DEVELOP SECURE APPLICATIONS ON AZURE

Disclaimer

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

This document is provided "as-is." Information and views expressed in this document, including URL and other internet website references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

NOTE: Certain recommendations in this white paper may result in increased data, network, or compute resource usage, and may increase your license or subscription costs.

Executive summary

This paper presents security activities and controls to consider when you develop applications for the cloud. The paper introduces the phases of the Microsoft Security Development Lifecycle (SDL) and security questions and concepts to consider during each phase of the lifecycle. The goal is to help you define activities and Azure services that you can use in each phase of the lifecycle to develop, test, and release a more secure application.

The recommendations in this paper come from our experience with Azure security and from the experiences of our customers. You can use this paper as a reference for what you should consider during a specific phase of your development project, but we suggest that you also read through this paper from beginning to end at least once. Reading the entire paper introduces you to concepts that you might have missed in earlier phases of your project. Implementing these concepts before you release your product can help you build secure software, address security compliance requirements, and reduce development costs.

This paper is intended to be a resource for software designers, developers, and testers at all levels who build and deploy secure Azure applications.

A document that describes how to develop secure applications on Azure will be released in the future. When the how-to guidance is available, a link to the guidance will be added to this document.

Table of contents

Overview

Security is one of the most important aspects of any application, and it's not a simple thing to get right. Fortunately, Azure provides many services that can help you secure your application in the cloud. This paper addresses activities and Azure services you can implement at each stage of your software development lifecycle to help you create more secure code and deploy a more secure application in the cloud.

Security Development Lifecycle

Following best practices for secure software development requires integrating security into each phase of the software development lifecycle, from requirement analysis to maintenance, regardless of the project methodology (waterfall, agile, or DevOps). In the wake of high-profile data breaches and the exploitation of operational security flaws, more developers are understanding that security needs to be addressed throughout the development process.

The later you fix a problem in your development lifecycle, the more that fix will cost you. Security issues are no exception. If you disregard security issues in the early phases of your software development, each phase that follows might inherit the vulnerabilities of the preceding phase. Your final product will have accumulated multiple security issues and the possibility of a breach. Building security into each phase of the development lifecycle helps you catch issues early, and it helps you reduce your development costs.

In this paper, we follow the phases of the Microsoft Security Development Lifecycle (SDL) to introduce activities and Azure services that you can use to fulfill secure software development practices in each phase of the lifecycle.

The SDL phases are:

- Training
- Requirements
- Design
- Implementation
- Verification
- Release
- Response

This paper is a general guide to the security questions and decisions you should consider at each phase of the SDL. We encourage you to return to this paper at each phase of your software development project for a reminder of the key issues you need to think about. Implementing these concepts before you release your product can help you build more secure software, address security compliance requirements, and reduce development costs.

Engage your organization's security team

Your organization might have a formal application security program that assists you with security activities from start to finish during the development lifecycle. If your organization has security and compliance teams, be sure to engage them before you begin developing your application. Ask them at each phase of the SDL whether there are any tasks you missed.

We understand that many readers might not have a security or compliance team to engage. This paper can help guide you in the security questions and decisions you need to consider at each phase of the SDL.

Training

Before you begin developing your cloud application, take time to understand security and privacy on Azure. By taking this step, you can reduce the number and severity of exploitable vulnerabilities in your application. You'll be more prepared to react appropriately to the ever-changing threat landscape.

Use the following resources during the training stage to familiarize yourself with the Azure services that are available to developers and with security best practices on Azure:

- Developer's guide to Azure shows you how to get started with Azure. The guide shows you which services you can use to run your applications, store your data, incorporate intelligence, build IoT apps, and deploy your solutions in a more efficient and secure way.
- Get started guide for Azure developers provides essential information for developers who are looking to get started using the Azure platform for their development needs.
- SDKs and tools describes the tools that are available on Azure.
- Azure DevOps Services provides development collaboration tools. The tools include high-performance pipelines, free Git repositories, configurable Kanban boards, and extensive automated and cloud-based load testing. The DevOps Resource Center combines our resources for learning DevOps practices, Git version control, agile methods, how we work with DevOps at Microsoft, and how you can assess your own DevOps progression.
- Top 5 security items to consider before pushing to production shows you how to help secure your web applications on Azure and protect your apps against the most common and dangerous web application attacks.
- Secure DevOps Kit for Azure is a collection of scripts, tools, extensions, and automations that caters to the comprehensive Azure subscription and resource security needs of DevOps teams that use extensive automation. The Secure DevOps Kit for Azure can show you how to smoothly integrate security into your native DevOps workflows. The kit addresses tools like security verification tests (SVTs), which can help developers write secure code and test the secure configuration of their cloud applications in the coding and early development stages.

- Security best practices for Azure solutions provides a collection of security best practices to use as you design, deploy, and manage your cloud solutions by using Azure.

Requirements

The requirements definition phase is a crucial step in defining what your application is and what it will do when it's released. The requirements phase is also a time to think about the security controls that you will build into your application. During this phase, you also begin the steps that you will take throughout the SDL to ensure that you release and deploy a secure application.

This phase is the best time to consider foundational security and privacy issues. Defining acceptable levels of security and privacy at the start of a project helps a team:

- Understand risks associated with security issues.
- Identify and fix security bugs during development.
- Apply established levels of security and privacy throughout the entire project.

When you write the requirements for your application, be sure to consider security controls that can help keep your application and data safe.

Ask security questions like:

- Does my application contain sensitive data or personally identifiable information (PII)?
- Does my application collect or store data that requires me to adhere to industry standards and compliance programs like the Federal Financial Institution Examination Council (FFIEC) or the Payment Card Industry Data Security Standards (PCI DSS)?
- Does my application collect or contain sensitive personal or customer data that can be used, either on its own or with other information, to identify, contact, or locate a single person?
- Does my application collect or contain data that can be used to access an individual's medical, educational, financial, or employment information? Identifying the sensitivity of your data during the requirements phase helps you classify your data and identify the data protection method you will use for your application.
- Where and how is my data stored? Consider how you will monitor the storage services that your application uses for any unexpected changes (such as slower response times). Will you be able to influence logging to collect more detailed data and analyze a problem in depth?
- Will my application be available to the public (on the internet) or internally only? If your application is available to the public, how do you protect the data that might be collected from being used in the wrong way? If your application is available internally only, consider who in your organization should have access to the application and how long they should have access.
- Do you understand your identity model before you begin designing your application? How will you determine that users are who they say they are and what a user is authorized to do?

- Does my application perform sensitive or important tasks (such as transferring money, unlocking doors, or delivering medicine)? Consider how you will validate that the user performing a sensitive task is authorized to perform the task and how you will authenticate that the person is who they say they are. Authorization (AuthZ) is the act of granting an authenticated security principal permission to do something. Authentication (AuthN) is the act of challenging a party for legitimate credentials.
- Does my application perform any risky software activities, like allowing users to upload or download files or other data? If your application does perform risky activities, consider how your application will protect users from handling malicious files or data.

Consider reviewing the OWASP Top 10 Application Security Risks. The OWASP Top 10 addresses critical security risks to web applications. Awareness of these security risks can help you make requirement and design decisions that minimize these risks in your application.

Thinking about security controls to prevent breaches is important. However, you also want to assume a breach will occur. Assuming a breach helps answer some important questions about security in advance, so they don't have to be answered in an emergency:

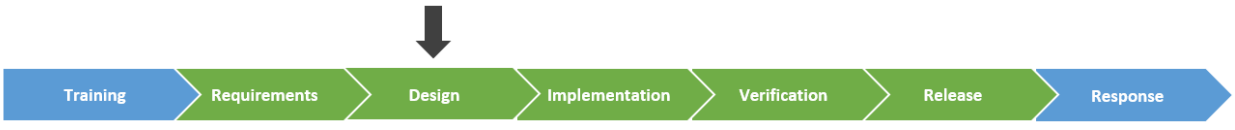
- How will I detect an attack?
- What will I do if there is an attack or breach?
- How am I going to recover from the attack like data leaking or tampering?

In the following sections of this paper, we recommend security controls and activities that can help you develop and release secure applications. The security controls that you choose to implement in your code and the activities that you choose to perform during the project should be requirements that are documented during the requirements definition phase. This helps prevent the control or activity from being overlooked in a later phase of your project.

You're on your way to creating a more secure application when developers know from the beginning the security requirements that they need to adhere to.

Design

The design phase is critical for establishing best practices for design and functional specifications. It also is critical for performing risk analysis that helps mitigate security and privacy issues throughout a project.



When you have security requirements in place and use secure design concepts, you can avoid or minimize opportunities for a security flaw. A security flaw is an oversight in the design of the application that might allow a user to perform malicious or unexpected actions after your application is released.

During the design phase, also think about how you can apply security in layers; one level of defense isn't necessarily enough. What happens if an attacker gets past your web application firewall (WAF)? You want another security control in place to defend against that attack.

With this in mind, we discuss the following secure design concepts and the security controls you should address when you design secure applications:

- Use a secure coding library and a software framework.
- Scan for vulnerable components.
- Use threat modeling during application design.
- Reduce your attack surface.
- Adopt a policy of identity as the primary security perimeter.
- Require re-authentication for important transactions.
- Use a key management solution to secure keys, credentials, and other secrets.
- Protect sensitive data.
- Implement fail-safe measures.
- Take advantage of error and exception handling.
- Use logging and alerting.

Use a secure coding library and a software framework

For development, use a secure coding library and a software framework that has embedded security. Developers can use existing, proven features (encryption, input sanitation, output encoding, keys or connection strings, and anything else that would be considered a security control) instead of developing security controls from scratch. This helps guard against security-related design and implementation flaws.

Be sure that you're using the latest version of your framework and all the security features that are available in the framework. Microsoft offers a comprehensive set of development tools for all developers, working on any platform or language, to deliver cloud applications. You can code with the language of your choice by choosing from various SDKs. You can take advantage of full-featured integrated development environments (IDEs) and editors that have advanced debugging capabilities and built-in Azure support.

Microsoft offers a variety of languages, frameworks, and tools that you can use to develop applications on Azure. An example is Azure for .NET and .NET Core developers. For each language and framework that we offer, you'll find quickstarts, tutorials, and API references to help you get started fast.

Azure offers a variety of services you can use to host websites and web applications. These services let you develop in your favorite language, whether that's .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Azure App Service Web Apps (Web Apps) is one of these services.

Web Apps adds the power of Microsoft Azure to your application. It includes security, load balancing, autoscaling, and automated management. You can also take advantage of DevOps capabilities in Web Apps, like package management, staging environments, custom domains, SSL/TLS certificates, and continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources.

Azure offers other services that you can use to host websites and web applications. For most scenarios, Web Apps is the best choice. For a micro service architecture, consider Azure Service Fabric. If you need more control over the VMs that your code runs on, consider Azure Virtual Machines. For more information about how to choose between these Azure services, see a comparison of Azure App Service, Virtual Machines, Service Fabric, and Cloud Services.

Apply updates to components

To prevent vulnerabilities, you should continuously inventory both your client-side and server-side components (for example, frameworks and libraries) and their dependencies for updates. New vulnerabilities and updated software versions are released continuously. Ensure that you have an ongoing plan to monitor, triage, and apply updates or configuration changes to the libraries and components you use.

See the Open Web Application Security Project (OWASP) page on using components with known vulnerabilities for tool suggestions. You can also subscribe to email alerts for security vulnerabilities that are related to components you use.

Use threat modeling during application design

Threat modeling is the process of identifying potential security threats to your business and application, and then ensuring that proper mitigations are in place. The SDL specifies that teams should engage in threat modeling during the design phase, when resolving potential issues is relatively easy and cost-effective. Using threat modeling in the design phase can greatly reduce your total cost of development.

To help facilitate the threat modeling process, we designed the SDL Threat Modeling Tool with non-security experts in mind. This tool makes threat modeling easier for all developers by providing clear guidance about how to create and analyze threat models.

Modeling the application design and enumerating STRIDE threats—Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege—across all trust boundaries has proven an effective way to catch design errors early on. The following table lists the STRIDE threats and gives some example mitigations that use features provided by Azure. These mitigations won't work in every situation.

Threat	**Security property**	**Potential Azure platform mitigation**
Spoofing	Authentication	Require HTTPS connections.
Tampering	Integrity	Validate SSL/TLS certificates. Applications that use SSL/TLS

		must fully verify the X.509 certificates of the entities they connect to. Use Azure Key Vault certificates to manage your x509 certificates.
Repudiation	Non-repudiation	Enable Azure monitoring and diagnostics.
Information Disclosure	Confidentiality	Encrypt sensitive data at rest and in transit.
Denial of Service	Availability	Monitor performance metrics for potential denial of service conditions. Implement connection filters. Azure DDoS protection, combined with application design best practices, provides defense against DDoS attacks.
Elevation of Privilege	Authorization	Use Azure Active Directory Privileged Identity Management.

Reduce your attack surface

An attack surface is the total sum of where potential vulnerabilities might occur. In this paper, we focus on an application's attack surface. The focus is on protecting an application from attack. A simple and quick way to minimize your attack surface is to remove unused resources and code from your application. The smaller your application, the smaller your attack surface. For example, remove:

- Code for features you haven't released yet.
- Debugging support code.
- Network interfaces and protocols that aren't used or which have been deprecated.

- Virtual machines and other resources that you aren't using.

Doing regular cleanup of your resources and ensuring that you remove unused code are great ways to ensure that there are fewer opportunities for malicious actors to attack.

A more detailed and in-depth way to reduce your attack surface is to complete an attack surface analysis. An attack surface analysis helps you map the parts of a system that need to be reviewed and tested for security vulnerabilities.

The purpose of an attack surface analysis is to understand the risk areas in an application so developers and security specialists are aware of what parts of the application are open to attack. Then, you can find ways to minimize this potential, track when and how the attack surface changes, and what this means from a risk perspective.

An attack surface analysis helps you identify:

- Functions and parts of the system you need to review and test for security vulnerabilities.
- High-risk areas of code that require defense-in-depth protection (parts of the system that you need to defend).
- When you alter the attack surface and need to refresh a threat assessment.

Reducing opportunities for attackers to exploit a potential weak spot or vulnerability requires you to thoroughly analyze your application's overall attack surface. It also includes disabling or restricting access to system services, applying the principle of least privilege, and employing layered defenses wherever possible.

We discuss conducting an attack surface review during the verification phase of the SDL.

Note: What's the difference between threat modeling and attack surface analysis?

Threat modeling is the process of identifying potential security threats to your application and ensuring that proper mitigations against the threats are in place. Attack surface analysis identifies high-risk areas of code that are open to attack. It involves finding ways to defend high-risk areas of your application and reviewing and testing those areas of code before you deploy the application.

Adopt a policy of identity as the primary security perimeter

When you design cloud applications, it's important to expand your security perimeter focus from a network-centric approach to an identity-centric approach. Historically, the primary on-premises security perimeter was an organization's network. Most on-premises security designs use the network as the primary security pivot. For cloud applications, you are better served by considering identity as the primary security perimeter.

Things you can do to develop an identity-centric approach to developing web applications:

- Enforce multi-factor authentication for users.
- Use strong authentication and authorization platforms.
- Apply the principle of least privilege.
- Implement just-in-time access.

Enforce multi-factor authentication for users

Use two-factor authentication. Two-factor authentication is the current standard for authentication and authorization because it avoids the security weaknesses that are inherent in username and password types of authentication. Access to the Azure management interfaces (Azure portal/remote PowerShell) and to customer-facing services should be designed and configured to use Azure Multi-Factor Authentication.

Use strong authentication and authorization platforms

Use platform-supplied authentication and authorization mechanisms instead of custom code. This is because developing custom authentication code can be prone to error. Commercial code (for example, from Microsoft) often is extensively reviewed for security. Azure Active Directory (Azure AD) is the Azure solution for identity and access management. These Azure AD tools and services help with secure development:

- Azure AD identity platform (Azure AD for developers) is a cloud identity service that developers use to build apps that securely sign in users. Azure AD assists developers who are building single-tenant, line-of-business (LOB) apps and developers who are looking to develop multi-tenant apps. In addition to basic sign-in, apps that are built by using Azure AD can call Microsoft APIs and custom APIs that are built on the Azure AD platform. The Azure AD identity platform supports industry-standard protocols like OAuth 2.0 and OpenID Connect.
- Azure Active Directory B2C (Azure AD B2C) is an identity management service you can use to customize and control how customers sign up, sign in, and manage their profiles when they use your applications. This includes applications that are developed for iOS, Android, and .NET, among others. Azure AD B2C enables these actions while protecting customer identities.

Apply the principle of least privilege

The concept of least privilege means giving users the precise level of access and control they need to do their jobs and nothing more.

Would a software developer need domain admin rights? Would an administrative assistant need access to administrative controls on their personal computer? Evaluating access to software is no different. If you use role-based access control (RBAC) to give users different abilities and authority in your application, you wouldn't give everyone access to everything. By limiting access to what is required for each role, you limit the risk of a security issue occurring.

Ensure that your application enforces least privilege throughout its access patterns.

Note: The rules of least privilege need to apply to the software and to the people creating the software. Software developers can be a huge risk to IT security if they are given too much access. The consequences can be severe if a developer has malicious intent or is given too much access. We recommend that the rules of least privilege be applied to developers throughout the development lifecycle.

Implement just-in-time access

Implement *just-in-time* (JIT) access to further lower the exposure time of privileges. Use Azure AD Privileged Identity Management to:

- Give users the permissions they need only JIT.
- Assign roles for a shortened duration with confidence that the privileges are revoked automatically.

Require re-authentication for important transactions

Cross-site request forgery (also known as *XSRF* or *CSRF*) is an attack against web-hosted apps in which a malicious web app influences the interaction between a client browser and a web app that trusts that browser. Cross-site request forgery attacks are possible because web browsers send some types of authentication tokens automatically with every request to a website. This form of exploitation is also known as a *one-click attack* or *session riding* because the attack takes advantage of the user's previously authenticated session.

The best way to defend against this kind of attack is to ask the user for something that only the user can provide before every important transaction, such as a purchase, account deactivation, or a password change. You might ask the user to reenter their password, complete a captcha, or submit a secret token that only the user would have. The most common approach is the secret token.

Use a key management solution to secure keys, credentials, and other secrets

Losing keys and credentials is a common problem. The only thing worse than losing your keys and credentials is having an unauthorized party gain access to them. Attackers can take advantage of automated and manual techniques to find keys and secrets that are stored in code repositories like GitHub. Don't put keys and secrets in these public code repositories or on any other server.

Always put your keys, certificates, secrets, and connection strings in a key management solution. You can use a centralized solution in which keys and secrets are stored in hardware security modules (HSMs). Azure provides you with an HSM in the cloud with Azure Key Vault.

Key Vault is a *secret store*: it's a centralized cloud service for storing application secrets. Key Vault keeps your confidential data safe by keeping application secrets in a single, central location and providing secure access, permissions control, and access logging.

Secrets are stored in individual *vaults*. Each vault has its own configuration and security policies to control access. You get to your data through a REST API or through a client SDK that's available for most programming languages.

Important: Azure Key Vault is designed to store configuration secrets for server applications**.** It's not intended for storing data that belongs to app users. This is reflected in its performance characteristics, API, and cost model.

User data should be stored elsewhere, like in an Azure SQL Database instance that has Transparent Data Encryption (TDE) or in a storage account that uses Azure Storage Service Encryption. Secrets that are used by your application to access these data stores can be kept in Azure Key Vault.

Protect sensitive data

Protecting data is an essential part of your security strategy. Classifying your data and identifying your data protection needs helps you design your app with data security in mind. Classifying (categorizing) stored data by sensitivity and business impact helps developers determine the risks that are associated with data.

Label all applicable data as sensitive when you design your data formats. Ensure that the application treats the applicable data as sensitive. These practices can help you protect your sensitive data:

- Use encryption.
- Avoid hard-coding secrets like keys and passwords.
- Ensure that access controls and auditing are in place.

Use encryption

Protecting data should be an essential part of your security strategy. If your data is stored in a database or if it moves back and forth between locations, use encryption of data at rest (while in the database) and encryption of data in transit (on its way to and from the user, the database, an API, or service endpoint). We recommend that you always use SSL/TLS protocols to exchange data. Ensure that you use the latest version of TLS for encryption (currently, this is version 1.2).

Avoid hard-coding

Some things should never be hard-coded in your software. Some examples are hostnames or IP addresses, URLs, email addresses, usernames, passwords, storage account keys, and other cryptographic keys. Consider implementing requirements around what can or can't be hard-coded in your code, including in the comment sections of your code.

When you put comments in your code, ensure that you don't save any sensitive information. This includes your email address, passwords, connection strings, information about your application that would only be known by someone in your organization, and anything else that might give an attacker an advantage in attacking your application or organization.

Basically, assume that everything in your development project will be public knowledge when it is deployed. Avoid including sensitive data of any kind in the project.

Earlier, we discussed Azure Key Vault. You can use Key Vault to store secrets like keys and passwords instead of hard-coding them. When you use Key Vault in combination with managed identities for Azure resources, your Azure web app can access secret configuration values easily and securely without storing any secrets in your source control or configuration. To learn more, see Manage secrets in your server apps with Azure Key Vault.

Implement fail-safe measures

Your application must be able to handle errors that occur during execution in a consistent manner. The application should catch all errors and either fail safe or closed.

You should also ensure that errors are logged with sufficient user context to identify suspicious or malicious activity. Logs should be retained for a sufficient time to allow delayed forensic analysis. Logs should be in a format that can be easily consumed by a log management solution. Ensure that alerts for errors that are related to security are triggered. Insufficient logging and monitoring allows attackers to further attack systems and maintain persistence.

Take advantage of error and exception handling

Implementing correct error and exception handling is an important part of defensive coding. Error and exception handling are critical to making a system reliable and secure. Mistakes in error handling can lead to different kinds of security vulnerabilities, such as leaking information to attackers and helping attackers understand more about your platform and design.

Ensure that:

- You handle exceptions in a centralized manner to avoid duplicated try/catch blocks in the code.
- All unexpected behaviors are handled inside the application.
- Messages that are displayed to users don't leak critical data but do provide enough information to explain the issue.
- Exceptions are logged and that they provide enough information for forensics or incident response teams to investigate.

Azure Logic Apps provides a first-class experience for handling errors and exceptions that are caused by dependent systems. You can use Logic Apps to create workflows to automate tasks and processes that integrate apps, data, systems, and services across enterprises and organizations.

Use logging and alerting

Log your security issues for security investigations and trigger alerts about issues to ensure that people know about problems in a timely manner. Enable auditing and logging on all components. Audit logs should capture user context and identify all important events.

Check that you don't log any sensitive data that a user submits to your site. Examples of sensitive data include:

- User credentials
- Social Security numbers or other identifying information
- Credit card numbers or other financial information
- Health information
- Private keys or other data that can be used to decrypt encrypted information
- System or application information that can be used to more effectively attack the application

Ensure that the application monitors user management events such as successful and failed user logins, password resets, password changes, account lockout, and user registration. Logging for these events helps you detect and react to potentially suspicious behavior. It also allows you to gather operations data, like who is accessing the application.

Implementation

The focus of the implementation phase is to establish best practices for early prevention and to detect and remove security issues from the code. Assume that your application will be used in ways that you didn't intend it to be used. This helps you guard against accidental or intentional misuse of your application.

Perform code reviews

Before you check in code, conduct code reviews to increase overall code quality and reduce the risk of creating bugs. You can use Visual Studio to manage the code review process.

Perform static code analysis

Static code analysis (also known as *source code analysis*) is usually performed as part of a code review. Static code analysis commonly refers to running static code analysis tools to find potential vulnerabilities in non-running code by using techniques like taint checking and data flow analysis.

Azure Marketplace offers developer tools that perform static code analysis and assist with code reviews.

Validate and sanitize every input for your application

Treat all input as untrusted to protect your application from the most common web application vulnerabilities. Untrusted data is a vehicle for injection attacks. Input for your application includes parameters in the URL, input from the user, data from the database or from an API, and anything that is passed in that a user could potentially manipulate. An application should validate that data is

syntactically and semantically valid before the application uses the data in any way (including displaying it back to the user).

Validate input early in the data flow to ensure that only properly formed data enters the workflow. You don't want malformed data persisting in your database or triggering a malfunction in a downstream component.

Blacklisting and whitelisting are two general approaches to performing input syntax validation:

- Blacklisting attempts to check that a given user input doesn't contain "known to be malicious" content.
- Whitelisting attempts to check that a given user input matches a set of "known good" inputs. Character-based whitelisting is a form of whitelisting where an application checks that user input contains only "known good" characters or that input matches a known format. For example, this might involve checking that a username contains only alphanumeric characters or that it contains exactly two numbers.

Whitelisting is the preferred approach for building secure software. Blacklisting is prone to error because it's impossible to think of a complete list of potentially bad input.

Do this work on the server, not on the client side (or on the server and on the client side).

Verify your application's outputs

Any output that you present either visually or within a document should always be encoded and escaped. Escaping, also known as *output encoding*, is used to help ensure that untrusted data isn't a vehicle for an injection attack. Escaping, combined with data validation, provides layered defenses to increase security of the system as a whole.

Escaping makes sure that everything is displayed as *output*. Escaping also lets the interpreter know that the data isn't intended to be executed, and this prevents attacks from working. This is another common attack technique called *cross-site scripting* (XSS).

If you are using a web framework from a third party, you can verify your options for output encoding on websites by using the OWASP XSS prevention cheat sheet.

Use parameterized queries when you contact the database

Never create an inline database query "on the fly" in your code and send it directly to the database. Malicious code inserted into your application could potentially cause your database to be stolen, wiped, or modified. Your application could also be used to run malicious operating system commands on the operating system that hosts your database.

Instead, use parameterized queries or stored procedures. When you use parameterized queries, you can invoke the procedure from your code safely and pass it a string without worrying that it will be treated as part of the query statement.

Remove standard server headers

Headers like Server, X-Powered-By, and X-AspNet-Version reveal information about the server and underlying technologies. We recommend that you suppress these headers to avoid fingerprinting the application. See removing standard server headers on Azure websites.

Segregate your production data

Your production data, or "real" data, should not be used for development, testing, or any other purpose than what the business intended. A masked (anonymized) dataset should be used for all development and testing.

This means fewer people have access to your real data, which reduces your attack surface. It also means fewer employees see personal data, which eliminates a potential breach in confidentiality.

Implement a strong password policy

To defend against brute-force and dictionary-based guessing, you must implement a strong password policy to ensure that users create a complex password (for example, 12 characters minimum length and requiring alphanumeric and special characters).

You can use an identity framework to create and enforce password policies. Azure AD B2C helps you with password management by providing built-in policies, self-service password reset, and more.

To defend against attacks on default accounts, verify that all keys and passwords are replaceable and that they're generated or replaced after you install resources.

If the application must auto-generate passwords, ensure that the generated passwords are random and that they have high entropy.

Validate file uploads

If your application allows file uploads, consider precautions that you can take for this risky activity. The first step in many attacks is to get some malicious code into a system that is under attack. Using a file upload helps the attacker accomplish this. OWASP offers solutions for validating a file to ensure that the file you're uploading is safe.

Antimalware protection helps identify and remove viruses, spyware, and other malicious software. You can install Microsoft Antimalware or a Microsoft partner's endpoint protection solution (Trend Micro, Symantec, McAfee, Windows Defender, and System Center Endpoint Protection).

Microsoft Antimalware includes features like real-time protection, scheduled scanning, malware remediation, signature updates, engine updates, samples reporting, and exclusion event collection. You can integrate Microsoft Antimalware and partner solutions with Azure Security Center for ease of deployment and built-in detections (alerts and incidents).

Don't cache sensitive content

Don't cache sensitive content on the browser. Browsers can store information for caching and history. Cached files are stored in a folder like the Temporary Internet Files folder, in the case of Internet Explorer. When these pages are referred to again, the browser displays the pages from its cache. If sensitive information (address, credit card details, Social Security number, username) is displayed to the user, the information might be stored in the browser's cache and be retrievable by examining the browser's cache or by simply pressing the browser's **Back** button.

Verification

The verification phase involves a comprehensive effort to ensure that the code meets the security and privacy tenets that were established in the preceding phases.

Find and fix vulnerabilities in your application dependencies

You should scan your application and its dependent libraries to identify any known vulnerable components. Products that are available to perform this scan include OWASP Dependency Check, Snyk, and Black Duck.

Vulnerability scanning powered by Tinfoil Security is available for Azure App Service Web Apps. Tinfoil Security scanning through App Service offers developers and administrators a fast, integrated, and economical means of discovering and addressing vulnerabilities before a malicious actor can take advantage of them.

Note: You can also integrate Tinfoil Security with Azure AD. Integrating Tinfoil Security with Azure AD provides you with the following benefits:

- In Azure AD, you can control who has access to Tinfoil Security.
- Your users can be automatically signed in to Tinfoil Security (single sign-on) by using their Azure AD accounts.
- You can manage your accounts in a single, central location, the Azure portal.

Test your application in an operating state

Dynamic application security testing (DAST) is a process of testing an application in an operating state to find security vulnerabilities. DAST tools analyze programs while they are executing to find security vulnerabilities such as memory corruption, insecure server configuration, cross-site scripting, user privilege issues, SQL injection, and other critical security concerns.

DAST is different from static application security testing (SAST). SAST tools analyze source code or compiled versions of code when the code is not executing in order to find security flaws.

Perform DAST, preferably with the assistance of a security professional (a penetration tester or vulnerability assessor). If a security professional isn't available, you can do this yourself with a web proxy scanner and some training. Plug in a DAST scanner early on to ensure that you don't introduce obvious security issues into your code. See the OWASP site for a list of web application vulnerability scanners.

Perform fuzz testing

In fuzz testing, you induce program failure by deliberately introducing malformed or random data to an application. Inducing program failure helps reveal potential security issues before the application is released.

Security Risk Detection is the Microsoft unique fuzz testing service for finding security-critical bugs in software.

Conduct attack surface review

Reviewing the attack surface after code completion helps ensure that any design or implementation changes to an application or system have been considered. It helps ensure that any new attack vectors that were created as a result of the changes, including threat models, have been reviewed and mitigated.

You can build a picture of the attack surface by scanning the application. Microsoft offers an attack surface analysis tool called Attack Surface Analyzer. You can choose from many commercial dynamic testing and vulnerability scanning tools or services, including OWASP_Zed_Attack_Proxy_Project, Arachni, Skipfish, and w3af. These scanning tools crawl your app and map the parts of the application that are accessible over the web. You can also search the Azure Marketplace for similar developer tools.

Perform security penetration testing

Ensuring that your application is secure is as important as testing any other functionality. Make penetration testing a standard part of the build and deployment process. Schedule regular security tests and vulnerability scanning on deployed applications, and monitor for open ports, endpoints, and attacks.

Run security verification tests

Secure DevOps Kit for Azure (AzSK) contains SVTs for multiple services of the Azure platform. You should run these SVTs periodically to ensure that your Azure subscription and the different resources that comprise your application are in a secure state. You can also automate these tests by using the continuous integration/continuous deployment (CI/CD) extensions feature of AzSK, which makes SVTs available as a Visual Studio extension.

Release

The focus of the release phase is readying a project for public release. This includes planning ways to effectively perform post-release servicing tasks and address security vulnerabilities that might occur later.

Check your application's performance before you launch

Check your application's performance before you launch it or deploy updates to production. Run cloud-based load tests by using Visual Studio to find performance problems in your application, improve deployment quality, make sure that your application is always up or available, and that your application can handle traffic for your launch.

Install a web application firewall

Web applications are increasingly targets of malicious attacks that exploit common known vulnerabilities. Common among these exploits are SQL injection attacks and cross-site scripting attacks. Preventing these attacks in application code can be challenging. It might require rigorous maintenance, patching, and monitoring at many layers of the application topology. A centralized WAF helps make security management simpler. A WAF solution can also react to a security threat by patching a known vulnerability at a central location versus securing each individual web application.

The Azure Application Gateway WAF provides centralized protection of your web applications from common exploits and vulnerabilities. The WAF is based on rules from the OWASP core rule sets 3.0 or 2.2.9.

Create an incident response plan

Preparing an incident response plan is crucial to help you address new threats that might emerge over time. Preparing an incident response plan includes identifying appropriate security emergency contacts and establishing security servicing plans for code that's inherited from other groups in the organization and for licensed third-party code.

Conduct a final security review

Deliberately reviewing all security activities that were performed helps ensure readiness for your software release or application. The final security review (FSR) usually includes examining threat models, tools outputs, and performance against the quality gates and bug bars that were defined in the requirements phase.

Certify release and archive

Certifying software before a release helps ensure that security and privacy requirements are met. Archiving all pertinent data is essential for performing post-release servicing tasks. Archiving also helps lower the long-term costs associated with sustained software engineering.

Response

The response post-release phase centers on the development team being able and available to respond appropriately to any reports of emerging software threats and vulnerabilities.

Execute the incident response plan

Being able to implement the incident response plan instituted in the release phase is essential to helping protect customers from software security or privacy vulnerabilities that emerge.

Monitor application performance

Ongoing monitoring of your application after it's deployed potentially helps you detect performance issues as well as security vulnerabilities. Azure services that assist with application monitoring are:

- Azure Application Insights
- Azure Security Center

Application Insights

Application Insights is an extensible Application Performance Management (APM) service for web developers on multiple platforms. Use it to monitor your live web application. Application Insights automatically detects performance anomalies. It includes powerful analytics tools to help you diagnose issues and understand what users actually do with your app. It's designed to help you continuously improve performance and usability.

Azure Security Center

Azure Security Center helps you prevent, detect, and respond to threats with increased visibility into (and control over) the security of your Azure resources, including web applications. Azure Security Center helps detect threats that might otherwise go unnoticed. It works with various security solutions.

Security Center's Free tier offers limited security for your Azure resources only. The Security Center Standard tier extends these capabilities to on-premises resources and other clouds. Security Center Standard helps you:

- Find and fix security vulnerabilities.
- Apply access and application controls to block malicious activity.
- Detect threats by using analytics and intelligence.
- Respond quickly when under attack.

Resources

Use the following resources to learn more about developing secure applications and to help secure your applications on Azure:

Microsoft Security Development Lifecycle (SDL) – The SDL is a software development process from Microsoft that helps developers build more secure software. It helps you address security compliance requirements while reducing development costs.

Open Web Application Security Project (OWASP) – OWASP is an online community that produces freely available articles, methodologies, documentation, tools, and technologies in the field of web application security.

Pushing Left, Like a Boss – A series of online articles by Tanya Janca (Microsoft) that outlines different types of application security activities that developers should complete to create more secure code.

Microsoft identity platform – The Microsoft identity platform is an evolution of the Azure AD identity service and developer platform. It's a full-featured platform that consists of an authentication service, open-source libraries, application registration and configuration, full developer documentation, code samples, and other developer content. The Microsoft identity platform supports industry-standard protocols like OAuth 2.0 and OpenID Connect.

Security best practices for Azure solutions – A collection of security best practices to use when you design, deploy, and manage cloud solutions by using Azure. This paper is intended to be a resource for IT pros. This might include designers, architects, developers, and testers who build and deploy secure Azure solutions.

Security and Compliance Blueprints on Azure – Azure Security and Compliance Blueprints are resources that can help you build and launch cloud-powered applications that comply with stringent regulations and standards.

