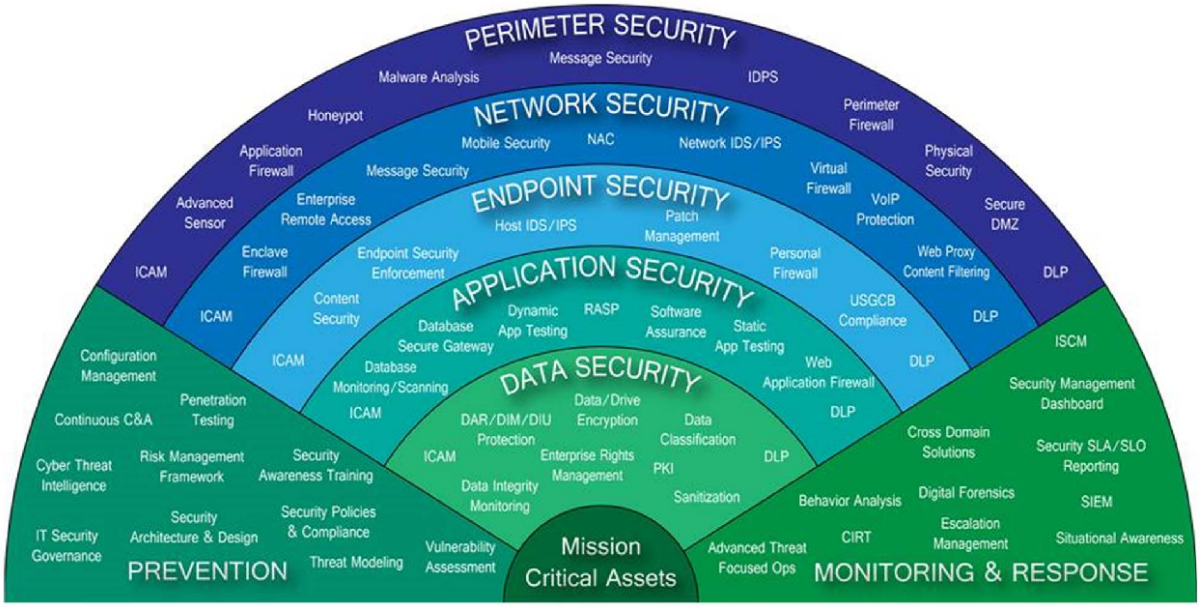

Figure 1: Illustration of Defence in Depth Security - source unknown

Objectives

The major objective for this Security Test Plan is to provide a baseline for testing Identity and Network Services Application Platform deployment to enterprise on premises and cloud hosting environments.

1. Ensure that security is correctly applied in the platform and infrastructure hosting environment
2. Ensure compliance with enterprise security regulatory regimens.
3. Provide high performance low latency application platform services in the platform and infrastructure deployment services while maintaining the requisite security posture.

APPLICATION SECURITY

Secure DevOps and Containers

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. **End User 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 like Selenium/WebDriver.

2. **Specific non-functional tests against known weaknesses.** Vulnerability scanners includes testing for known weaknesses and misconfigurations such as lack of the web Http Only 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 (if not by the development team, then by the security team). Because they 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: Vulnerability scanners now use both browser automation together with a proxy server to inspect and inject requests.

3. **Security scanning of web application and infrastructure.** Even manually driven penetration tests usually kick off with an automated scan using vulnerability scanning tools encompassing OWASP ZAP and other advisory guidelines. Coverage is for web and application tier scanners in that inspect and test at the HTTP layer by injecting attack data into parameters and evaluating the application's response.

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 thought architecture and design analysis and developer peer review.

Build

1. Code Analysis. Analyze code for application specific vulnerabilities.
2. Container Hardening. Remove unneeded libraries and packages; restrict functions.
3. Image Scanning. Scan images for vulnerabilities at build; regularly in registries.

Ship

1. Image Signing, e.g. Content Trust. Ensure trust with signing and author / publisher verification.
2. User Access Controls, e.g. Registries. Restrict and monitor access to trusted registries and deployment tools.

Run - Preparation

1. Host and Kernel Security. Use SECCOMP, AppArmor, or SELinux or equivalent host security settings.
2. Access Controls. Enable restricted access to system and Docker daemon.
3. Auditing, e.g. Docker Bench. Perform security audit using Docker CIS benchmark.

Run - Production

1. Network Inspection & Visualization. Inspect all container to container connections and build visualization for application stack behaviour.
2. Threat Detection. Monitor applications for DDoS, DNS attacks and other network-based application attacks.
3. Host & Container Privilege Escalation Detection. Detect privilege escalations on hosts and containers to predict breakouts and attacks.
4. Packet Capture & Event Logging. Capture packets and event logs to enable forensics.

Automating Security

Automation should be applied whereever possible to ensure security tests are performed. Security configurations can be automatically created and updated even as applications and deployment policies are constantly changed. Security automation examples for containers include:

1. Code analysis for application security flaws
2. Image scanning for known vulnerabilities (e.g. CVEs) in applications and libraries
3. Image signing, tagging, and access controls
4. Container security testing for images, daemon, hosts, and containers
5. Orchestration policies requiring deployment of monitoring and security containers
6. Host and kernel security settings when deploying new nodes
7. Security policy enforcement adapting as containers scale up, down and across
8. Logging, packet capturing, and integration with SIEM systems

Test Case Overview

The strategic direction is to provide detailed test cases for the following Security Testing topics. The test cases categorization provides a useful summary of application security for container, microservices and VM applications in the context of a secure network.

Topic	Description
Data Validation	Reducing the range of accepted input to disable script injection, SQL injection, and other kinds of intercepts
Authentication	Verifying and validation the identity of the client/server parties in network connections
Session management	Ensuring that short-term tokens are correctly used and expired
Authorization	Ensuring that resource policy and the granting of resource requests are only made when the requestor has the appropriate access rights
Cryptography	Ensuring that the strength and application of encryption techniques is current, not deprecated and of the appropriate strength for the classification of the information
Error handling	Ensuring that debugging level error handling is disabled, and that any unauthorized information is never exposed inadvertently
Logging	Ensuring that logging is tokenized, anonymized and pseudonymized so that no identifying information is every recorded
Security Configuration	Making best use of application platform services configuration to minimize inadvertent vulnerabilities and security loopholes.
Network Architecture	Ensuring best practice network connectivity, protocols, encryption at runtime is supported by best practice virtual network design including network level authentication, network segmentation and other software defined network capabilities.
Infrastructure Architecture	Ensuring that best practice security services are applied to hosting virtual infrastructure, content delivery and performance configuration using identity policies

SCOPE

Overview

What is being tested is that application, application platform, hosting infrastructure and network implementation and deployment comply with what is currently understood to be security best practice in view of the current regulatory, legislative and enterprise standards for a positive security capability that takes into account the advances in Zero Trust network connectivity for enterprise, public and private cloud connectivity.

Security Testing Summary

For the detailed approach see the Security Reference Architecture, Infrastructure Architecture and practical guidelines.

Testing	Verviam			Customer		
Description	Technology Services	Metrics	Expected output	Technology Services	Client metrics	Expected output
Coding and Infrastructure design guidelines	See services design	Review	Approval	Security Architecture, Design and Coding Reviews	Assurance	Assurance
SAST, SCA, IAST	See services design	Scan Results	Remediation of defects	Scanning technology	Scan Results	Remediation of defects
DAST	See services design	Scan Results	Remediation of defects	Scanning technology	Scan Results	Remediation of defects
Operations & Container Scanning	See services design	Scan Results	Remediation of defects	Scanning technology	Scan Results	Remediation of defects
Infrastructure Security	See services design	Infrastructure Security Architecture	Assurance	Infrastructure Security Architecture Monitoring	Assurance	Assurance
Network Security	See services design	Network Security Architecture	Assurance	Network Security Architecture Architecture	Assurance	Assurance

TACTICAL APPROACH

In addition to the usual functional and non-functional testing of the Identity and Network Services testing regimen, the use of static, dynamic and containerized scanners and test frameworks is to

expedite the identification of build and runtime issues for a particular CSP and enterprise hosting environment. This is to expedite the identification and remediation of both configuration and structural issues and defects, as well as to highlight any defects in coding and connectivity approaches and practices.

STRATEGIC APPROACH

The strategic approach to security is based on Zero Trust networking and the least privilege principle. A zero-trust security model is a philosophy for designing network security architecture in a way that withholds access until a user, device or even an individual packet has been thoroughly inspected and authenticated. Even then, only the least amount of necessary access is granted. According to Forrester, there are three main concepts of Zero Trust:

1. Introducing the concept of trust to the network, so that it becomes natural to ensure that all resources are securely accessed no matter who creates the traffic or from where it originates, regardless of location or hosting model, cloud, on-premises or collocated resources.
2. Adopting a least privilege strategy (LPS) that enforces access control to eliminate the human temptation to access restricted resources.
3. Continuously logging and analyzing user traffic inspection for signs of suspicious activity.

The reason for this approach is that malicious actors are continuously developing new variants of application security threats while simultaneously discovering new vulnerabilities in all scripting and code libraries, networking, identity and application protocols, including TCP/IP and TLS 1.3, IPSec, SSL VPN.

Security testing depends primarily on network and internetwork connectivity security. In addition, security depends on a combination of infrastructure perimeter, identity management, server application, data and database security in the context of data classification security requirements. For example, http and https connections may pose no security risk for publicly accessible data, whereas field level encryption may be required for PII data in process, in transit and at rest.

The Verviam Security Test strategy is prioritised as follows:

1. Defence in depth security is applied across a layered model such as the OSI model or the updated model using TCP/IP. As there are currently many vulnerabilities in the best security posture, no one measure can be considered isolated from the operating environment.
2. Apply strong security measures to network connectivity and interconnectivity using best practice Zero Trust measures such as Control Plane/Data Plane separation, and Single Packet Authentication containing identity information such as a token or identifier attribute.
3. Use Cloud Service Provider (CSP) infrastructure perimeter protection for VPCs/VNETs, subnet connectivity, VPC peering, network and internet gateways, data store security, key management services, identity policy based short term token authentication. This is because CSPs have collectively addressed security with the immense human, financial and

infrastructure resources at their disposal, providing a security posture that cannot be provided by a single enterprise. However there are relatively invisible internal and external vulnerabilities in Cloud Service Provider infrastructure, so it is important to apply a security reference architecture around scanning and testing for vulnerabilities.

4. Use dynamic runtime event operational intelligence and network packet scanning at the edge-of-network to prevent, detect and identify automated and well-orchestrated attacks.
5. Use secure development and network operations with an organization specific Target Operating Model to ensure best practice development, operations and networking. Support coding practice with manual and automated application scanning for the OWASP top 10 vulnerabilities in view of, and taking into account Zero Trust Software Defined Networking perimeter controls and Cloud Service Provider CSP role and attribute based identity policies, secure API gateways and operational port scanners.
6. Use automated dynamic scanning on VM and container workloads in a pre-production environment (a replica of production except for the use of test data or real data that is appropriately anonymized/pseudonymized/tokenized)

While automated and manual security scanning is mandatory, scan results must be read in the context of an application specific threat model, otherwise it may be misleading. For example, verbose logging of non-critical information may in fact be very useful for AI Security Orchestration Automation Response (SOAR) security analytic intelligence. It is also noted that automated scanning results must be read as an aid to identifying security risk, not as security advice as they must be evaluated in the context of other factors such as network and infrastructure perimeter security. For example, Cloud Service Providers (CSPs) provide comprehensive anti-DDoS measures. These attacks have changed their form recently and are evolving rapidly and are best addressed by using CSP anti-DDoS perimeter measures, rather than at the application level, where the porous nature of code means that this is too resource intensive to be practicable.

The testing strategy is to use industry recognized static and dynamic scanners applied to code, scripting, deployment images and runtime environments.

This provides for an evolution of security detection and prevention methods based on the emergence of Artificial Intelligence applied to aggregate analysis of logs as well as behavioral models detecting probable ingress.

This approach is necessarily proactive as advances in security technology are currently extremely rapid, and therefore may require some structural changes to be retrofitted as new intelligence emerges from global collective research into development. This is because there are many not-for-profit organizations as well as major CSPs and industry consultancies engaging in security research, and freely providing the results to the security community in view of the current heightened level of threats and vulnerabilities from well-organized actors engaged in criminal activities such as fraud and identity theft.

As identity management is one of the key areas of weakness in the current generation of information technology applications, attention is paid to authentication and authorization of end

users, devices and applications using Role Based Access Controls and Attribute Based Access Controls as well as identity directories and short-lived token validation.

TEST CASES BY CATEGORY

Manual Application Security Testing

Definition: Developer Peer Review to ensure the security of code

Participants: Developers, Operations, Network and Security Architects

Methodology:

API Security

APIs are pervasive in cloud environments. It is imperative that secure development methods are adopted, and validation of API security is performed during the testing process. At a minimum, the following approach is recommended:

1. Understand how the API acquires information – document functionality
2. Map the API flows. Understand all methods and functionality at the start of assessment. – Sequence and Class diagrams of flows
3. Capture the runtime traffic with logging and monitoring using operational intelligence feeding back into developer peer review
4. The common API vulnerabilities developers must be aware of include: a. No or weak authentication – Multi Factor Authentication where possible
   a) Weak password complexity – Password Rules
   b) Session tokens with no expiration
   c) Lack of logout/session expiration

Container Security

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 (Secom,

“SECure COMPuting”) is essential to addressing specific threats. Ensure feedback from container automated scanning is communicated into developer peer review

Operations Monitoring Test Cases

1. Ensure servers not unintentionally exposed publicly to the internet
2. Ensure no unencrypted data storage
3. Ensure least-privilege policies
4. Test for poor password policies or missing MFA
5. Test for misconfigured backup and restore settings
6. Test for data exposure and privilege escalation

Data in Process, in Transit and at Rest Test Cases

Ensure that appropriate level of encryption, secrets management and protection is provided for all levels of data classification across all application workflows

Data Validation Test Cases

Data validation is ubiquitous. Regex expressions as well as range values can be specified. Common validations can be built and factored out into utility classes as objects. Where this is not possible, coded validations can be used.

1. Test for data validation of all input parameters in all public interfaces
2. Test for data validation of all input parameters in all application services

Authentication Test Cases

1. Ensure token based authentication uses appropriate signing, encryption, and expiry aligned with data classification policies
2. Ensure appropriate use of IAM roles for users, groups and services, and that appropriate policies are enforced for role based access controls (RBAC)
3. Do not use predictable patterns for role mapping
4. Do not provide authentication failed due to an incorrect account identifier or an incorrect password messages when a brute force attack may be employed
5. Restrict access permissions to the minimum set required for the role

Session Management Test Cases

1. Ensure session tokens including authentication tokens have an appropriate expiry time and that they are expired on time.
2. Ensure browser session tokens do not leak information

Authorization Test Cases

1. Ensure scopes are appropriate for identity roles and groups
2. Ensure fine grained access control is correctly implemented
3. Test RBAC frameworks for correct implementation
4. Test ABAC and tagged values work as intended

Cryptography Test Cases

1. Ensure the use of TLS where appropriate, that the certificates are up to date, and the latest version of TLS is enabled.
2. Ensure PKI asymmetric keys (e.g. RSA public/private keypairs) have an appropriate key size. For medium term keys, 2048 bits is recommended. For long term keys, 4096 is recommended although there may be performance implications. For short lived ephemeral key pairs, the key size can be less.
3. Ensure AES symmetric keys follow current standard practice (e.g. key size of 128, 192, or 256 bits, block size 128 bits, and rounds 10,12 or 14 depending on key size)
4. Ensure PII data is appropriately encrypted

Error handling Test Cases

1. Ensure appropriate error handling according to best practice for language/script.

Logging Test Cases

1. Data does not enter an application from an untrusted source and is not written to an application or system log file preventing
   a) Injection of log events
   b) Injection of XSS to view log events
   c) Injection of scripting commands that could be executed.
2. Debug level logging is not enabled for production
3. Stacktrace is not enabled
4. Logging is at the appropriate level for monitoring (not unnecessarily verbose) and no unintended print or write to console or log is enabled (e.g. println commands)

Security Configuration Test Cases

CORS (Cross Origin Resource Sharing)

1. Ensure appropriate handling of browser Allow CORS: Access-Control-Allow-Origin to perform cross-domain http requests in web applications.
2. Ensure CORS configured correctly for the platform API Gateway as per provider instructions to specify allowed domains
3. Test for use CSRF protection method. Cross-Site Request Forgery is a vulnerability allowing third party actions on web applications
4. Test for deployment of deprecated code libraries, software versions with known vulnerabilities

Insecure Direct Object Reference (IDOR)

1. Ensure that in code object references do not contain format/pattern that can be used to mount an enumeration attack to try to probe access to the associated objects. (Enumeration attack can be described in the way in which the attacker builds a collection of valid identifiers using the discovered format/pattern and test them against the application.)

Security Configuration Enabled

1. Ensure all appropriate security configuration is enabled

Query Parameterization

1. Ensure prevention of SQL and other query injection by using parameterized queries suitable for the code/script language

Network Test Cases

The network test cases comprise checks on both inbound (ingress) and outbound (egress) connectivity.

1. Ensure network connectivity between source and destination endpoints inside CSP Virtual Private Cloud (VPC) networks
2. Ensure network connectivity from CSP VPC networks to and from the internet
3. Ensure network connectivity from CSP VPC networks to and from on-premises network
4. Ensure Kubernetes namespace pods are network security configured

The tests must cover the correct configurations of the following CSP cloud network services

1. Inbound and outbound firewalls to CSP VPC routing correctly configured
2. CSP VPC routing to Internet gateway correctly matched to NAT tables
3. CSP VPC routing to VPN gateways
4. CSP VPC routing to internal load balancers
5. CSP VPC routing to VPC peering

Integration Security Testing

End to end security testing must be designed on a case by case basis, in consideration of the application specific threat modelling.

Performance Stress and Security Testing

This is of importance as the increasing cryptographic key strength required to secure tokens, data and network connections requires some innovative thinking to ensure that latency is not a factor in rejecting cryptography as a primary security measure.

Performance Stress Testing must be designed on a case by case basis in consideration of the level of encryption required in accord with the application specific threat modelling.

User Acceptance Performance Testing

In consideration of the client requirement for security for PII and other sensitive data, user acceptance of performance is in consideration of the level of encryption required in accord with the application specific threat modelling.

Identity Management Testing

As per enterprise Identity Management Testing.

TEST ENVIRONMENT REQUIREMENTS

Test environments should as far as possible mirror production environments to ensure the relevance of testing to real deployments.

TIMEFRAME/SCHEDULES

To meet with the accelerated timeframes of rapid releases and/or continuous integration and deployment the approach taken is a just-in-time agile testing regimen.

This provides for faster results than waterfall testing methods, and within an agile delivery methodology based on collaborative team co-operation processes can produce excellent results.

The schedule for testing activities is as follows:

Description	Metrics	Expected output	Duration	Activity	Resource
Security Documents	Review	Approval	5 days	Security Architecture	Security Architects
SAST	Scan Results	Remediation of defects	5 days	Scan, Manual Review & Remediation	Security Architect & Designers
DAST	Scan Results	Remediation of defects	5 days	Scan, Manual Review & Remediation	Security Architect & Designers
Operations & Container Scanning	Scan Results	Remediation of defects	5 days	Scan, Manual Review & Remediation	Security Architect & Designers

RISKS/ASSUMPTIONS

The major risk is that security requirements of client application deployment platform are different from security requirements and context for Verviam's application development and deployment platforms.

For that reason, the following assumptions have been made for this test plan

1. Verviam to provide manual and automated security testing in an agile collaborative approach
2. Client to provide automated security testing in an agile collaborative approach
3. Client to conduct network security testing of client network with Verviam co-operation
4. Verviam and client to conduct application platform testing collaboratively on client platform
5. Client Identity Management integration testing to be conducted by client with Verviam's co-operation
6. Verviam and client to conduct separate security testing.
7. Remediation of defects for the client's application platform to be provided as soon as is practicable in accord with Verviam's security testing regimen.

TESTING TOOLS AND FRAMEWORKS

This section lists the capabilities of the automation tools and technologies to be used for security testing. The list also includes bug and defect tracking capabilities.

Defect Tracking

Bug and issue tracking software with planning, workload, search and monitoring across the Software Development Lifecycle (SDLC)

SAST (Static Application Security Testing)

Software providing automated security feedback to developers in the IDE and the pipeline, and conducts a full policy scan before deployment to ensure compliance with industry standards and regulations

DAST (Dynamic Application Security Testing)

Black box testing without a view into the internal source code or application architecture using the same techniques that an attacker would use to find potential weaknesses

Container Build and Runtime Scanning

Isolate and protect applications within containers in development and at runtime.

Operations Security Scanning

Runtime scanning, and reporting including in-depth remediation steps. Risk detection, remediation actions and continual reassessment.

REMEDIATION OF DEFECTS

Static Analysis Security Tests
See separate document

Dynamic Application Security Testing
See separate document

Container Vulnerability Security Testing
See separate document

Production Operations Security Testing
See separate document

Vulnerability Management / Penetration Testing
See separate document