Guide to Application Security Concepts

Published 29 July 2020 - ID G00729065 - 40 min read

By Analysts Michael Isbitski, Frank Catucci

Initiatives: Security Technology and Infrastructure for Technical Professionals

This guide for security and risk management technical professionals helps those new to the application security domain form a baseline of concepts and technology. It also helps experienced readers understand new or emerging concepts to expand their application security strategy and architecture.

Overview

Key Findings

- Organizations often lack formalized SDLC processes and standardized technology stacks, creating significant barriers to establishing, scaling and maturing an effective application security program.

- Application security testing (AST) tools exist in dynamic, interactive and static analysis variants, each carrying inherent strengths and weaknesses. Adopting traditional testing approaches late in the SDLC increases developer friction and reduces the success rate of application security efforts.

- Secure design and security testing are just two critical facets of application security programs. Organizations must also protect applications as part of enterprise architecture and during runtime (with mechanisms external to code). Application attack patterns include denial of service, exploits, abuse, access violations, and reverse-engineering and tampering.

- With the rising adoption of cloud-native design patterns and automation, containers have become the preferred compute model for powering modern applications, blurring the lines between the application and infrastructure domains. This paradigm creates a growing attack surface and a complex set of security risks that impact production applications.

Recommendations

Security and risk management technical professionals focusing on application security:

- Start with a skeleton application security program if time and resources are at a premium. Focus these initial efforts on protecting exposed applications and on implementing a base level of security testing. Then, flesh out other program elements over time.
Application security continues to be one of the most requested topic coverage areas for the security technology and infrastructure initiative. Organizations are building or integrating applications with increasing frequency, and they’re also pushing hard to automate as they adopt DevOps practices. They must build, integrate and automate with security in mind to protect employee, customer and citizen data.

This guide will help you understand the core concepts around application security — from people and process, to the technology needed at each of the layers to keep pace with your application security strategy. Figure 1 shows the different areas of coverage for the application security domain. These are mapped to application types and platforms. A given application security process or technology area may map to all types, or it may map only partially. For example, with in-app protection, the bulk of technology resides in client-side code but may still use server-side elements (e.g., mobile app authentication).
The sections that follow provide introductory descriptions for each item, along with links to more detailed information. At the end of this research is a glossary of common terms that come up in application security discussions.

**Figure 1: Gartner for Technical Professionals Application Security Coverage**

<table>
<thead>
<tr>
<th>People and Process</th>
<th>Application Security Programs and Secure SDLC</th>
<th>Secure Design and Threat Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foundational Security</td>
<td>Secure Coding Practices</td>
<td>Security Coaches and Champions</td>
</tr>
<tr>
<td>Types and Platforms</td>
<td>DevSecOps</td>
<td>Secure Continuous Delivery</td>
</tr>
<tr>
<td>Advanced Security</td>
<td>Mobile Application Security</td>
<td>In-App Protection (Mobile)</td>
</tr>
<tr>
<td></td>
<td>JavaScript Security</td>
<td>Runtime Application Self-Protection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Secrets Management</td>
</tr>
<tr>
<td></td>
<td></td>
<td>API Security and API Gateways</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Web Application and API Protection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Container Security</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kubernetes Security</td>
</tr>
<tr>
<td></td>
<td>Application Security Testing</td>
<td>Software Composition Analysis</td>
</tr>
<tr>
<td></td>
<td>Web Application Firewalls</td>
<td>Containers</td>
</tr>
<tr>
<td></td>
<td>Front Ends (JS and Mobile)</td>
<td>Kubernetes</td>
</tr>
</tbody>
</table>

**Foundational Security**

This aspect of application security is considered critical and impactful for the large majority of organizations. Resources invested here will either have an amplifying effect across IT projects or serve as essentials to provide some degree of application security within an organization. The respective subtopics include:

- Container Security
- Kubernetes Security
- Application Security Testing
Container Security

Modern applications are often packaged as container images and instantiated as running containers on infrastructure in data centers or cloud providers. Client interest in container security has shifted from the segregation capabilities of containers to the complexity of the container ecosystem, which blends the application and infrastructure domains. Figure 2 illustrates a container life cycle that includes an automated continuous integration/continuous delivery (CI/CD) deployment pipeline and new threat vectors that emerge in build time, delivery time and runtime. These deployment chains are composed of many elements running on different platforms from different vendors, raising concerns about end-to-end controls. Although CI/CD deployment pipelines are instrumental for the adoption of containers, their security properties are often not fully known, given how heterogeneous a complete build and release pipeline often is. Client concerns often center around questions such as:

- What security is offered natively by a container engine or platform provider?
- What third-party security augmentations, such as CWPPs or container image SCA, are considered essential for production workloads?
- Where do infrastructure layers and application layers begin and end so that protection can be deployed effectively?

This topic is examined further in the research “Containers: 11 Threats and How to Control Them.”

Figure 2: Threat Vectors in the Container Life Cycle
Kubernetes Security

Kubernetes (K8s) provides a platform for operating application workloads as containers at scale. It underpins many cloud provider offerings, and it enables organizations to deliver web-scale applications to support their employees and customers. It is often the heart of cloud-native design. Unfortunately, the technology also brings a great deal of complexity. This indirectly and inadvertently introduces new attack vectors, some of which are depicted in Figure 3. Kubernetes is also rapidly evolving software, with new features being added and bugs being fixed regularly. Best practices for implementing and operating K8s, let alone securing it, are still emerging. Many practitioners and Gartner clients are new to K8s, adopting it as they migrate applications and compute out of their data center into cloud provider environments. Application development teams also frequently adopt K8s to support their microservices architectures.

Figure 3: Threat Vectors in Kubernetes Clusters
Kubernetes’ inherent complexity often leads to outdated versions and misconfiguration by organizations, making clusters susceptible to compromise. Though some security mechanisms are included by design, K8s by itself is not a security offering, and security settings aren’t always enabled by default. Protecting a K8s cluster is a significant undertaking, requiring both substantial understanding of the underlying technology and engineering expertise to configure it all.

“Guidance Framework for Securing Kubernetes” presents six focus areas for securing production K8s deployments:

1. **Cluster foundations** — Selecting and securing container runtimes, using native container security controls, and enabling logging mechanisms to support security activities.

2. **Secure continuous delivery** — Securing the code repositories, binary repositories and container registries that house the variety of application and infrastructure artifacts for K8s
3. **Secrets management** — Leveraging the native secrets management functions of K8s Secrets, backed by etcd, and augmenting with third-party tooling for higher-security use cases.

4. **Secure master node** — Securing the K8s API server, which controls all aspects of a cluster, and etcd, which provides persistence and service discovery in K8s.

5. **Secure worker nodes** — Securing kubelets that communicate with master node components, implementing K8s Pod Security Policies and using custom admission controllers.

6. **Secure network communication** — Securing ingress, using K8s Network Policy to secure microservices communication, enabling Transport Layer Security (TLS) and expanding with third-party protection.

### Application Security Testing

Application security testing (AST) is a critical practice and set of tooling within the secure software development life cycle. It covers multiple techniques, from early development stages through to, and including, production. Efficient, successful AST relies on knowing when and how frequently to test during the SDLC, as well as which technology is the best fit for a given stage. This knowledge is also critical as an organization embraces agile development methodologies and DevOps practices, where security testing needs to be integrated into the development workflow and automated as much as possible.

The forms of AST include:

- **Static application security testing (SAST)** — Analyzing application code in various ways, such as expression matching, execution flow analysis and data flow analysis.

- **Dynamic application security testing (DAST)** — Testing for vulnerabilities by interacting with web applications or APIs in real time via URLs or API endpoints. DAST tools will passively analyze HTTP traffic or actively manipulate requests and responses in order to find exploitable conditions.

- **Interactive application security testing (IAST)** — Testing for vulnerabilities by blending static and dynamic analysis techniques through instrumentation of the running code. Some vendor approaches also involve the use of SDKs or libraries embedded in code.

In lieu of AST, some organizations opt to rely on more traditional practices like vulnerability assessment and security configuration assessment as basic testing measures. They will also augment their application security strategy with protection mechanisms like a web application firewall (WAF) at the perimeter of their cloud segment or on-premises data center. Every organization should test the application code they create with AST tools fit for purpose, but not all do.
The practice of AST skates a thin line between foundational and advanced security, but this dynamic can also be a factor of conflicting security priorities in organizations. The reality is that implementing AST where it is properly integrated and automated as part of application development and release is challenging for many organizations. Organizations should strive to employ at least one type of AST to analyze their custom applications for vulnerabilities and weaknesses. They should also aim to integrate and automate any AST into their build pipelines, irrespective of the development methodology. Figure 4 provides a process overview of a traditional software build-and-release pipeline, annotated with the phases and types of AST.

This topic is examined further in the research “How to Deploy and Perform Application Security Testing.”

**Figure 4: Application Security Testing in the Phases of an SDLC**

**Application Security Testing in the Phases of an SDLC**

Software Composition Analysis

In modern application development, a significant portion — or, usually, the majority — of a codebase consists of external dependencies, such as open-source components, libraries and

Gartner, Inc. | 729065

Page 8/35
frameworks. This holds true for both in-house-developed applications and off-the-shelf software. Open-source libraries can carry significant vulnerabilities, and due to reuse, potential impact can be very large (e.g., Apache Struts and remote code execution vulnerabilities).

License constraints can also be an issue with OSS component use, depending on whether the software is for internal employee use or for public consumption and resale. SCA tools analyze an application package to provide an inventory of the publicly available OSS components contained within, sometimes referred to as a “bill of materials.” This analysis includes identification of license types as well as known vulnerabilities that map to common vulnerabilities and exposures (CVEs).

SAST and IAST offerings typically include some level of SCA capability, by nature of having deep insight into the underlying code. In addition, dedicated SCA tools exist that can be:

- Run ad hoc via command line interface or UI, such as to scan software packages acquired from internet sources
- Integrated with developer IDEs to enable ad hoc component analysis as part of nonsecurity development workflow on developer workstations prior to code check-in
- Integrated within version control systems, typically Git-based and cloud-hosted, to scan codebases as development teams commit code
- Integrated with artifact repositories to scan binary artifacts, external dependencies used in the application or application infrastructure builds
- Integrated with container registries to scan container images during build time and prior to container instantiation

Figure 5 illustrates the typical flow for SCA, showing what the target scan types are, where vulnerability data originates from and what the resulting output consists of.

This topic is examined further in the research “How to Deploy and Perform Application Security Testing.”

Figure 5: Software Composition Analysis
Web Application Firewalls

A web application firewall protects applications by filtering and monitoring HTTP traffic between the application and the device, user or automated interactor with which the app communicates. WAFs protect applications from attacks such as XSS, CSRF, SQLi and other Layer 7 (application layer) attacks, exploits or abuses. Some WAFs also offer bot mitigation (abuse) and DDoS protection, depending on the vendor, and depending on whether the WAF is part of a web application and API protection (WAAP) offering. They are employed for homegrown applications as well as acquired or commercial off-the-shelf (COTS) applications. They are commonly employed to protect internet-facing, public web applications and web APIs. They can also be used to protect web applications and web APIs designed for internal employee use or shared with trusted business partners.

WAF capability may be integrated into or offered through other components, such as API gateways, ADCs and/or CDNs. There are also variations of WAF product design that use a combination of host-based and network- or cloud-based components. WAFs exist as traditional, physical appliances, but also as virtual machines or cloud services. They can also be hosted on-premises, in the cloud within an IaaS provider, or in some hybrid combination of the two. Deployment options vary by vendor and product, and may also impact the available functionality. Typically, the options are one of the following:
Architectures on which applications are deployed and hosted are often varied, changing with, and increasing in, complexity. As such, the means with which teams protect these applications are of equal complexity, particularly in the case of modern hosting and deployment patterns.

Web application firewalls are covered further in the research “Decision Point for Deploying WAFs for Application Protection” and “Protecting Web Applications and APIs From Exploits and Abuse.”

Advanced Security

This is an aspect of application security where organizations sometimes deviate in how they invest budget for tooling and employee resources. The respective technology areas may be emerging, niche or complex to implement successfully. Organizations may be selective in these focus areas, depending on the exposure of their applications or customer usage patterns. The respective subtopics include:

- Mobile Application Security
- In-App Protection
- JavaScript Security
- Runtime Application Self-Protection (RASP)
- Secrets Management
- API Security and API Gateways
- Web Application and API Protection

Mobile Application Security

Mobile applications are a mixture of different types of client- and server-side code, including one or more of the following:

- Client-side code:
  - A native rich client or binary that must be installed onto the mobile device. The native app is a mixture of compiled code and supporting files bundled into a platform-specific application
Mobile application security is a superset of application security testing and client-side code protection, the latter of which is covered in the In-App Protection section. The type and depth of testing can vary greatly among product and service types, but testing falls into one of three categories:

- **Package.** This is an Android Package (APK) in the case of Google Android-based devices and an iOS App Store Package (IPA) in the case of Apple iOS-based devices.
  - JavaScript, in the case of a hybrid mobile application (which also makes use of a lightweight rich client) or a web application formatted for mobile device screens.

- **Server-side code:**
  - Web APIs to provide application functionality. These APIs are commonly REST-based, but they sometimes use XML-RPC or other formats. Web APIs are often a mix of custom first-party-developed APIs along with third-party APIs.
  - Web applications, in the case of hybrid mobile applications or web applications formatted for mobile.

Mobile application security is a superset of application security testing and client-side code protection, the latter of which is covered in the In-App Protection section. The type and depth of testing can vary greatly among product and service types, but testing falls into one of three categories:

- **Mobile-application-focused AST** — Sometimes labeled as “MAST” and often provided as an assessment type by AST SaaS vendors, this service uses some combination of traditional SAST, DAST and IAST approaches to analyze both client- and server-side components. It is often supplemented with manual testing to uncover other potential weaknesses or vulnerabilities. It is also the most time-intensive approach to mobile application security testing, but the most comprehensive.

- **Mobile application reputation service (MARS)** — MARS typically decomposes and analyzes Android APK or iOS IPA binaries, as opposed to original source code, to identify risky behaviors, such as communicating with untrusted hosts or storing sensitive data on-device insecurely. Depending on the vendor, MARS may scan apps published to the respective public app stores, or it may support uploading of a mobile binary for ad hoc analysis. The scanning is typically client-side-focused, with some passive analysis of communications to server-side components. Some vendors may also perform sandboxed dynamic analysis, running and analyzing the mobile app on-device or in an emulator. Scans are designed to complete quickly to enable vetting of third-party mobile apps within an enterprise, often to support allow lists within an enterprise mobility management (EMM) app catalog.

- **Mobile threat defense (MTD)** — Also a protection mechanism, these agent-based security solutions are designed to run on mobile devices such as Android and iOS. MTD solutions also employ some type of MARS-like capability to quickly analyze and catalog risky mobile app behaviors, often supplementing with their own internal security research. MTD solutions combine that application risk intelligence with device-level information, such as OS security...
patch levels, and networking details, such as connections to unsecure Wi-Fi. This combination of techniques provides a holistic approach to securing fleets of mobile devices and preventing them from running unsecure mobile apps. For additional research into the MTD area, see “Market Guide for Mobile Threat Defense” and “Advance and Improve Your Mobile Security Strategy.”

In-App Protection

In-app protections address and reduce the likelihood of client-side reverse-engineering, debugging, and code or functionality manipulation. These client-side attacks can potentially compromise an application or steal sensitive data from the associated resources of the application. These protections are split into three categories:

- Application shielding:
  - Prevention (or deterrent) of reverse-engineering, tampering or debugging on client-side code installed on mobile devices. App shielding support for Google’s Android and Apple’s iOS is common, though some vendors also cover other binaries, JavaScript and so on.
  - Code-level obfuscation and integrity checking.
  - Rooted-OS, compromised-device and emulator detection capabilities.
  - White-box cryptography capabilities that hide keys in an “untrusted” application, protecting any keys in memory. Vendors may also inherently provide a level of anti-automation by nature of the implementation (e.g., SDK to enforce authentication).
  - Postcoding controls.

- Application wrapping:
  - Protection applied postcompilation that essentially safeguards or “wraps” the binary. Wrapping may also extend the functionality of the original application.
  - Integration with unified endpoint management (UEM), enterprise mobility management (EMM), mobile device management (MDM) and mobile application management (MAM) for managing an organization’s mobile devices and the applications installed on them.
  - Extended application functionality, such as SSO/IAM integrations or certificate pinning to prevent man in the middle (MITM) attacks on encrypted transit.
  - Binary obfuscation.

- In-app mobile threat defense (MTD), MTD delivered as an SDK or MTD delivered as part of MAM:
These technologies may also help enforce application-level authentication by verifying the integrity of client-side code and establishing a trusted channel with server-side components. The topic of in-app protection is covered further in the research “Protecting Web Applications and APIs From Exploits and Abuse.”

JavaScript Security

Some organizations and Gartner clients express growing concern around JavaScript threats like ad fraud, credit card skimming and Magecart attacks. Once content is served to a requesting user or machine, JavaScript is parsed and executed client-side by the web browser or mobile WebView (i.e., the user agent). This code execution is not visible to network-based or server-side controls. Although a given piece of JavaScript code may be partially first-party code, it is also frequently outsourced to a third party, or it often contains its own transitive dependencies. Any traffic that is sent to third-party URLs will likely never be proxied by your network-based controls. Some other variations on how JavaScript makes its way into an organization’s applications include:

- In-line frames (iframes), either visible or hidden, that will in turn invoke JavaScript
- Payment-processing JavaScript
- First-party or third-party chatbots that use JavaScript
- First-party or third-party scripts used to track users and support analytics or marketing

HTML5 and HTTP security headers can be used to control behavior of client-side JavaScript, including X-Frame-Options and Content Security Policy. The unfortunate reality with these headers is that many are not deployed widely. Either the headers are too complex to enforce at scale or the organizations are unaware of the potential dangers and security risks of JavaScript in a digital software supply chain. Blocking execution of JavaScript entirely is not a viable option, because it either breaks application functionality or impedes regular business. Some of the methods that security vendors are using to mitigate these types of attacks include:
The topic of JavaScript security is covered further in the research “Protecting Web Applications and APIs From Exploits and Abuse” and “Decision Point for Securing Application Architecture.”

Runtime Application Self-Protection

RASP offers high-accuracy attack mitigation, especially for injection-type attacks, by virtue of its deep visibility into the application stack and its instrumentation approach. It also scales and moves with applications through its integration into the runtime environment (RTE) or instrumentation libraries, reducing some of the WAF challenges, but not replacing WAFs outright for all capabilities. RASP capabilities focus on interpreted languages (i.e., those that compile into bytecode or some variation of it) and take either an SDK or host-based agent approach. SDKs require that developers integrate the RASP function calls directly within the codebase. Host-based agent approaches require that organizations install and instantiate a separate agent or sensor on the application server.

When considering requirements and vendor offerings, you must account for support of:

- Runtime environments
- Application server types and versions
- Application framework types and versions
- Middleware and database types and versions

For a load-balanced, n-tier application, RASP requires installation and maintenance of agents on all tiers and servers, creating a potential operational burden for infrastructure teams. While the agents are designed to consume minimal resources (in the range of low-single-digit percentages), there is still the concern that they can degrade performance of a given application and application server in production. When RASP agents are combined with other non-RASP agents that may be instantiated on a given application server, that application server runs the risk of performance degradation or compatibility issues. While vendors do try to test and account for these scenarios, a given application setup and environment can be incredibly complex and beyond the realm of normal test setups. Like with any host-based technology, proofs of concept and evaluations are...
critical for RASP solutions, especially because they can negatively impact production application usage.

Client-side exploit and abuse protection is typically out of scope for RASP capabilities and the realm of in-app protections. RASP, in general, is not appropriate for most types of DoS mitigation and certain anti-automation approaches. Some vendors will mitigate against certain Layer 7 cases, such as brute forcing and account takeover. The topic of RASP is examined further in the research “Protecting Web Applications and APIs From Exploits and Abuse” and “Decision Point for Securing Application Architecture.”

Secrets Management

Secrets management has emerged as a set of practices and associated tooling to automate secure creation, storage, retrieval and revocation of the secrets necessary in cloud-native application architecture. The term “secret” is used broadly to refer to a variety of data types, including:

- Machine (or service) accounts and passwords
- One-time passwords
- Authentication tokens
- Private-key portions of certificates used in Transport Layer Security (TLS) encryption
- Key material for symmetric encryption

Figure 6 shows a typical secrets management flow for a secrets-vault-aware workload that needs access to some back-end DBMS credential.

**Figure 6: Example Database Access Secrets Management Workflow**
Secrets management tooling is an umbrella of multiple capabilities, usually consisting of:

- Cryptographically secure vaults. Tools may also support integration with hardware security modules (HSMs) for higher security.
- CLIs and GUIs to interface with vaults interactively, such as for administration purposes.
- REST APIs to integrate disparate metadata formats, application platforms and native authentication mechanisms. These APIs are particularly useful in hybrid or multicloud scenarios.
- SDKs for inclusion in application source code to enable secrets management client capabilities in custom applications.

Specifically in the area of modern application development and architecture, secrets management provides functionality to avoid embedding or hard-coding secrets data into source code, build scripts, IaC and so on. The capabilities can dynamically allocate whatever secret is necessary for a given application service or workload (such as a container or a Kubernetes cluster) to instantiate and operate. Secrets management also provides the ability to periodically rotate or revoke secrets. This capability reduces the potential risk of an attacker compromising longer-living secrets or workloads. The topic of secrets management is explored further in the research "Guidance"
API Security and API Gateways

API security is largely a strategy organizations should employ, though some vendors have specific product offerings that help organizations address multiple aspects of an API security strategy. Specific focus areas of API security are depicted in Figure 7 and include:

- **Design** — Ensuring APIs are built using secure design principles and secure coding practices to avoid potential weakness or vulnerabilities. For acquisition and integration scenarios, this typically involves attestation from suppliers of their process, but may still entail additional testing from the consumer. APIs should also be validated and tested regularly, particularly those that an organization builds.

- **Discover** — Continuously scanning and inventorying APIs in the environment, including first-party and third-party APIs. APIs also need to be assessed and classified to determine the appropriate level of monitoring and protection.

- **Monitor** — Mediating APIs to maintain visibility, typically accomplished with the use of API management and API gateways. Monitoring helps an organization observe API usage patterns and form baselines. It also helps inform logging, alerting and incident response processes.

- **Protect** — Protecting APIs through a mix of technology. Security protections provided by API gateways often need to be paired with application protections provided by web application firewalls and bot mitigation services. Additionally, emerging API-focused protections may be necessary in newer architectures or environments with rapidly changing APIs.

**Figure 7: API Security Strategy**
Properly securing APIs requires more than secure design, secure coding and a WAF. Most development teams are familiar with basic security, such as authentication and the use of TLS, but more advanced security requirements have to be added externally. API gateways and API management solutions focus primarily on the development and management of the APIs. However, they also offer varying levels of the following security features, which mostly focus on access control and some mitigation of abuse:

- Security management, in particular the management of access control policies, encryption policies and basic threat protection policies.
- Authentication using access tokens, basic authentication and X.509 certificates, as well as application identification via API keys. Integration with IAM systems is common and considered a best practice.
- Authorization using policy definitions, as well as basic abuse prevention in the form of rate limiting and IP blocking.
- Content inspection and data transformation.

---

### API Security Strategy

<table>
<thead>
<tr>
<th>How Do I Secure My APIs to Mitigate the Risk of Incidents and Breaches?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Design</strong></td>
</tr>
<tr>
<td>Secure Design</td>
</tr>
<tr>
<td>Acquisition and Integration</td>
</tr>
<tr>
<td>Security Testing</td>
</tr>
</tbody>
</table>

Source: Gartner

729065_C
API gateways and API management solutions are platforms in and of themselves, and work alongside IAM and other security solutions. Most are available as a cloud service, and some can also be deployed on-premises or in IaaS as a virtual appliance. The topic of API security strategy and API gateways is covered further in the research “Solution Path for Forming an API Security Strategy.”

Web Application and API Protection

Not all organizations or their applications are subject to the same level of threats and attacks, and application architectures will vary. Web application and API protection (WAAP) combines WAF, DDoS protection, bot mitigation and API security capabilities to address four distinct categories of application attacks: DoS/DDoS, exploits, abuse and access violations. Of the four categories, only exploits can be potentially addressed by developers performing secure coding and configuration. The others require design-level considerations that cannot be reasonably compensated for in code by developers. For example, although it’s arguably possible to defend against account takeovers in individual application code, it is much more economical and error-proof to do so in an integrated identity and access management (IAM) system or another external capability.

Bot mitigation, sometimes referred to by vendors as “anti-bot” or “anti-automation,” can be used to reduce or block automated attacks and abuse. Its primary focus is on the “abuse” attack category, but it can also cover “exploit” attack types in some cases. Bot mitigation solutions employ a combination of techniques, including, but not limited to, the following:

- Client-side fingerprinting of web browser and mobile app consumers
- Seamless user-agent challenges
- HTTP traffic analysis and behavior anomaly detection
- Evaluation of IP reputation, especially against known malicious bots or bot networks
- Dynamic rate limits
- Client-side SDKs to provide cryptographically secure client/server authentication

The reference architecture in Figure 8 depicts WAAP deployed at the edge to protect applications hosted in inner architecture that might include PaaS or CaaS deployments.

Figure 8: WAAP Deployed at the Edge
The topics of WAAP and bot mitigation are covered further in the research “Decision Point for Securing Application Architecture” and “Protecting Web Applications and APIs From Exploits and Abuse.”

People and Process

A great deal of application security is underpinned by technology, since activities must support modern application design and technology stacks. However, no application security effort can succeed without defined processes and support from a larger program. Specifically, the areas of security practice with people and process include:

- Application Security Programs and Secure SDLC
- Secure Design and Threat Modeling
- Secure Coding Practices
- Security Coaches and Champions
- DevSecOps
- Secure Continuous Delivery
Application security requires a structured, programmatic approach to deal with the seeming chaos of new technology and an evolving threat landscape. Numerous frameworks and maturity models have been published that organizations can leverage to tackle the challenge. Many frameworks and models overlap in how they map application security activities to software development life cycle (SDLC) processes and application architectures. That is, they overlap in how they recommend that practitioners should secure the design, build, release and maintenance of applications and supporting environments. However, SDLCs often lack concise and actionable activities, neglect recent industry trends, or exclude certain aspects that Gartner considers essential to success based on client interactions. As a result, the secure SDLC (S-SDLC) is a subset of larger application security program work.

“A Guidance Framework for Establishing and Maturing an Application Security Program” outlines six areas of practice, as depicted in Table 1:

1. **Preparation** — Preparing for a program by identifying owners, defining metrics, and launching training and awareness efforts

2. **Program foundations** — Laying the groundwork for the program by formalizing communication processes, identifying collaboration mechanisms and defining SLAs

3. **Requirements management** — Incorporating elements of risk assessment to steer secure design, establishing threat modeling processes and ensuring requirements are mapped to engineer workflow

4. **Technology architecture** — Leveraging architectural components to provide application security beyond just code-level focus and hardening the infrastructure on which applications run

5. **Secure coding practices** — Providing specific coding guidance to engineers for the technology stacks they build for and providing mechanisms for open-source hygiene

6. **Security testing and validation** — Verifying that the code and infrastructure created align with security requirements through various forms of security testing tooling

<table>
<thead>
<tr>
<th>Areas of Practice</th>
<th>Supporting Activities</th>
<th>Enabling Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table 1: Application Security Program Guidance Framework</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Gartner, Inc. | 729065
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope the effort</td>
<td>Establish mechanisms for communication</td>
<td>Establish threat modeling process</td>
<td>Use trusted components</td>
</tr>
<tr>
<td>Identify owners</td>
<td>Establish cross-team collaboration</td>
<td>Include corporate policy, regulatory restrictions and security best practices</td>
<td>Incorporate related security technology and controls</td>
</tr>
<tr>
<td>Define metrics</td>
<td>Integrate with SDLC or vulnerability management processes</td>
<td>Raise accessibility and availability of requirements</td>
<td>Harden application infrastructure and runtime environment</td>
</tr>
<tr>
<td>Launch training and awareness</td>
<td>Define SLAs</td>
<td></td>
<td>Leverage API mediation and management</td>
</tr>
<tr>
<td>Identify champions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review maturity models and benchmark</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formalize the SDLC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maturity models</td>
<td>Application development life cycle management</td>
<td>Application security requirements and threat management</td>
<td>Vetted SDKs and libraries</td>
</tr>
<tr>
<td>Standards and guidelines</td>
<td>Version control systems</td>
<td>Application development life cycle management</td>
<td>Production scanners</td>
</tr>
<tr>
<td>Computer-based training, interactive labs and gamification</td>
<td>Collaboration platforms</td>
<td>Threat modeling</td>
<td>Identity and access management</td>
</tr>
<tr>
<td></td>
<td>ChatOps</td>
<td></td>
<td>Production security controls</td>
</tr>
<tr>
<td></td>
<td>Vulnerability management</td>
<td></td>
<td>Infrastructure and image hardening baselines</td>
</tr>
</tbody>
</table>
Secure Design and Threat Modeling

Secure design and threat-modeling activities are increasingly prevalent where an organization’s focus is on “building security in.” Unfortunately, drafting sets of security requirements, gathering security requirements for specific applications, assessing risk for a given release and modeling threats are largely manual activities, even with available tools. The challenge is exacerbated with modern application design, where you may be dealing with hundreds of interconnected web APIs, as well as virtualized systems distributed on-premises and in the cloud. A given architecture diagram may devolve into a barely legible spaghetti chart.

Threat modeling, while a valuable application security exercise, is not easily integrated into a secure DevOps toolchain. It is largely a manual process, and dedicated tooling to programmatically analyze an application design and all its interactions — from end to end, with accuracy — is still mostly aspirational. As a result, many organizations skip threat modeling entirely, even in traditional waterfall development and non-DevOps approaches. However, the resulting knowledge of the target system and the development of skill in threat/vulnerability identification are important
processes to adopt into an organization's overall secure DevOps program, and not just the automated pipeline. Threat modeling can also benefit application security training and awareness, which enforce principles of secure design for application teams. If individuals can rapidly identify a weakness as they're creating or configuring an application, they can proactively mitigate it without the need to test for or protect against it later.

Lower-maturity organizations performing threat modeling might use Microsoft STRIDE and DREAD models, supported by diagrams, spreadsheets and free tooling. Increasingly, mature organizations tweak preexisting threat modeling frameworks and adopt commercial ASRTM tooling to support and automate threat-modeling activities. The topics of secure design and threat modeling are covered further in the research "A Guidance Framework for Establishing and Maturing an Application Security Program" and "5 Ways EA Can Help the Organization Focus on Security."

Secure Coding Practices

There are numerous security requirements, standards and specifications that help define the types of controls you should put in place for a given application or system. However, they often provide little technical guidance on how to go about accomplishing this. Your secure coding practices must provide code-specific guidance to address security challenges that arise in application development. They should recommend trusted functions, such as native secure functions, trusted third-party libraries or externalized security mechanisms, over custom code.

Secure coding practices and technical guidance come from a variety of sources, including the Software Engineering Institute (SEI) Cert Division, the Open Web Application Security Project (OWASP), and each respective vendor for a given technology. Others can be adapted from AST tools such as SAST. In some cases, particularly with modern languages or frameworks, specific resources may be sparse. You must account for the potentially numerous languages and technology stacks in use at your organization. Seek to standardize as much as possible to ease secure DevOps efforts overall.

Such standardization is very much a matter of software development life cycle development maturity, specifically formalization of the technology that development teams use and the associated development processes. Once you have made those decisions, you can draft secure coding practices using the variety of technical reference materials available. Ideally, this is a collaborative effort among teams to account for language and technology stack complexities. Additionally, these practices promote application security awareness among security, development and operations teams. Creating secure coding practices is a manual process similar to translating security requirements from document form into nonfunctional or operational requirements for application development life cycle management. However, information on secure coding is less linked to the SDLC ecosystem, and integration with developer IDEs and ADLM is not well-supported today. Integrating secure coding guidance into development workflow requires some creativity in getting contextualized guidance to developers where it is most essential, such as storing and linking information within ADLM and possibly augmenting further with ASRTM tools.
This topic is covered further in the research “A Guidance Framework for Establishing and Maturing an Application Security Program.” Figure 9 shows the layers of translation that must occur for compliance mandates, security requirements and secure coding practices.

**Figure 9: Translation of Security Requirements Into Secure Coding Practices**

**Translation of Security Requirements Into Secure Coding Practices**

<table>
<thead>
<tr>
<th>Compliance Focus</th>
<th>Risk Assessment Focus</th>
<th>Development Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Incidents</td>
<td>• Security Domain</td>
<td>• Language</td>
</tr>
<tr>
<td>• Ethical Norms</td>
<td>• Specifics</td>
<td>• Framework</td>
</tr>
<tr>
<td>• Legal Precedent</td>
<td>• Enterprise</td>
<td>• Specifics</td>
</tr>
<tr>
<td></td>
<td>• Specifics</td>
<td>• Platform</td>
</tr>
<tr>
<td></td>
<td>• Industry Best</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Practice</td>
<td></td>
</tr>
</tbody>
</table>

Security Coaches and Champions

Because dedicated application security resources will always be at a premium, organizations must seek to scale by designating security champions or coaches within respective application teams to partner and collaborate with. This concept shares similarities to security-focused communities of practice within the Scaled Agile Framework (SAFe), or to security guilds within the Spotify Model. “Security champions” may also refer to general security awareness efforts, such as ransomware or phishing. To avoid confusion with that role, emphasize the application security focus, and align with agile methodologies. Organizations may opt to use the label “security coaches.”

The reality for most organizations is that an application security team — if the organization even has one — can never be large enough to support all the applications and application development in the enterprise. A decentralized approach with delegation of some responsibilities is almost always required to carry out all activities and to ensure application security work is completed in a timely manner.

As individuals demonstrate application security knowledge from focused security training initiatives and advanced-level classes, delegate security tasks and responsibilities to distribute the workload and augment dedicated application security staff. Training is often gamified, involving interactive labs and cyber ranges, but it may also still include traditional computer-based training (CBT). Programs that leverage security coaches usually include some level of advancement,
reward or public recognition as individuals become more adept with application security. Because this role carries increased responsibility and workload, you need to make it interesting and rewarding. Figure 10 presents an example of what progression might look like for a developer in a security coach role, based loosely on Brazilian jiu-jitsu belt progression.

Figure 10: Example Security Coach Program for Developers

### Example Security Coach Program for Developers

<table>
<thead>
<tr>
<th>Level</th>
<th>Guideline Training Prerequisites</th>
<th>Advancement</th>
<th>Application Security Task to Maintain Level</th>
</tr>
</thead>
</table>
| ![Belt] 1/4 | • Completed 10 training courses  
• Completed basic security coding training, including OWASP Top 10 | • Compete in 3 CTFs, with an average score of 500  
• Complete 10 code sprints with no high-severity vulnerabilities | • Assist in gathering security testing requirements for applications relevant to their domain |
| ![Belt] 1/2 | • Completed 15 training courses  
• Received advanced training on specific vulnerability types, security architecture and/or language specifics | • Compete in 5 CTFs, with an average score of 1,000  
• Complete 25 code sprints with no high-severity vulnerabilities | • Provide feedback on secure coding practices during annual review  
• Assist with AST remediation activity for applications relevant to their domain |
| ![Belt] 3/4 | • Completed 20 training courses | • Compete in 5 CTFs and place in top 10  
• Complete 50 code sprints with no high-severity vulnerabilities | • Assist with drafting secure coding practices  
• Perform AST independently for applications relevant to their domain |
| ![Belt] Full | • Completed 25 training courses  
• Achieved industry certification or presented on topics related to secure coding or design | N/A | • Author or oversee secure coding practices in their domain  
• Generate threat models for applications relevant to their domain  
• Assist in bug triaging for reported incidents |

Source: Gartner  
729065_C

The topics of security coach programs and application security training gamification are covered further in the research “A Guidance Framework for Establishing and Maturing an Application Security Program.”

**DevSecOps**

DevOps practices combine application development, infrastructure and operations activities, with a heavy push toward automation to shorten delivery times while improving quality. A main goal of DevSecOps is to integrate security processes and tooling into engineering workflows and supporting technology stacks. Four patterns of application security concentration emerge as organizations embrace DevOps practices. All are critical to an overall application security program.
and DevSecOps approach. But organizations may put heavier emphasis on a particular area due to a variety of factors, including organizational politics or structure, budgets for staffing or tooling, or presence of preexisting tooling that may be repurposed. The focus areas are commonly one of the following:

1. **Secure design** — Heavy emphasis on process and “building security in.” Common elements include security requirements gathering and enforcement, threat modeling, secure coding practices, and use of trusted, externalized components.

2. **Development verification** — A focus on validating that secure coding practices have been followed, security requirements are satisfied, and the application is reasonably free of weaknesses or vulnerabilities. Tools like SCA and AST are used to verify.

3. **Externalized security** — A focus on securing applications in production or runtime, mostly outside of the codebase. A variety of technologies come into play, including WAFs, RASP, API gateways, bot mitigation and application shielding. This area also often includes workload-specific security mechanisms, such as ensuring hardened server builds via continuous configuration automation.

4. **Production security monitoring** — Continuous discovery and monitoring of applications and systems, within on-premises networks or cloud providers. This is also the realm of security operations centers (SOC), vulnerability assessment and vulnerability management. There may also be overlap with other tooling used to initially secure workloads and tenants, such as CSPM and CWPP.

Where you choose to initially focus your efforts impacts which tools you need to procure upfront, as well as what you can integrate or automate. Figure 11 maps application security practices and technologies across these four concentration areas to the Gartner DevSecOps model. This topic is covered further in the research “*Structuring Application Security Tools and Practices for DevOps and DevSecOps.*”

**Figure 11: Gartner DevSecOps Model**
Secure Continuous Delivery

Continuous delivery is supported by multiple integrated systems and processes. There is no single solution or product for securing the resulting technology and artifacts. Organizations will adopt a variety of development and delivery tools — both proprietary and open-source. Securing continuous delivery requires a diverse set of development, operations and security tooling and skill sets. Secure continuous delivery is also a discipline within the larger umbrella of DevSecOps. As such, Gartner observes that organizations with higher-maturity DevOps practices are often best-equipped to run with these security strategies. However, all organizations can benefit from adopting at least a subset of these practices. The tooling that enables continuous delivery practices also greatly enhances security through increased visibility, consistency and governance.

Following security best practices and ensuring good hygiene in continuous delivery are enablers to creating higher-quality or more-secure code. These practices are directly relevant to the abuse cases organizations are experiencing, such as loss of intellectual property or malicious manipulation of the digital supply chain. A formalized SDLC is also foundational within certain certifications, standards and frameworks. This area of focus is about securing the processes and
systems used to create, manage, build and deliver application code and the infrastructure it runs on. Secure approaches must account for the interconnectedness of IDEs, VCS, CI/CD, binary repositories, container registries and platforms such as Kubernetes to run workloads. Secure continuous delivery is also arguably a more advanced area of practice, as many organizations are still early in their adoption of DevOps or even basic security testing of custom application code.

Figure 12 highlights best practices for secure continuous delivery. This topic is also covered further in the research “Best Practices for Securing Continuous Delivery Systems and Artifacts.”

**Figure 12: Best Practices for Secure Continuous Delivery**

<table>
<thead>
<tr>
<th>Best Practices for Securing Continuous Delivery Systems and Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>After deploying developer tooling, version control, binary repositories and registries, and secrets management, DevOps teams should:</td>
</tr>
<tr>
<td><strong>Ensure Application Source Code Integrity</strong></td>
</tr>
<tr>
<td>• Enable VCS security features</td>
</tr>
<tr>
<td>• Secure registries, managers and repositories</td>
</tr>
<tr>
<td>• Audit source code for embedded secrets</td>
</tr>
<tr>
<td><strong>Secure Operating Environments</strong></td>
</tr>
<tr>
<td>• Hardened IaC baselines</td>
</tr>
<tr>
<td>• Use vetted images, and manage them centrally</td>
</tr>
<tr>
<td>• Audit images and IaC for embedded secrets</td>
</tr>
<tr>
<td><strong>Secure Application Build Pipeline</strong></td>
</tr>
<tr>
<td>• Enable CI/CD tooling security features</td>
</tr>
<tr>
<td>• Limit pipeline push/pull to trusted sources</td>
</tr>
<tr>
<td>• Verity asset compliance during instantiation</td>
</tr>
<tr>
<td><strong>Beware of Risks and Pitfalls</strong></td>
</tr>
<tr>
<td>• Neglecting to scan for embedded secrets</td>
</tr>
<tr>
<td>• Omitting a strategy for cloud platforms</td>
</tr>
<tr>
<td>• Overly restricting development environments</td>
</tr>
</tbody>
</table>

**Application Security Logical Architecture**

Now that we have discussed all the layers and components, you can construct a logical architecture to see how the processes and technology components of application security fit together. In addition, you must thoroughly test any application code that you build or source from suppliers to ensure that it is reasonably free from vulnerabilities and weaknesses. Such testing is imperative to guarantee a level of security and privacy.

Applications and APIs may live in multiple locations, including on-premises data centers, public cloud environments or private cloud segments. Applications and APIs may also be consumed or integrated purely as cloud services. Consumers of applications and APIs also take a variety of
forms, including customers, citizens, students, employees and business partners. Finally, newer application architecture patterns, cloud-native design and variance in the type of compute may necessitate new approaches to your application protection. Deploying application protection in the appropriate positions in your enterprise architecture — based on thorough consideration of edge, outer and inner architecture — is just as critical as vendor product selection. Figure 13 shows a logical representation of an application security architecture incorporating all of the possible processes and technology options.

Figure 13: Application Security Logical Architecture

Recommendations

Gartner recommends starting with an application security program or S-SDLC. If time and resources are at a premium, create a skeleton application security program that includes a combination of:
Application protection, likely using WAAP, to stem the broad range of app layer attacks against exposed production applications.

SCA to analyze third-party and open-source libraries for known issues. Scan application code, container images and IaC with SCA tooling to detect outdated, vulnerable libraries and license issues in third-party libraries.

A base level of security testing capabilities for the organization’s highest-risk (based on business criticality, data sensitivity, etc.) or most exposed applications. Select either SAST or DAST depending on your organization’s culture and resource availability. If you have the ability to inspect original source code, emphasize SAST early in code creation and as part of code commits. Alternatively, if original code is unavailable, consider DAST or IAST.

Ensure that you address both application security and container security as part of your strategy by including a combination of WAAP and workload protection. The latter may be available natively from your container platform or procured through a CWPP.

Flesh out other program elements over time, collaborating with security and nonsecurity stakeholders to ensure the program covers all the technology stacks in use in the organization. The program should also reflect the realities of the organization with identified owners, defined SLAs and defined deliverables.

As you mature in your application security program, integrate AST tooling as part of the development workflow, and seek to automate AST as part of application release pipelines. Doing so will:

- Position you to support DevOps initiatives
- Provide consistency with security testing
- Support change management process
- Help satisfy compliance requirements around software delivery

Analyze the architecture of the applications you aim to protect, and perform threat modeling to understand what attacks may become prevalent. Prefer the combined protections of WAAP platforms for most security use cases, though any solution must be configured and tuned appropriately for your unique applications and APIs.

Conclusion

Technology evolution, rapid development, heavy degrees of automation and integration complexity surround application security. As organizations move toward DevOps, significant intersections form between application development and I&O, particularly with regard to the workflow and
tooling used by engineers. And as organizations embrace cloud-native architecture, containers and cloud provider technologies enter the fray. There is a diverse set of supporting application security technologies and processes that organizations must account for in this modern IT era. This guide will continue to be updated as new acronyms and concepts arise in the application security space. This topic is a rapidly changing area for security and risk management technical professionals. Your strategy will need to embrace change to ensure your organization and the applications it builds are protected from the latest security threats. By following the recommendations outlined by Gartner, both here and in forthcoming research, you will be able to build a strong application security strategy and architecture that will help protect your organization.

Acronym Key and Glossary Terms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aPaaS</td>
<td>application platform as a service</td>
</tr>
<tr>
<td>API</td>
<td>application program interface</td>
</tr>
<tr>
<td>ASRTM</td>
<td>application security requirements and threat management</td>
</tr>
<tr>
<td>CaaS</td>
<td>containers as a service</td>
</tr>
<tr>
<td>CASB</td>
<td>cloud access security broker</td>
</tr>
<tr>
<td>CIS</td>
<td>Center for Information Security</td>
</tr>
<tr>
<td>CSP</td>
<td>cloud service provider</td>
</tr>
<tr>
<td>CSPM</td>
<td>cloud security posture management</td>
</tr>
<tr>
<td>CWPP</td>
<td>cloud workload protection platform</td>
</tr>
<tr>
<td>DAST</td>
<td>dynamic application security testing</td>
</tr>
<tr>
<td>DDoS</td>
<td>distributed denial of service</td>
</tr>
<tr>
<td>DoS</td>
<td>denial of service</td>
</tr>
<tr>
<td>fPaaS</td>
<td>function platform as a service</td>
</tr>
<tr>
<td>HTTP</td>
<td>hypertext transfer protocol</td>
</tr>
<tr>
<td>I&amp;O</td>
<td>infrastructure and operations</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>IaaS</td>
<td>infrastructure as a service</td>
</tr>
<tr>
<td>IAST</td>
<td>interactive application security testing</td>
</tr>
<tr>
<td>K8s</td>
<td>Kubernetes</td>
</tr>
<tr>
<td>SaaS</td>
<td>software as a service</td>
</tr>
<tr>
<td>SAST</td>
<td>static application security testing</td>
</tr>
<tr>
<td>SCA</td>
<td>software composition analysis</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>WAAP</td>
<td>web application and API protection</td>
</tr>
<tr>
<td>WAF</td>
<td>web application firewall</td>
</tr>
</tbody>
</table>

**Recommended by the Authors**

A Guidance Framework for Establishing and Maturing an Application Security Program
Best Practices for Securing Continuous Delivery Systems and Artifacts
Containers: 11 Threats and How to Control Them
Decision Point for Deploying WAFs for Application Protection
Decision Point for Securing Application Architecture
Guidance Framework for Securing Kubernetes
How to Deploy and Perform Application Security Testing
Protecting Web Applications and APIs From Exploits and Abuse
Solution Path for Forming an API Security Strategy
Solution Comparison for Cloud-Based Web Application Firewall Services
Structuring Application Security Tools and Practices for DevOps and DevSecOps

**Recommended For You**
Summary Translation: 2020 Retail Digital Transformation and Innovation Trends

Summary Translation: How to Build a Successful Business Case for Desktop Virtualization

Critical Capabilities for Sales Force Automation

Product Managers Must Rethink Their Product Roadmaps as 2 Different Semiconductor Technology Stacks Emerge

Summary Translation + Localization: Smarter Spending: An Overview

© 2020 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates. This publication may not be reproduced or distributed in any form without Gartner’s prior written permission. It consists of the opinions of Gartner’s research organization, which should not be construed as statements of fact. While the information contained in this publication has been obtained from sources believed to be reliable, Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Although Gartner research may address legal and financial issues, Gartner does not provide legal or investment advice and its research should not be construed or used as such. Your access and use of this publication are governed by Gartner’s Usage Policy. Gartner prides itself on its reputation for independence and objectivity. Its research is produced independently by its research organization without input or influence from any third party. For further information, see “Guiding Principles on Independence and Objectivity.”