Cloud Application Security

Developing Software in Cloud

- Software developed in cloud / for cloud focuses on horizontal scaling across inexpensive instances and the ability to abstract the instances from the sessions.

Application Deployment Pitfalls

- **Performance**
  - Cloud software deployment relies on loosely coupled services; performance mandates brings in more complexity
- **Scalability**
  - Cloud applications should be able to run across instances at once, retain the state of sessions and handle faults within individual instances cleanly
- **Interoperability**
  - Cloud applications should be able to work across platforms, cloud providers
  - This feature can help in reduced cost, increased options for hosting and service providers
- **Portability**
  - Applications dependency on custom components that can prevent portability between cloud providers needs to be factored
- **Availability and Reliability**
  - Provider outages and outages impacting the provider are key considerations during application deployment
- **API Security**
  - Application security and API design are key considerations when developing cloud applications

Cryptography

- Cryptography is a key element of data security in cloud
- The primary factor in security of the encryptions scheme is the safe storage and management of the crypto keys
- **Whole Instance / Full Disk encryption:**
  - Encrypts the complete systems disk / Storage
  - **Volume Encryption**
    - Encrypts only a partition on a HDD or cloud storage that is presented as volume.
  - **Opportunistic Encryption**
    - Encryption during transit, where two systems attempt to encrypt communications, failing which it will fall back to unencrypted communications

Application Virtualization

- Application Virtualization allows the application to be delinked from the underlying operating systems
- This is achieved by Application Virtualization tools that insert themselves between application and the operating system, by virtualizing the interface
- This allows for portability and segmentation while consuming fewer resources

Application Programming Interfaces

Application Programming Interfaces

- There are two common Types of APIs
  - RESTful API
  - SOAP

RESTful API

- It is a software approach designed to scale the capabilities of web-based applications
- Characteristics:
  - Low processing overhead
  - Uses Simple URL / URI
  - Not reliant on single Programming Knowledge
  - Scalable and efficient
  - Outputs in many formats
  - Uses HTTP verbs

RESTful API Use

- REST API works well:
  - When bandwidth is low
  - When stateless operations are used
  - When caching is needed

Simple Object Access Protocol

- Protocol specification for exchange of structured information in web services (as well as SMTP, FTP)
- Characteristics
  - Standards based
  - Reliant on XML
  - Highly tolerant of errors
  - Slower
  - Built-in error handling

SOAP Use

- SOAP works well for
    - Asynchronous processing
    - Format contracts
    - Stateful operations

API Models

- API is tended to be offered in the following models
  - **Public API**
    - Provided outside the organization, pay-per-use model
  - **Partner API**
    - Provided to business partners part of shared business process
  - **Private or Internal API**
    - Only for internal use and not made available for third-parties

API Threats

- Common attacks against API are:
    - Injection Attacks
    - DoS
    - Poorly secured API servers
    - On-path attacks (MTM)
    - Poor API key generation techniques

API Best Practices

- Common best practices are:
  - Authentication and Authorization
  - Validating all requests
  - Encrypting the connection
  - Logging and throttling
  - Security testing on periodic basis
  - API inventory
  - Only sending information required in an API request

API Key Best Practices

- API keys are unique identifiers used for authentication and authorization to an API
- API Key security includes:
  - Avoiding API keys in code repositories
  - Restricting their use
  - Deleting unneeded API keys
  - Regenerating long-lived keys

Supplemental Security components

Web Application Firewalls

- Used to protect Web applications by monitoring HTTP and HTTPS traffic
- WAF relies on polices to analyze traffic
- They also typically act as a reverse proxy
- Filtration is typically based on
  - User, Session information, application context and content

Database Monitoring Activity

- Combines network data and database audit information in real-time to analyze anomaly in database activities
- Used to monitor application access, privileged use and to identify suspicious activity based on behavioral analysis techniques

XML Firewalls

- Helps protect services that rely on XML based interfaces
- They provide the following capabilities
  - Validation
  - Filtering
  - Rate-limiting and managing traffic flow

API Gateways

- Used to Mange, monitor and aggregate API requests to produce results for requesting systems
- It can be used for
  - Access control
  - Throttling

  Traffic flow control

  Filtering

Cloud Application Security Broker

- Used as an interface between the customer and cloud provider
- It can be used to provide access control, data loss prevention and Threat management capabilities
- Can be deployed on-premise, hybrid or cloud-hosted model
- Heavily used by organizations that require high level of control and assurance regarding cloud useage

Software Development Phases

SDLC Lifecycle

8 Phases

- **Planning –** feasibility and cost
- **Requirements Gathering** - why, what, whom, business requirements
- **Design** – how, functionality, architecture, integration points, data flow, business process
- **Development** – coding, unit testing, source code review
- **Testing and validation** – verify and validate the software, formal testing with customers, UAT is done, Integrations done
- **Training and Transition –** acceptance, installation and deployment happens
- **Release/maintenance** – Longest phase, ongoing maintenance and operations; patching, updating and modifications
- **Decommissioning** – shut down, replacement, data preservation, sanitization

Software Development Frameworks

NIST SSDF

- **NIST Secure Software Development Framework**
  - Useful for developing secure IT systems, ICS, IoT system and other Cyber Physical Systems
  - Is organized into 4 groups:
    - **Prepare the Organization**: people, process, technology
    - **Protect the software**: Protect from tampering and unauthorized access
    - **Produce Well-secured Software**: Software with minimal vulnerabilities
    - **Respond to Vulnerabilities**

OWASP SAMM

- Used to assess the current state of secure development
- Ideal for organizations with well-established SDLC processes
- SAMM domains include
    - Governance
    - Design
    - Implementation
    - Verification
    - Operations

SDLC Models

Waterfall Model

- Sequential model ~ each phase is followed by the next phase
- Phases do not overlap and logically leads to the next
- Typically recommended for fixed scope and a known timeframe for delivery
- 6 Phase typical process
  - Gather requirements
  - Architecture and design
  - Implementation
  - Testing and validation
  - Deployment
  - Operations / Maintenance

Agile Model

- It focuses on incremental and iterative development methods that promotes cross-functional teamwork and continuous feedback mechanisms
- It is considered "light weight" – it is nimble, flexible enough to adapt
- The model focuses on small increments of functional code that are created based upon business need
- Focuses on individual interactions instead of process and tools.
- Promotes customer collaboration instead of customer negotiation
- Has ability to respond to change
- It breaks the product down into individual features that are constantly being delivered
- It focuses on user stories
- Development team can take pieces and parts of all the available SDLC models and combine them in a manner that best suits the project requirement

Agile Model – 12 Principles

- Ensure Customer satisfaction
- Welcome changing requirements
- Deliver working product frequently
- Ensure daily cooperation between developers and business
- Motivated individuals
- Face-to-face conversations
- Progress is working software
- Sustained pace of development
- Technical excellence and good design
- Simplicity
- Self-organizing teams
- Effective behavioral changes to improve efficiency

Agile Model – Specialized Terms

Term	Description
Backlog	List of features that are required to complete a project
Sprint retrospectives	Meetings at end of each sprint to discuss on the outcomes, good / bad and improvements going forward
Planning Poker	Tool for estimation and planning
Timeboxing	Agreed upon time a team works upon a specific goal
User stories	Capturing high-level requirements of the user
Velocity Tracking	Validating the work estimates with the planned estimates to see the progress of the projects.

Spiral Model

- Uses an **iterative approach** to software development
- Places **emphasis on risk analysis**
- Has 4 main phases
  - Determine objectives
  - Risk analysis
  - Development and test
  - Plan the next iteration
- Advantages
  - As more information about the project is gathered it is integrated into the risk analysis process, improve prototype, test the prototype, **allows for testing to take place early**, **allows new requirements to be addressed** as they are uncovered
- Best **suited for complex projects** that **have fluid requirements**

Rapid Application Development (RAD)

- Relies more on the use of **rapid prototyping** than on extensive **upfront planning**
- **Planning** is **interleaved** with the process of developing the software
- **Delivery** of software can happen in **½ the time** compared to waterfall method
- **Combines** the use of **prototyping** and **iterative development** methods
- Model provides input to allow for the improvement of the prototype
- **Allows for customer to be involved during the development phases**

V-Shaped Model

- **Emphasizes the verification and validation** of the product **at each phase** and provides a **formal method of developing testing plans** as each coding phase is executed
- Lays out a **sequential path of execution process**.
- **Each phase must be completed before the next phase begins**
- Requires **testing through-out the development** phases
- It is also **rigid model**, does not allow for flexibility and thus adapting to changes is more difficult and expensive
- Does not allow for handling of events concurrently
- Does not integrate iterations of phases
- It **does not contain risk analysis activities**
- **Best used when requirements are best understood,** and **scope changes are very small**

Secure Coding and SDLC

ASVS

- Application Security Verification Standard (ASVS) sets the **community standard** for testing application security controls
- Objective:
  - **To be used as a metric:** ASVS can serve as a yardstick for app developers to determine their app's level of security.
  - **To be used as guidance**: It also provides guidance regarding the security controls to be implemented so as to effectively meet the requirements.
  - **To be used during procurement**: It acts as a guideline to specify application security verification requirements in contracts when procuring tools and services.

ASVS Level 1 Basic

- Relevant for applications that don’t deal in sensitive information and are less susceptible to attacks.
- Outlines security controls that are designed to safeguard against known vulnerabilities.
- The security measures listed in this level are pen-testable and integration testable.
- Level 1 is suited for small and medium sized businesses facing no major security risks.

ASVS Level 2 Standard

- Applicable to applications that conduct business-to-business transactions.
- Helps app developers secure their applications against illegitimate access, injection flaws, validation and authentication errors.
- Ensures that the measures implemented are in line with the vulnerabilities and threats that pose a risk to the application in focus.
- Recommended by security experts for safeguarding most applications. Testing at this level requires access to source code, documentation, configuration as well as people involved in the development.

ASVS Level 3 Advanced

- Applications that deal in highly sensitive information
- App developers must embed security layers into the application right from the earlier stages
- All the security efforts must be documented and audited.
    - Examples of such applications include healthcare, Defense, finance, legal document management apps among others.

SAFECode

- Industry group that focuses on secure software development
- Provides secure fundamental practices guideline focusing on
    - Design, secure coding practices, third-party component risks, testing and validation, managing findings and handling Vulnerability disclosure process

Threat Modelling

Threat Model

- A process where potential threats are identified, categorized, and analysed
- Can be performed both pro-actively as well as reactively
- Two goals of threat modelling
  - Reduce the number of security related coding and design defects
  - Reduce the severity of remaining defects

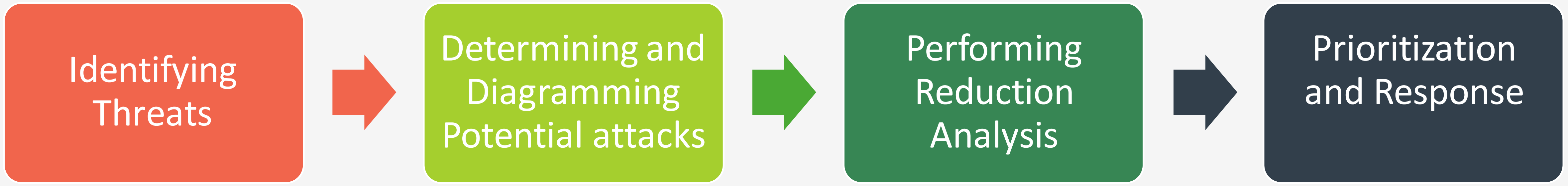
Threat Model – Proactive Approach

- Also known as defensive approach
- Takes place during early stages of systems development
- Based on predicting threats and design specific counter measures during the coding and crafting process

Threat Model – Reactive Approach

- Also known as adversarial approach
- Takes place after a product has been created and deployed
- This is the core concept behind ethical hacking, PT, source code review and Fuzz testing

Threat Model Steps


Identifying Threats
Determining and Diagramming Potential attacks
Performing Reduction Analysis
Prioritization and Response

Threat Model – STRIDE

- **Microsoft Threat categorization scheme**
  - **S**POOFING
  - **T**AMPERING
  - **R**EPUDIATION
  - **I**NFORMAITON DISCLOSURE
  - **D**ENIAL OF SERICE
  - **E**LEVATION OF PRIVILEGES

Threat Model – DREAD

- **DREAD** Rating System
  - **D**amage potential
  - **R**eproducibility
  - **E**xploitability
  - **A**ffected Users
  - **D**iscoverability

Threat Model – ATASM

- **Architecture, Threats, Attack Surface and Mitigations**
  - Seek to understand the architecture
  - List all threat agents, goals, methods and objectives
  - Evaluate the attack surface
  - Review security controls to remove the attack surfaces

Threat Model – PASTA

- **Process for Attack Simulation and Threat Analysis**
  - Define business objective
  - Define technical scope of assets and components
  - Factoring applications and identifying application controls
  - Perform threat analysis based on Intelligence
  - Vulnerability Detection
  - Analysing and modelling attacks
  - Perform risk assessment, impact analysis and develop countermeasures

Quality Assurance and Testing Techniques

Quality Assurance

- **Involves**
  - Reviews
  - Testing
  - Reporting
- Goal is to ensure that software meets the standards

Functional Testing

- **Evaluates if the software meets the functional mandates**
- Multiple testing options
  - Integration testing: to validate if components work together
  - Regression testing: to verify if any bugs are reintroduced during version changes
  - User Acceptance Testing

Non-Functional Testing

- Evaluates the quality of the software
- Test support for encrypted connections
- Looks for stability of the software, performance etc
    - Load Testing
    - Stress Testing

Software Testing Methodologies

- **Static Application Testing**
  - Evaluates the security of software without running it
  - Usually involves the use of automated tools designed to detect common software flaws, such as Buffer overflows
  - In mature development environments, developers are given access to static analysis tools and use them throughout the design, build and test process
  - Helps developers identify programming flaws and vulnerabilities.
  - Static analysis can never reveal logical errors and design flaws
- **Dynamic Application Testing**
  - Evaluates security of software in a runtime environment and is often the only option for organizations deploying applications by someone else
  - Testers do not often have access to source code
  - Dynamic testing can involve the use of synthetic testing
  - It is effective for compatibility testing, detecting memory leakages, and identifying dependencies, and for analysing software without having to access the software’s actual source code

Software Testing Methodologies

- **Interactive Application Security Testing (IAST)**
  - Analyses code for vulnerabilities while it is being used
  - **Focuses on real-time reporting** to optimize testing and analysis process
  - Often **associated with CI/CD process**
  - **Analyses the internal function** of the application while it is running
    - While DAST focuses on testing what the application is presenting to users and
    - While SAST focuses on source code testing

Software Testing Methodologies

- **Runtime Application Self Protection (RASP)**
  - Security tool is often integrated with IAST tools
  - RASP runs on a server and works whenever the application is running
  - It intercepts all calls and validates all data requests
  - It helps provide capabilities to respond to unusual application behaviour

Software Testing Methodologies

- **Software Composition Analysis (SCA)**
  - Used to track the components of a software package
  - It is frequently used with open-source components
  - Helps to identify vulnerabilities in the components and dependencies
  - Properly managed SCM can make rolling back changes possible
  - A key role of SCM occurs at deployment and during O&M release management

Software Testing Methodologies

- **Fuzz Testing**
  - Specialized dynamic testing technique that provides many different inputs to software to stress its limits and find previously unknown flaws
  - Two main categories of Fuzz Testing are
    - **Mutation (dumb) Fuzzing:**
      - Takes previous input values from actual operation of the software and manipulates it to create fuzzed input.
      - It might alter the characters of the content, append strings etc.
      - ZZUF tool automates the process of mutation fuzzing
    - **Generational (intelligent) Fuzzing:**
      - Develops data models and creates new fuzzed input based on an understanding of the types of data used by the program

Software Testing Methodologies

- **Interface Testing**
- An interface is an exchange point of data between the system/user
- Interface testing is a systematic evaluation of a given set of exchange points
- The testing should include known good and bad exchanges
- The primary task of interface testing is to build all the test cases ahead of time, document them, and then insert them into a repeatable and automated test engine.
- Interface testing is a special case of Integration testing ~ which is the assessment of how different parts of a system interact with each other

Software Testing Methodologies

- **Misuse case or Abuse case Testing**
  - Misuse case is a use case that includes threat actors and the actions they want to perform on a system
  - The misuse case is meant to threaten a specific portion or legitimate use case of our system
  - Misuse case testing helps to ensure we have effectively addressed each of the risks we identified and decided to mitigate during risk assessment phase
  - Misuse case doesn’t require to include all the possible threats to the system, but it should include the ones that was decided to be addressed
  - It is also referred to as abuse case testing
  - They are used by software developers to evaluate the vulnerability of their software to known risks

Identity and Access Management

General Aspects

- **Object**
  - A passive entity that contains information or functionality
- **Subject**
  - A user, program or process that requests access to an object or the data within an object
- **Access**
  - The flow of information between a subject and an object
- **Access control**
  - Security features that control how user and systems communicate with each other and the resources
  - Access control systems need to applied in a layered defense-in-depth method
  - It is the extremely important first line-of-defense

General Aspects

- **Identification**
  - Describes a method by which a subject claims to have a specific identity
  - It is the assertion of unique identity for a person or system
  - It is the critical first step in applying access control
  - Can be provided by the use of username or account number etc
- **Authentication**
  - Describes a method to validate a subject claims of who it claims it to be
  - Authentication involves two step process; entering the public information (identification) and then entering the private information
  - It establishes trust between the user and the system for the allocation of privileges

General Aspects

- **Authorization**
  - Providing access to an authenticated resource based on its rights
  - Identification and Authentication are “All or nothing” aspects of access control, in contrast authorization occupies a wide range of variations
- **Accountability**
  - Keeping a track of actions performed by the subject on an object
  - Identification and auditing are the key aspects for ensuring accountability
  - Accountability relies on identification and authentication, but it does not require effective authorization

Identification and Authentication

- Three general factors for authentication
    - Something a person knows (knowledge)
    - Something a person has (ownership)
    - Something a person is (characteristic)
- Use of more than one factor is called multi-factor authentication
- Multi-factor authentication is the most secure authentication mechanism

1:1 Verification

- Measurement of an identity against a single claimed identity
    - Access card

1:N verification

- Measurement of an identity against multiple identities
    - Fingerprint database

Mutual Authentication

- Two communicating entities must be authenticated to each other before passing data

Identity Proofing and Registration

- Process of collecting and verifying information about a person for the purpose of providing an account, credential
- It is performed before an account is created or the credential is issued or special privilege is granted
- It is lengthier the first time it is created
- FIPS 201-2 is the identity verification chain-of-trust for federal agencies
    - Chain-of-trust assures all parties involved, that each participating entity followed a vetting process to securely and accurately validate an individuals' identity
- Registration happens once Identity Proofing is completed

Identification Process

- Creating and issuing identity should involve 3 aspects
- **Uniqueness**
    - The identifiers must have a unique identity to be accountable
- **Non-descriptive**
    - Neither piece of the credential set should indicate the purpose of the account
- **Issuance**
    - Another authority should be providing the identity after proper verification

Single Sign-on

- Allows the user to login one time and then access resources in the environment without having to authenticate again
- SSO software intercepts requests from network resources and fills in the necessary identification/authentication information for the user
- If the attacker uncovers the credential, all access will become available
- It is also a bottleneck or single-point of failure
- It is expensive to implement in large complex environments

Common Security Vulnerabilities

Term	Description
Missing function-level access control	As the user traverses the application and accesses different functions, if the application does not verify authorization for each function, it is possible for the user to be able to elevate access, either intentionally or accidentally. The application should verify authorization as a user accesses each new function or piece of data
Security misconfiguration	Security misconfiguration refers to a system where baselines were not correctly applied, unauthorized changes were made, or security patches and updates from the vendor were not applied
Sensitive data exposure	Sensitive data exposure occurs when **protected data, such as PII or credit card information, is not properly encrypted or masked**, and is thus susceptible to attackers. This exposure refers to specific data within the application and is not directly related to specific functions of the application.
Unvalidated redirects and forwards	Unvalidated redirects and forwards occur when an application allows external links or redirects but does not properly validate or secure them. This enables an attacker to potentially redirect users through a legitimate and secure application to an external site for phishing attempts or other malware attacks. This site will appear to be safe and legitimate to the users because it originated from within a trusted application

Testing Options

Term	Description
DAST	Dynamic application security testing (DAST) is run as a "**black-box**" test where those running the test have no internal or particular knowledge of the system and must discover everything they know about it through the use of utilities. DAST **does not attempt to fully execute malicious actions**
SAST	Static application security testing (SAST) assesses both the source code and components of an application. It is done as a "**white-box**" test, as those performing the tests have full access to the actual source code and configuration documentation of the application. Also, tests are done against an offline system
Pen Test	Pen testing is also done as a "**black-box**" test but using the same tools and methodologies that an attacker would use in order to evaluate the security of the application. It is designed to test vulnerabilities in various types of real-world scenarios and to **actually execute exploits**
RASP	Runtime application self-protection (RASP) involves testing **against systems that have the ability to detect attacks and threats as well as to automatically adjust their security settings** or other configurations to compensate for and mitigate them. RASP is designed to be done in real time on live systems

Common Threats

Term	Description
Cross-site scripting	An attack where a malicious actor is able to send untrusted data to a user's browser without it going through any validation or sanitization processes. The code is then **executed on the user's browser** with their own access and permissions, thus allowing an attacker to redirect the user's web traffic, steal data from their session, or potentially access information on the user's own computer that their browser has the ability to access
Insecure direct object references	Insecure direct object references occur when a developer has in their code a reference to something on the application side, such as a database key, the directory structure of the application, configuration information about the hosting system, or any other information that pertains to the workings of the application that should not be exposed to users or the network
Injection	Injection attack is where a malicious actor sends commands through input and data fields with the intent of having the application or system execute the code as part of its normal processing and queries. This can trick an application into exposing data that is not intended or authorized to be exposed
Cross-site request forgery	The attack **forces an authenticated client** that a user has open to send forged requests under the user's own credentials to execute commands and requests that the application thinks are coming from a trusted client and user
Unvalidated redirects and forwards	Occurs when an application allows external links or redirects but does not properly validate or secure them. This allows an attacker to potentially redirect users through a legitimate and secure application to an external site for phishing attempts or other malware attacks. The site will appear to be safe and legitimate to the user because it originated from within a trusted application

Open source Packages

Term	Description
Shibboleth	It is an **open source, federated identity system** that is widely used by universities, non-profits, government agencies, and other resource- or technology-related organizations
Puppet	Puppet is an open source software package that is **used to maintain configurations on different types of systems** and operates via an agent installed on each system
Chef	Chef is a software package **for deploying and configuring servers** in a manner where configuration is treated like code and deployed to servers for execution
GitHub	GitHub is a very widely used software platform that **provides a code repository for collaborative development**, including branching and versioning capabilities

Scanning Types

Term	Description
Vulnerability Scanning	It is done **using a predefined set of signatures and parameters** to evaluate a system and give a risk rating based on its findings. It **does not have extensive knowledge of a system or go beyond the preconfigured tests** that it runs for compliance
Pen Testing	Pen testing is also done as a "**black-box**" test but using the same tools and methodologies that an attacker would use in order to evaluate the security of the application. It is designed to test vulnerabilities in various types of real-world scenarios and to **actually execute exploits**
Baseline Scanning	It is done to confirm that servers and systems have appropriately and correctly applied baseline configuration requirements. Baseline scanning does not test vulnerabilities specifically; instead, it is solely **focused on ensuring baseline compliance** and flagging any deviations for remediation
Compliance Scanning	It can refer to either baseline scanning or scanning to ensure that server configurations and policies are in place to **meet regulatory requirements**.

All the best