Domain 10: Application Security

Introduction

Application security encompasses an incredibly complex and large body of knowledge–everything from early design and threat modeling to maintaining and defending production applications. Application security is also evolving at a rapid pace as the practice of application development continues to progress and embrace new processes, patterns, and technologies. Cloud computing is one of the biggest drivers of these advancements and brings with it the urgent and imperative need to ensure that progress is stable, scalable, and secure.

From the initial design phase to ongoing maintenance, the security of cloud-based applications requires careful consideration and proactive measures. An overview of the unique challenges and opportunities presented by application security in the cloud environment (which apply to many private cloud and on-premises environments as well) is described below.

- Applications are often built as a constellation of microservices and external services, which necessitates a more detailed analysis of attack surfaces and control boundaries.
- Attack surface often includes significant exposure over APIs.
- In a cloud context, applications are often developed using development and operations (DevOps) approaches with rapid feature development, which can be a risk as well as an opportunity.
- Applications can be built on libraries that are under the control of the provider (e.g., PaaS provider, or serverless), which requires attention to the shared responsibility model.
- Applications frequently leverage third-party libraries, including open-source components, introducing supply chain risks and additional attack vectors.
- Security features, such as identity management, logging, and monitoring, are often sourced from a cloud provider, which may or may not match the application requirements.
- Applications are often deployed on programmable infrastructure (Infrastructure as Code (IaC), orchestrators such as Kubernetes, and so on).
- Applications operating at scale within cloud environments necessitate a keen awareness of the underlying infrastructure's vulnerabilities. Stateless architectures, which prioritize scalability and resilience, are commonly employed to mitigate the impact of infrastructure failures. However, while these architectures offer flexibility and agility, they also introduce complexities that can undermine the overall security posture.

Learning Objectives

The learning objectives for this domain aim to provide readers with knowledge on:

- Implement a secure development process for creating secure applications.
- Recognize the critical role of architecture in ensuring the security of cloud applications.
- Automate the integration of security throughout the Secure Software Development Lifecycle (SSDLC) using DevSecOps.

10.1 Secure Development Lifecycle

Secure applications start with a secure development process. Simply hoping safe and secure code will be created by teams of developers whose primary job is building functionality is not sufficient. To address these concerns the development and security professions have merged around a set of processes called the Secure Development Lifecycle (SSDLC) or sometimes referred to as the Secure Software Development Lifecycle (SSDLC).

10.1.1 SSDLC Stages

The Cloud Security Alliance (CSA) Development, Security, Operations (DevSecOps) *Secure Development Lifecycle*[41] (SSDLC) defines five stages that map to the commonly agreed-upon phases of application development[42]. Each of these stages identifies key processes, tools, and design patterns to be implemented in successful DevSecOps programs[43].

Figure 25: Stages of the Secure Software Development Lifecycle (SSDLC)

10.1.1.1 Stages of the SSDLC

1. **Secure Design and Architecture:** The design section references the technologies and tools that can be applied during the design of a product. We recognize that design is continuous, so new product features and changes will route through design activities. Without including security in the design phase, security measures will be introduced with a higher operating impact and cost at deployment or runtime. As a result, measures will be difficult to scale, resulting in security bottlenecks that slow development speed and impact release timelines.

[41] CSA. (2024) Secure Development Lifecycle.
[42] Similar development lifecycles have been created by software vendors, such as Microsoft's Security Development Lifecycle (SDL) that offer additional guidance of secure development practices.
[43] CSA. (2022) Specific implementation details are available in Pillar 3 - Pragmatic Implementation of the Six Pillars of DevSecOps Series.

2. **Secure Coding:** Coding security controls that rely on automation as tools can better and more consistently identify weaknesses and vulnerabilities in code compared to manual human reviews. Without including security in the coding phase, there is a risk that vulnerabilities in source code will be unidentified and deployed into production environments. Security vulnerabilities and weaknesses will be more expensive to fix later in the SSDLC.

3. **Continuous Build, Integration, and Testing:** Integration and testing include the tools and processes to security test the functionality of an application/product. Without the tools and methods, security vulnerabilities and weaknesses will be the source of exploitation and result in data breaches and availability issues.

4. **Continuous Delivery and Deployment:** Pre-deployment safety checks ensure that an application/product is deployed onto secure infrastructure. Without including security in deployment, there is a risk that vulnerabilities and poor security practices weaken an application/product, exploiting it for attacks in production environments.

5. **Runtime Defense and Monitoring:** The capabilities and practices that can be applied after an application/product has been released into production. Runtime security enables continuous improvement by identifying inefficiencies, vulnerabilities, and weaknesses and enabling incident response.

10.1.2 Threat Modeling

Threat modeling is a structured process used in risk management to identify, assess, and address potential security threats to an organization's assets. In SDLC, this process is used in the context of application risks. It involves understanding the system architecture, identifying security objectives, and analyzing the potential threats that could affect those objectives. This process helps create a more secure system design by focusing attention on areas more vulnerable to attacks. STRIDE[44] is an example of threat modeling. The STRIDE framework is used for identifying and categorizing security threats. It lists six categories of application threats. Here's a summary of each threat category:

1. **Spoofing**: Involves an attacker pretending to be someone else, such as a user or a system, to gain unauthorized access. Examples include phishing attacks where attackers mimic legitimate sites to steal login credentials.

2. **Tampering**: Refers to unauthorized alterations of data or messages. This could happen in transit or within a storage system, compromising the data's integrity.

3. **Repudiation**: Occurs when a party denies acting despite having done so. This undermines the system's ability to attribute actions to the correct source, complicating accountability.

4. **Information Disclosure**: This involves unauthorized access to confidential information. Techniques include eavesdropping on unsecured communications or exploiting vulnerabilities to access sensitive data.

[44] OWASP. (2024) *Threat Modeling Process.*

5. **Denial of Service (DoS)**: Exhaustion of system resources leading to the unavailability of the system. This disrupts operations and can cause significant downtime.

6. **Elevation of Privilege**: When an attacker gains higher access levels than permitted, bypassing access controls to execute actions reserved for more privileged accounts.

Understanding these threats allows for the identification of potential vulnerabilities in a system and guides the development of effective countermeasures to protect against these security risks.

10.1.3 Testing: Pre-Deployment

Teams can save significant time and resources by integrating testing early in the development process, specifically before deployment to the test environment. This approach allows for the early detection and correction of issues, contributing to a more secure and reliable software product.

The following are examples of key pre-deployment testing methods. All of the examples are relevant to cloud-based, as well as non-cloud-based, applications:[45]

- **Automated security code review**: This process is also called Static Application Security Testing (SAST). It is a white-box testing approach that examines an application's source code to identify existing security flaws or vulnerabilities. It is an automated way to perform a Security Code Review and might be integrated into a CI/CD pipeline (see below) or in the developer's IDE (shift-left approaches for testing can be found later on this domain). It specifically looks for logic errors, examines spec implementation, and checks style guidelines, and security flaws (like that listed by OWASP Top 10 and SANS Top 25) It also scans the code for hard-coded credentials, keys, tokens, and secrets not allowing those to enter the repository and potentially leak. SAST can be prone to false positives, which is why it will need tuning and prioritization to not alienate developers.

- **Software Composition Analysis (SCA)**: SCA involves auditing the external open-source components your software relies on, such as libraries and system components. This method ensures these components are current and free from known vulnerabilities, and the type of license of these components helps avoid bringing licensing risks into the project. SCA can also assist in creating a Software Bill of Materials (SBOM) which is required by certain standards, providing transparency on all components used in the software.

- **Secrets, Images, and IaaC templates scanning**:On top of scanning the code, there is also a need to detect application secrets embedded in code and configuration files, scan images for vulnerabilities, and detect Infrastructure as Code policy violations. Those types of tests can be made pre-deployment or after, and could be integrated with SCA and static application security testing (SAST) tools or be standalone.

[45] Non-cloud-based applications are referred to as on-premises applications. Examples for the span of applications include specialized and enterprise software, software running on desktops and local servers, custom-built applications, games, and embedded system applications.

Integrating these testing methods into the development process ensures that security is considered at every stage, from initial design to deployment, aligning with best practices for secure software development.

10.1.4 Testing: Post Deployment

Post-deployment testing verifies the security and functionality of software after it's deployed to the test environments. This phase ensures the software operates effectively in real-world conditions, mirroring traditional data center testing processes.

The following are some examples of essential post-deployment tests:

- **Dynamic Application Security Testing (DAST)**: DAST is a black box testing in which a tester examines a web application while it is running but has no knowledge of the application, its internal interactions or designs at the system level, and no access or visibility into the source program. An application's responses to these simulations help determine whether the application is vulnerable and could be susceptible to a real malicious attack.
- **Interactive Application Security Testing**: IAST is a method of application security testing that tests the application while the application is run by an automated test, human, or any activity interacting with the application functionality. IAST can be considered a combination of SAST and DAST to achieve its objective of having an overview of issues on source code and execution on runtime.
- **Penetration Testing**: Conducted as a simulated cyberattack, penetration testing aims to exploit known vulnerabilities, testing the resilience of security measures and the software's ability to withstand attacks. Pen tests can be applied using automated tools or manual work. For the long term, it is recommended to apply both. Pentest can be white box or black box, and can also be applied to test the application components level or performed at the cloud deployment level to identify flaws in the cloud configurations therefore representing a much broader spectrum of possibilities and findings.
- **Bug Bounty Program**: These programs offer monetary rewards to ethical hackers for successfully discovering and reporting a vulnerability or bug on the live application. Bug bounty programs allow organizations to leverage the ethical hacker community to improve their systems' security posture over time, and require that the organization has a mature process to respond to reports.

10.2 Architecture's Role in Secure Cloud Applications

Architecture serves as a framework for creating cloud-based solutions, for example, by allocating functionality over components and services. A well-designed architecture enhances security and improves the scalability, reliability, and overall performance of cloud applications.

10.2.1 Cloud Impacts on Architecture-Level Security

Cloud computing shifts the paradigm of traditional software and infrastructure development, emphasizing that everything is software. This shift streamlines operations and tightly integrates infrastructure with applications, requiring a new approach to security.

1. **Infrastructure and Application Integration:** The cloud provides more distributed and scalable architecture options, sharing application services like federated identities, distributed databases, and temporary workloads that spin up and down processes, as needed.
2. **Application Component Credentials:** In the cloud, components, such as microservices, communicate using specific permissions and credentials often designated just for that service. Exposure or mismanagement can lead to significant security incidents.
3. **Infrastructure as Code and Pipelines:** Defining infrastructure through code has become a hallmark of cloud practices, offering consistency and efficiency in deployments.
4. **Immutable Infrastructure:** In immutable infrastructure, once an instance of an application, a server, or a system configuration is created, it is never modified. Instead, if changes are needed, a new instance is built from a common template and replaced entirely. This approach provides additional resiliency over traditional approaches.

10.2.2 Architectural Resilience

At the scale of the cloud, no individual service or infrastructure component can be assumed to be 100% reliable.[46] This is typically done through architectural tools, such as redundancy, which involves duplicating critical components to ensure availability in case of failure; load balancing, which distributes workloads across multiple resources to avoid overloading any single component; and auto-scaling, which automatically adjusts the number of active instances based on current demand to maintain performance and availability.

The discipline of Site Reliability Engineering (SRE) plays a crucial role in ensuring the reliability and efficiency of large-scale cloud systems. It takes a comprehensive engineering perspective on operations, infrastructure, and applications. Applications should be designed and maintained to be resilient.

10.3 Identity & Access Management and Application Security

IAM plays a pivotal role in enhancing application security. It encompasses the technologies and policies designed to manage identities and regulate user access within organizations.

[46] This topic is covered in more detail in Domain 7: *Infrastructure & Networking*.

10.3.1 Secrets Management

Secrets are digital authentication credentials, such as passwords, keys, and tokens, that are used by applications or infrastructure services to communicate between them or with other services (as opposed to authentication credentials used by humans). Secrets management is the practice of securely handling those credentials. Effective secrets management ensures these sensitive elements are stored, accessed, and managed securely, preventing unauthorized access, and mitigating the risk of data breaches. It involves tools and policies to create, distribute, rotate, and revoke access credentials systematically, along with secret leak detections, thus safeguarding the integrity and confidentiality of data across the infrastructure.

Most of the cloud providers today offer alternatives to static secrets. Depending on the deployment scenario, secrets can be avoided by assigning IAM roles/identities to services. For scenarios where secrets must be used, all IaaS/PaaS providers offer secure storage services for keeping secrets safe. These integrate with IAM and eliminate the need to keep secrets in application code, configuration files, or other insecure storage. Third-party services exist as well for multi-cloud and on-prem deployments.

The following figure illustrates the process of generating and managing X.509 certificates, including the creation of a key pair, submission of a certificate signing request (CSR), and issuance by a certificate authority (CA).

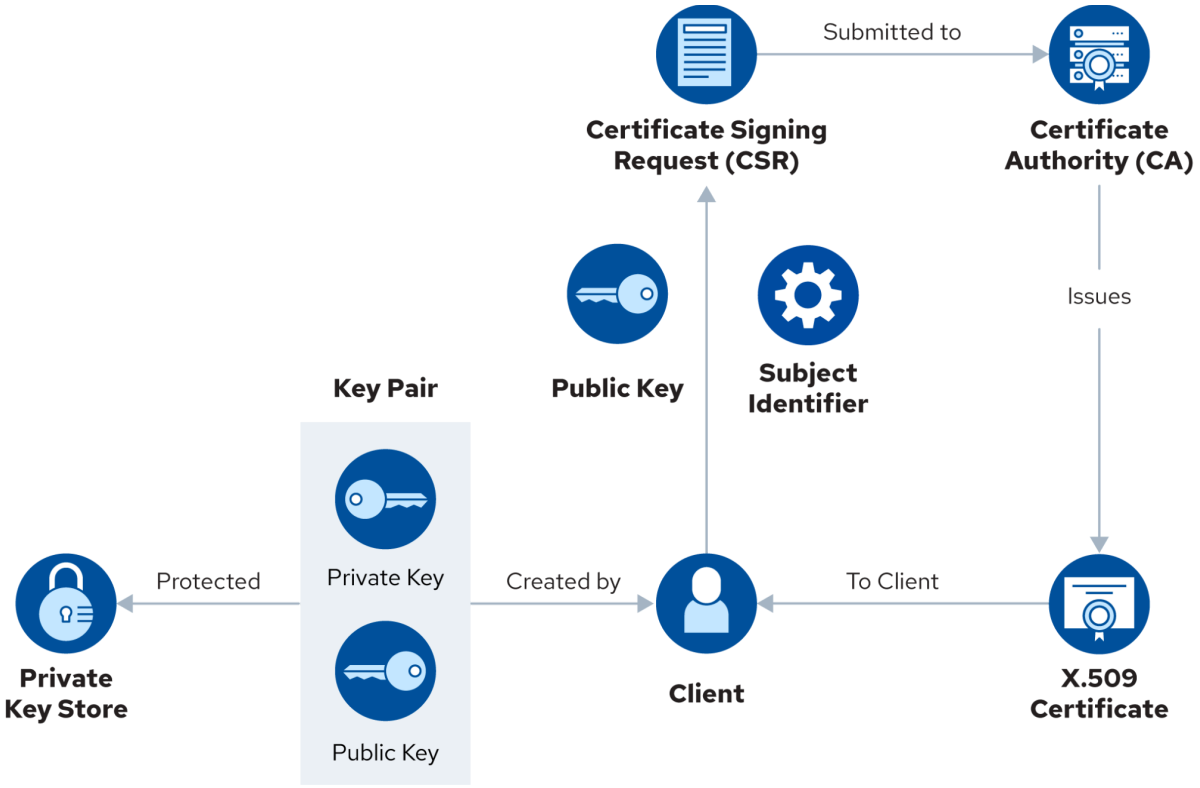

Figure 26: IAM and Secrets Management Process

10.4 Dev Ops & DevSecOps

DevOps combines software development (Dev) and IT operations (Ops) to speed up the software development lifecycle (SDLC) and deliver updates frequently. It focuses on collaboration and automation between developers and operations professionals to build and deploy code more efficiently. DevOps is widely used in cloud computing due to its agility and scalability.

DevSecOps is an extension of DevOps that integrates security practices throughout the entire software development lifecycle. It ensures that security is embedded in the development and deployment process from the beginning. To address these concerns, the development and security professions have merged around a set of processes sometimes also referred to as the Secure Software Development Lifecycle (SSDLC).

The Cloud Security Alliance (CSA) Development, Security, Operations (DevSecOps) working group released the "Six pillars of DevSecOps" which introduces the concepts of DevSecOps to security professionals.

An important concept in DevOps and DevSecOps is the automated deployment pipeline, which continuously executes security checks, scans, and tests. While these methodologies are not exclusive to cloud computing, they are commonly utilized in cloud deployments to enhance efficiency and security. Automated pipelines facilitate continuous integration/continuous delivery (CI/CD), ensuring that applications are securely and reliably deployed with minimal human intervention. This integration of security into the DevOps process helps maintain high standards of application security and compliance.

The following figure illustrates the DevSecOps CI/CD cycle, highlighting the integration of DevSecOps phases to ensure continuous and secure delivery of software.

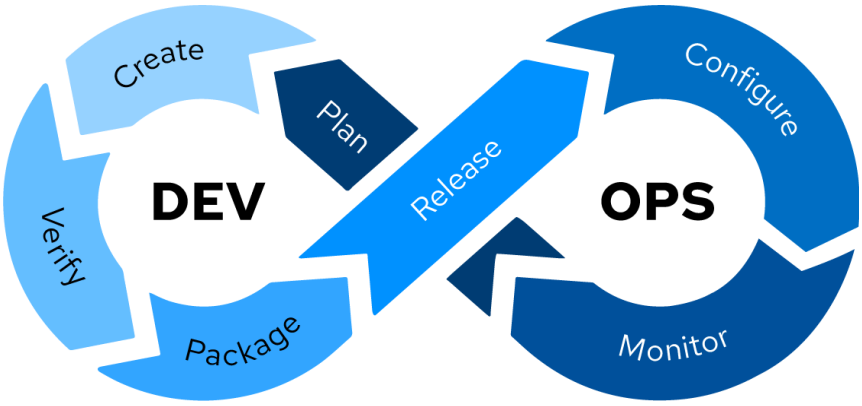

Figure 27: DevSecOps CI/CD Cycle

10.4.1 CI/CD Pipelines and Shift Left

"Shift Left" is the phrase used to indicate that "security" should move to earlier phases in the SSDLC to ensure secure-by-design and secure-by-default products. Shift-Left drives proactive security by

ensuring each phase of the SSDLC, starting from inception and planning, is viewed through the lens of security. This approach also serves to be cost-effective compared to "bolting-on security" at a later phase in the SSDLC.

Between software development and actual live production, there are many steps. Deployment pipelines aim to automate as many of these as reasonably possible. This has great benefits in reproducibility, transparency, and feature velocity. It can also be tremendously beneficial for security. The common way to group these steps gives these pipelines their name:

Continuous Integration (CI): Developers frequently merge their code changes into a shared repository. Pre-deployment automated security tests are required in this phase (e.g., SAST).

Continuous Deployment (CD): Once the code passes the CI stage, it is automatically deployed to testing or staging environments. This ensures that code changes are delivered quickly and consistently. Post-deployment automated security tests are required in this phase (e.g., DAST).

The following figure shows a simplified deployment pipeline. In reality, this pipeline can contain many more steps, such as functional testing and rolling updates.

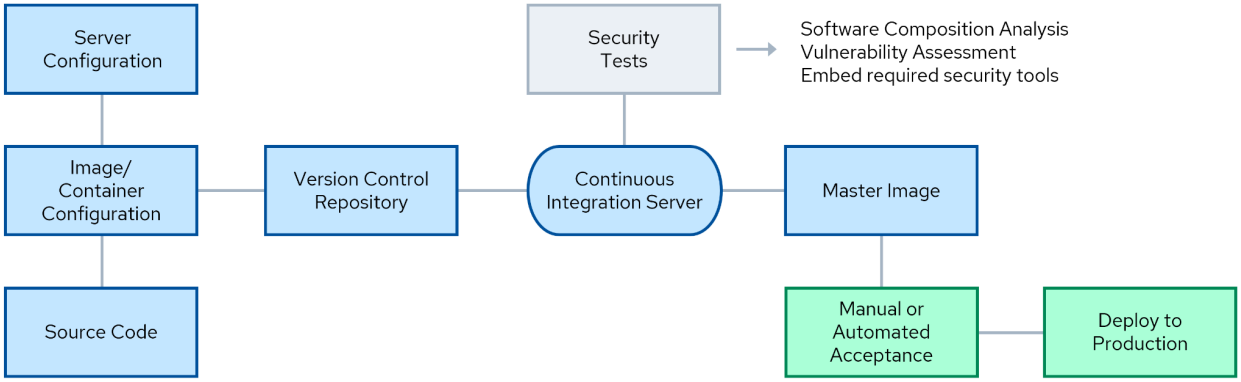

Figure 28: Secure Image Deployment Pipeline Process

The steps in the figure are:

1. **Server configuration**: The foundation of a secure image starts with a well-defined server configuration. Utilizing Infrastructure as Code (IaC), this step involves specifying the operating system, network settings, and security policies that will form the baseline of the server environment.

2. **Image/container configuration**: The focus shifts to the image or container. This stage involves packaging the application and its dependencies in a lean and secure configuration based on the predefined server setup.

3. **Source code:** The application source code (if any) that will be built and installed into the image.

4. **Version control repository**: The journey of the image configuration files continues as they are checked into a version control system. Employing tools like Git repositories or container registries facilitates change tracking, collaboration, and accountability in the build process.

5. **Continuous integration server:** Automation takes center stage here. A continuous integration service or server builds images from the configuration files and performs security checks each time changes are committed.

6. **Security testing and enforcement**: Security testing is an integrated pipeline component. With tools for static and dynamic analysis, software composition analysis, and vulnerability scanning, the pipeline identifies and rectifies security issues, fortifying the image before it progresses.

7. **Master image**: A secure master image is born. The culmination of the process results in a hardened, vetted master image that is securely stored and ready for deployment. This image could be a virtual machine (VM), a container image, or a similar deployable unit.

8. **Manual or automated acceptance**: At this juncture, the image undergoes a rigorous examination. Depending on its risk and criticality, it may pass through manual reviews or automated acceptance tests.

9. **Deploy to Production**: The master image is in the production environment. The image's transition to production is consistent, secure, and deployed with precision using IaC and automated tools.

In this approach, there is no longer a need for a process to patch systems in production. Instead, any changes will be applied upstream (to the left) and the deployed image can be immutable.

10.4.2 Web Application Firewalls & API Gateways

Web Application Firewalls (WAF) and API Gateways (GW) perform security functions on HTTP/S traffic incoming to CSC workloads. While WAF tends to secure regular website HTTP traffic against known attacks, API gateways address specific API security concerns including basic authentication and authorization actions, rate limiting, and request/response transformations. Other security tooling is often integrated into the WAF, like Distributed Denial of Services (DDOS) protection services, SSL termination, and more.

There are four common deployment scenarios for WAF & API GW protection in IaaS/PaaS services.

1. **Agent deployment**:When using IaaS virtual machines as web servers, a WAF Agent can be installed on top of the operating system. This option usually doesn't have DDoS mitigation capabilities.

2. **Cloud provider service**: IaaS/PaaS providers offer integrated WAF and DDoS protection services, usually deployed on the load balancer services.

3. **Third-party marketplace service**: IaaS/PaaS marketplace offers a wide variety of third-party commercial WAF software deployed on dedicated VMs. It is the customer's responsibility to deploy the WAF and ensure routing, redundancy, and load balancing.

4. **WAF and DDoS as a Service**: Using DNS redirect, consumer traffic is routed to a third-party WAF service, examined and filtered, and then routed to the cloud provider environment.