



Introduction

2 minutes

Security is top-of-mind for most people working in technology, and this is especially true when it comes to software. It seems we are constantly hearing about companies and governments that we entrusted our private data to, being compromised. When this happens - either maliciously or accidentally, it costs them customers and, ultimately money.

Did you know that the most common cause of data breaches is poor security in software? It's true. This means that the pressure is on for software developers to be more diligent than ever. But where do they begin? This module is the start of your journey into the world of application security, with the top five defenses for web applications. Learn how to secure your web applications on Azure and protect your apps against the most common and dangerous web application attacks.

Learning objectives

In this module, you will:

- Use Azure Security Center
- Verify your application's inputs and outputs
- Store your secrets into Key Vault
- Ensure you are using the latest version of your framework, and its security features
- Validate that your program dependencies and libraries are safe to use

Next unit: Azure Security Center

Continue >



Azure Security Center

15 minutes

One of the biggest problems with security is being able to see all the areas you need to protect and to find vulnerabilities before hackers do. Azure provides a service which makes this much easier called Azure Security Center.

What is Azure Security Center?

Azure Security Center (ASC) is a monitoring service that provides threat protection across all of your services both in Azure, and on-premises. It can:

- Provide security recommendations based on your configurations, resources, and networks.
- Monitor security settings across on-premises and cloud workloads and automatically apply required security to new services as they come online.
- Continuously monitor all your services and perform automatic security assessments to identify potential vulnerabilities before they can be exploited.
- Use machine learning to detect and block malware from being installed in your services and virtual machines. You can also white-list applications to ensure that only the apps you validate are allowed to execute.
- Analyze and identify potential inbound attacks and help to investigate threats and any post-breach activity which might have occurred.
- Just-In-Time access control for ports, reducing your attack surface by ensuring the network only allows traffic you require.

ASC is part of the [Center for Internet Security](#) (CIS) recommendations.

Activating Azure Security Center

Azure Security Center provides unified security management and advanced threat protection for hybrid cloud workloads and is offered in two tiers: Free and Standard. The

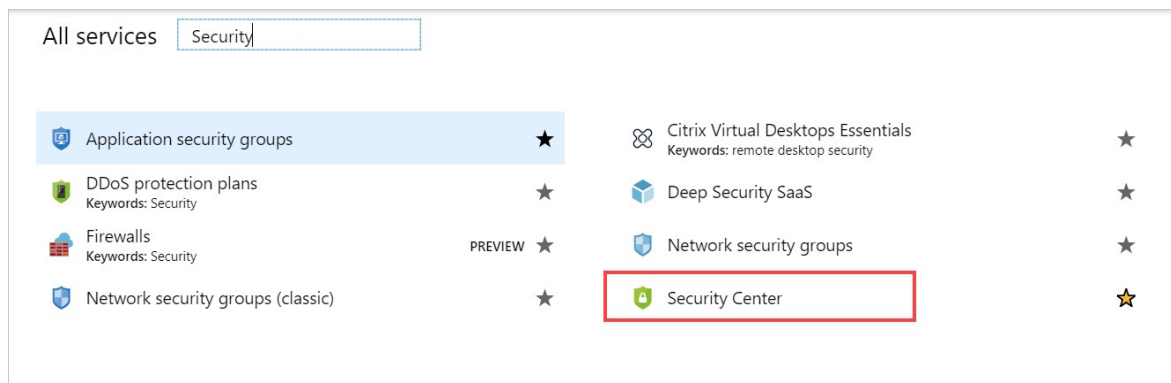
free tier provides security policies, assessments, and recommendations while the Standard tier provides a robust set of features, including threat intelligence.

Given the benefits of ASC, the security team at your company has decided that it be turned on for all subscriptions at your office. You got an email this morning to turn it on for your applications - so let's look at how to do that.

Important

Azure Security Center is not supported in the free Azure sandbox. You can perform these steps in your own subscription, or just follow along to understand how to activate ASC.

1. Open the [Azure portal](#) and select **Azure Security Center** from the left-hand menu, if you don't see it there, you can select **All services** and find **Security Center** in the security section as shown below.



2. If you have never opened ASC, the pane will start on the **Getting started** entry which might ask you to upgrade your subscription. Ignore that for now, select **Skip** at the bottom of the page, and then select **Overview**.
 - This will display the "big security picture" across all the elements available in your subscription.
 - This has a ton of great information you can explore.
3. Next, select **Coverage**, under "Policy and Compliance". This will display what subscription elements are being covered (or not covered) by ASC. Here you can


turn on ASC for any subscription you have access to. Try switching between the three coverage areas: "Not covered", "Basic coverage" and "Standard coverage".

4. Subscriptions that are not covered will have a prompt to activate ASC. You can press the "Upgrade Now" button to enable ASC for all the resources in the subscription.

[Not covered](#) [Basic coverage](#) [Standard coverage](#)

You have subscriptions that are not fully protected. Upgrade to the Standard tier for more recommendations. [Learn more](#) [Upgrade now!](#)

1 SUBSCRIPTIONS

NAME	MY ROLE	OWNER	RESOURCES	ID
	Owner	Display Owners	2	:

Free vs. Standard pricing tier

While you can use a free Azure subscription tier with ASC, it is limited to assessments and recommendations of Azure resources only. To really leverage ASC, you will need to upgrade to a Standard tier subscription as shown above. You can upgrade your subscription through the "Upgrade Now" button in the **Coverage** pane as noted above. You can also switch to the **Getting Started** pane in the ASC menu which will walk you through changing your subscription level. The pricing and features may change based on the region, you can get a full overview on the [pricing page](#).

ⓘ Note

To upgrade a subscription to the Standard tier, you must be assigned the role of Subscription Owner, Subscription Contributor, or Security Admin.

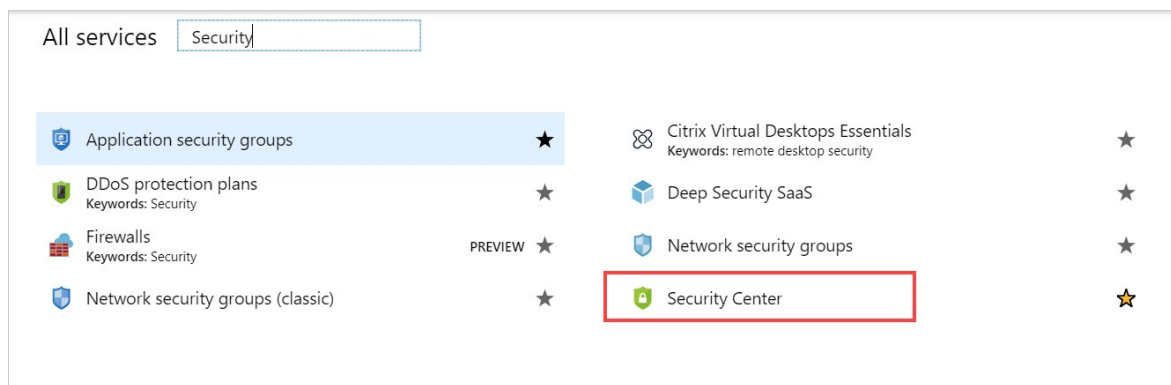
ⓘ Important

After the 60-day trial period is over, ASC is priced at **\$15/node per month** and will be billed to your account.

Turning off Azure Security Center

For production systems, you will definitely want to keep Azure Security Center turned on so it can monitor all your resources for threats. However, if you are just playing with ASC and turned it on, you will likely want to disable it to ensure you are not charged. Let's do that now.

1. Open the [Azure portal](#) and select **Azure Security Center** from the left-hand menu, if you don't see it there, you can select **All services** and find **Security Center** in the security section as shown below.



2. Select **Security Policy** from the left-hand menu.
3. Next, select **Edit settings** >, next to the subscription for which you want to downgrade ASC.
4. On the next screen select "Pricing Tier" from the left-hand menu.
5. A new page will appear that looks like the image below. Click on the box on the left that says "Free (for Azure resources only)".

Free (for Azure resources only)	Standard
✓ Security assessment	✓ Security assessment
✓ Security recommendations	✓ Security recommendations
✓ Basic security policy	✓ Basic security policy
✓ Connected partner solutions	✓ Connected partner solutions
✗ Just in time VM Access	✓ Just in time VM Access
✗ Adaptive application controls	✓ Adaptive application controls
✗ Network threat detection	✓ Network threat detection
✗ VM threat detection	✓ VM threat detection
0.00 USD/NODE/MONTH	15.00 USD/NODE/MONTH

6. Press the **Save** button at the top of the screen.

You have now downgraded your subscription to the free tier of Azure Security Center.

Congratulations, you have taken your first (and most important) step to securing your application, data and network!

Check your knowledge

1. True or false: Azure Security Center works for Azure resources and on-premises resources.

- ☐ True
- ☐ False

Check your answers

Next unit: Inputs and Outputs

[Continue >](#)



Inputs and Outputs


10 minutes

The most prevalent security weakness of current applications is a failure to correctly process data that is received from external sources, particularly *user input*. You should always take a close look at any input to make sure it has been validated before it is used. Failing to analyze user input for possible attacks can result in data loss or exposure, elevation of privilege, or even execution of malicious code on other users' computers.

The tragedy in this situation is that this scenario an easy problem to solve. In this unit we will cover how to treat data; when it's received, when it's displayed on the screen, and when it's stored for later use.

Why do we need to validate our input?

Imagine that you are building an interface to allow a user to create an account on your website. Our profile data includes a name, email, and a nickname that we will display to everyone who visits the site. What if a new user creates a profile and enters a nickname that includes some SQL commands? For example - what if a malicious user enters something like the following excerpt:

SQL	 Copy
<code>Eve'); DROP TABLE Users;--</code>	

If we blindly insert this value into a database, it could potentially alter the SQL statement to execute commands we absolutely don't want to run! This example is referred to as a "SQL Injection" attack, which is one of the *many* types of exploits that can potentially be done when you don't properly handle user input. So, what can we do to fix this situation? This unit will teach you when to validate input, how to encode output, and how to create parameterized queries (which solves the above exploit).

These techniques are the three main defense techniques against malicious input being entered into your applications.

When do I need to validate input?

The answer is *always*. You must validate **every** input for your application. This includes parameters in the URL, input from the user, data from the database, data from an API and anything that is passed in the clear that a user could potentially manipulate. Always use an *allow list* approach, which means you only accept "known good" input, instead of a *deny list* (where you specifically look for bad input) because it's impossible to think of a complete list of potentially dangerous input. Do this work on the server, not the client-side (or in addition to the client-side), to ensure that your defenses cannot be circumvented. Treat **ALL** data as untrusted and you will protect yourself from most of the common web app vulnerabilities.


If you are using ASP.NET, the framework provides [great support for validating input](#) on both the client and server side.

If you are using another web framework, there are some great techniques for doing input validation available on the [OWASP Input Validation Cheatsheet](#).


Always use parameterized queries

SQL databases are commonly used to store data; for example - your application could store user profile information in a database. You should never create inline SQL or other database queries in your code using raw user input and send it directly to the database; this behavior is a recipe for disaster, as we saw above.

For example - **do not** create code like the following inline SQL example:

C#	 Copy
<pre>string userName = Request.QueryString["username"]; // receive input from the user BEWARE! ... string query = "SELECT * FROM [dbo].[users] WHERE userName = '" + userName + "'";</pre>	

Here we concatenate text strings together to create the query, taking the input from the user and generating a dynamic SQL query to look up the user. Again, if a malicious user realized we were doing this, or just *tried* different input styles to see if there was a vulnerability, we could end up with a major disaster. Instead, use parameterized SQL statements or stored procedures such as this:

SQL	 Copy
<pre>-- Lookup a user CREATE PROCEDURE sp_findUser (@UserName varchar(50)) SELECT * FROM [dbo].[users] WHERE userName = @UserName</pre>	

With this method you can invoke the procedure from your code safely, passing it the `userName` string without worrying about it being treated as part of the SQL statement.

Always encode your output

Any output you present either visually or within a document should always be encoded and escaped. This can protect you in case something was missed in the sanitization pass, or the code accidentally generates something that can be used maliciously. This design principle will make sure that everything is displayed as *output* and not inadvertently interpreted as something that should be executed, which is another common attack technique that is referred to as "Cross-Site Scripting" (XSS).

Since XSS prevention is a common application requirement, this security technique is another area where ASP.NET will do the work for you. By default, all output is already encoded. If you are using another web framework, you can verify your options for output encoding on websites with the [OWASP XSS Prevention Cheatsheet](#).

Summary

Sanitizing and validating your input is a necessary requirement to ensure your input is valid and safe to use and store. Most modern web frameworks offer built-in features

that can automate some of this work. You can check your preferred framework's documentation and see what features it offers. While web applications are the most common place where this happens, keep in mind that other types of applications can be just as vulnerable. Don't think you're safe just because your new application is a desktop app. You will still need to properly handle user input to ensure someone doesn't use your app to corrupt your data, or damage your company's reputation.

Check your knowledge

1. Which of the following data sources need to be validated?

- ☐ Data from a 3rd party API
- ☐ Data from the URL parameter
- ☐ Data collected from the user via an input field

☒ All of the above

2. Parameterized queries (stored procedures in SQL) are a secure way to talk to the database because:

- ☐ They are more organized than inline database commands, and therefore less confusing for users.
- ☐ There is a clear outline of the script in the stored procedure, ensuring better visibility.

☒ Parameterized queries substitute variables before running queries, meaning it avoids the opportunity for code to be submitted in place of a variable.

3. Which of the following data needs to be output encoded?

- ☐ Data saved to the database
- ☒ Data to be output to the screen
- ☐ Data sent to a 3rd party API
- ☐ Data in the URL parameters

[Check your answers](#)

Next unit: Secrets in Key Vault

[Continue >](#)



Secrets in Key Vault

5 minutes

Secrets aren't secrets if they are shared with everyone. Storing confidential items like connection strings, security tokens, certificates and passwords in your code is just inviting someone to take them and use them for something other than what you intended them for. Even storing this sort of data in your web configuration is a bad idea - you are essentially allowing anyone who has access to the source code or web server access to your private data.

Instead, you should always put these secrets into **Azure Key Vault**.

What is Azure Key Vault

Azure Key Vault is a *secret store*: 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 with their own configuration and security policies to control access. You can then get to your data through a REST API, or through a client SDK available for most languages.

Important

Key Vault is designed to store configuration secrets for server applications. It's not intended for storing data belonging to your app's users, and it shouldn't be used in the client-side part of an app. This is reflected in its performance characteristics, API, and cost model.

User data should be stored elsewhere, such as in an Azure SQL database with Transparent Data Encryption, or a storage account with Storage Service Encryption.

Secrets used by your application to access those data stores can be kept in Key Vault.

Why use a Key Vault for my secrets

Key management and storing secrets can be complicated and error-prone when performed manually. Rotating certificates manually means potentially going without for a few hours, or days. As mentioned above, saving your connections strings in your configuration file or code repository means someone could steal your credentials.

Key Vault allows users to store connection strings, secrets, passwords, certificates, access policies, file locks (making items in Azure read-only), and automation scripts. It also logs access and activity, allows you to monitor access control (IAM) in your subscription, and it also has diagnostic, metrics, alerts and troubleshooting tools, to ensure you have the access you need.

Learn more about using an Azure Key Vault in [Manage secrets in your server apps with Azure Key Vault](#).

Summary

Credential theft, manual key rotation and certificate renewal can be a thing of the past if you manage your secrets well, using Azure Key Vault.

Next unit: Framework Updates

Continue >



Framework Updates

7 minutes

Many developers consider the frameworks and libraries they use to build their software with to be primarily decided by features or personal preference. However, the framework that you choose is an important decision, not only from a design and functionality perspective but also from a *security* perspective. Choosing a framework with modern security features and keeping it up-to-date is one of the best ways to ensure your apps are secure.

Choose your framework carefully

The most important factor regarding security when choosing a framework is how well supported it is. The best frameworks have stated security arrangements and are supported by large communities who improve and test the framework. No software is 100% bug-free or totally secure, but when a vulnerability is identified, we want to be certain that it will be closed or have a workaround provided quickly.

Often "well supported" is synonymous with "modern". Older frameworks tend to either be replaced or eventually fade in popularity. Even if you have significant experience with (or many apps written in) an older framework, you'll be better off choosing a modern library that has the features you need. Modern frameworks tend to build on the lessons learned by earlier iterations which makes choosing them for new apps a form of threat surface reduction. You will have one more app to worry about if a vulnerability is discovered in the older framework that your legacy applications are written in.

For more information on secure design and reducing threat surface, see [Design For Security in Azure](#).

Keep your framework updated

Software development frameworks, such as Java Spring and .NET Core release updates and new versions regularly. These updates include new features, removal of old features, and often security fixes or improvements. When we allow our frameworks to become out of date, it creates "technical debt". The further out of date we get, the harder and riskier it will be to bring our code up to the latest version. In addition, much like the initial framework choice, staying on older versions of the framework open you up to more security threats which have been fixed in newer releases of the framework.

As an example, from 2016-2017, [over 30 vulnerabilities](#) were found in the Apache Struts framework. These were quickly addressed by the development team, but some companies didn't apply the patches and [paid the price in the form of a data breach](#).

Make sure to keep your frameworks and libraries up-to-date.

How do I update my framework?

Some frameworks, like Java or .NET, require an install and tend to release on a known cadence. It's a good idea to watch for new releases and plan to make a branch of your code to try it out when it's released. As an example, .NET Core maintains a [release notes page](#) which you can check to find the latest versions available.

More specialized libraries such as JavaScript frameworks, or .NET components can be updated through a package manager. **NPM** and **Webpack** are popular choices for web projects and are supported by most IDEs or build tools. In .NET, we use **NuGet** to manage our component dependencies. Much like updating the core framework, branching your code, updating the components and testing is a good technique to validate a new version of a dependency.

⚠ Note

The `dotnet` command-line tool has an `add package` and `remove package` option to add or remove NuGet packages but doesn't have a corresponding `update package` command. However, it turns out you can run `dotnet add package <package-name>` in your project and it will automatically *upgrade* the package to the latest version. This is an easy way to update dependencies without having to open the IDE.

Take advantage of built-in security

Always check to see what security features your frameworks offer. **Never** roll your own security if there's a standard technique or capability built in. In addition, rely on proven algorithms and workflows because these have often been scrutinized by many experts, critiqued and strengthened so you can be assured that they are reliable and secure.

The .NET Core framework has countless security features, here are a few core starting places in the documentation.

- [Authentication -Identity Management](#)
- [Authorization](#)
- [Data Protection](#)
- [Secure Configuration](#)
- [Security Extensibility APIs](#)

Each one of these features was written by experts in their field, and then battered with tests to ensure that it works as intended, and only as intended. Other frameworks offer similar features - check with the vendor that provides the framework to find out what they have in each category.

Warning

Writing your own security controls, instead of using those provided by your framework, is not only wasting time, it's less secure.

Azure Security Center

When using Azure to host your web applications, Security Center will warn you if your frameworks are out of date as part of the recommendations tab. Don't forget to look there from time to time to see if there are any warnings related to your apps.

Use the latest supported PHP version for Web Application (Preview)

 20 web applications

Use the latest supported Node.js version for Web Application (Preview)

 3 web applications

Summary

Whenever possible, choose a modern framework to build your apps, always use the built-in security features, and make sure you keep it up-to-date. These simple rules will help to ensure your application starts on a solid foundation.

Next unit: Safe Dependencies

Continue >

Safe Dependencies

5 minutes

A large percentage of code present in modern applications are the libraries and dependencies chosen by you: the developer. This is a common practice that saves time and money. However, the downside is that you are now responsible for this code, even though others wrote it, because you used it in your project. If a researcher (or worse, a hacker) discovers a vulnerability in one of these 3rd party libraries, then the same flaw will likely also be present in your app.

Using components with known vulnerabilities is a huge problem in our industry. It is so problematic that it has made the [OWASP top ten list](#) of worst web application vulnerabilities, holding at #9 for several years.

Track known security vulnerabilities

The problem we have is knowing when an issue is discovered. Keeping our libraries and dependencies updated (#4 in our list!) will of course help, but it's a good idea to keep track of identified vulnerabilities that might impact your application.

Important

When a system has a known vulnerability, it is much more likely also to have exploits available, code that people can use to attack those systems. If an exploit is made public, it is crucial that any affected systems are updated immediately.

Mitre is a non-profit organization that maintains the [Common Vulnerabilities and Exposures list](#). This list is a publicly searchable set of known cybersecurity vulnerabilities in apps, libraries, and frameworks. **If you find a library or component in the CVE database, it has known vulnerabilities.**

Issues are submitted by the security community when a security flaw is found in a product or component. Each published issue is assigned an ID and contains the date discovered, a description of the vulnerability, and references to published workarounds or vendor statements about the issue.

How to verify if you have known vulnerabilities in your 3rd party components

You could put a daily task into your phone to go and check this list, but luckily for us, many tools exist to allow us to verify if our dependencies are vulnerable. You can run these tools against your codebase, or better yet, add them to your CI/CD pipeline to automatically check for issues as part of the development process.

- [OWASP Dependency Check](#), which has a [Jenkins plugin](#)
- [OWASP SonarQube](#)
- [Snyk](#), which is free for open source repositories in GitHub
- [Black Duck](#) which is used by many enterprises
- [RubySec](#) an advisory database just for Ruby
- [Retire.js](#) a tool for verifying if your JavaScript libraries are out of date; can be used as a plugin for various tools, including [Burp Suite](#)

Some tools made specifically for static code analysis can be used for this as well.

- [Roslyn Security Guard](#)
- [Puma Scan](#)
- [PT Application Inspector](#)
- [Apache Maven Dependency Plugin](#)
- [Source Clear](#)
- [Sonatype](#)
- [Node Security Platform](#)
- [WhiteSource](#)
- [Hdiv](#)
- [And many more...](#)

For more information on the risks involved in using vulnerable components visit the [OWASP page](#) dedicated to this topic.

Summary

When you use libraries or other 3rd party components as part of your application you are also taking on any risks they may have. The best way to reduce this risk is to ensure that you are only using components that have no known vulnerabilities associated with them.

Next unit: Conclusion

[Continue >](#)

Conclusion

1 minute

In this module, you learned how to safeguard your applications in Azure against common attacks. We looked at several key security tips:

- Use Azure Security Center
- Verify your application inputs and outputs
- Store your secrets into Key Vault
- Ensure you are using the latest version of your framework, and using its security features
- Verify your program dependencies and libraries are safe to use

By following the five items in this list, you will have taken leaps and strides towards ensuring that your applications are safe and that they remain that way.