OWASP Low-Code/No-Code Top 10

Security Risks for Low-Code/No-Code Development Platforms

Generated: 2025-09-05 01:27:52

Source: https://owasp.org/www-project-top-10-low-code-no-code-security-risks/

Overview

Low-Code/No-Code development platforms provide a development environment used to create application software through a graphical user interface instead of traditional hand-coded computer programming. Such platforms reduce the amount of traditional hand-coding, enabling accelerated delivery of business applications. As Low-Code/No-Code platforms proliferate and become widely used by organizations, there is a clear and immediate need to create awareness around security and privacy risks related to applications developed on such platforms. The primary goal of the "OWASP Low-Code/No-Code Top 10" document is to provide assistance and education for organizations looking to adopt and develop Low-Code/No-Code applications. The guide provides information about what the most prominent security risks are for such applications, the challenges involved, and how to overcome them.

Security Focus Areas

- Account impersonation and privilege escalation
- Authorization bypass and privilege misuse
- Data leakage and unintended consequences
- Authentication failures and communication security
- Security misconfigurations
- Injection handling failures
- Vulnerable and untrusted components
- Data and secret handling failures
- Asset management failures
- Security logging and monitoring failures

The OWASP Low-Code/No-Code Top 10

1. LCNC-SEC-01: Account Impersonation
2. LCNC-SEC-02: Authorization Misuse
3. LCNC-SEC-03: Data Leakage and Unexpected Consequences
4. LCNC-SEC-04: Authentication and Secure Communication Failures
5. LCNC-SEC-05: Security Misconfiguration
6. LCNC-SEC-06: Injection Handling Failures
7. LCNC-SEC-07: Vulnerable and Untrusted Components
8. LCNC-SEC-08: Data and Secret Handling Failures
9. LCNC-SEC-09: Asset Management Failures
10. LCNC-SEC-10: Security Logging and Monitoring Failures

LCNC-SEC-01: Account Impersonation

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-01: Account Impersonation

Low-code/no-code applications can be embedded with a developer account which is used implicitly by any application user.

This creates a direct path towards Privilege Escalation, allowing an attacker to hide behind another user's identity, circumventing traditional security controls.

Business User Description

A critical component of any system is tracking what user is taking actions in that system. When account impersonation occurs it "looks" like actions taken by one user are being done by another.

Identities are embedded within each built application, and multiple users can use that application.

This creates a direct path for application users to escalate privileges, which is atypical and should be avoided whenever possible.

Low-code/no-code applications can take advantage of embedded user accounts rather than having their own application identity.

Embedded identities can belong to the application creator, or they could be a common identity shared by teams, such as database credentials.

They could also be service accounts or shared identities.

The lack of application identity hides the application's existence from monitoring systems outside of the low-code/no-code platform.

As an outside viewer, any user that uses the application is impersonating the application's creator, and there is no way to distinguish between the application and its creator.

The problem becomes even more acute when applications use different identities to operate on various platforms.

In such a case, one user could be used to store files on a file-sharing SaaS and another user to retrieve on-premise data.

Example Attack Scenarios

A developer creates a simple application to view records from a database.

They use their identity to log into the database, creating a connection embedded within the application.

Every action that any user performs in this application ends up querying the database using the developer's identity.

A malicious user takes advantage of this and uses the application to view, modify or delete records they should not have access to.

Database logs indicate that all queries were made by a single user, the trusted developer.

A developer creates a business application that allows employees to fill out forms with their personal information.

To store form responses, the developer uses their personal email account.

Users have no way of knowing that the app is storing their data on the developer's personal account.

A developer creates a business application and shares it with an administrator.

The developer configures the app to use its user's identity.

Aside from its stated purpose, the app also uses its user's identity to elevate the privileges of the developer.

Once the admin uses the app, they inadvertently elevate the developer's privileges.

Example Attack & Misuse Scenarios - Business Users

A developer builds a No Code/Low Code Robotic Process Automation (RPA) application that connects to a database to update records. The connection uses the Admin's authentication (username and password) to log updates. Although 10 different users use this RPA process, all actions are being recorded as being done by the Admin. Logging systems can no longer track productivity, attribute errors to specific users, or identify malicious behavior.

A developer builds an application to help the sales team in the field. The developer uses their credentials (username and password) when writing the application, so all sales made through the application are attributed to the developer, not the sales person facilitating the sale.

Adhere to the principle of least privilege when provisioning connections to databases/services/SaaS

Ensure applications use dedicated service or application accounts rather than user accounts

Ensure applications use a single consistent identity across all their connections, rather than a different identity for each.

Use a dedicated service or application account for those connections

Ensure a proper audit trail is maintained to identify the actor behind actions performed through the shared connection, whether those connections are shared by virtue of users using the application or by granting users access to that connection directly

Low Code High Risk - Enterprise Domination via Low Code Abuse, DEF CON 2022

Watch Out for User Impersonation in Low-Code/No-Code Apps

Do low-code / no-code platforms pose a security risk?

Credential Sharing as a Service: The Hidden Risk of Low-Code/No-Code

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter

LCNC-SEC-02: Authorization Misuse

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-02: Authorization Misuse

Connections are first-class objects in most low-code/no-code platforms.

This means connections are made between applications, other users, and/or entire organizations and that if an application is deleted, the connection still remains ‘on’.

Applications can also be shared with users who should not have access to the underlying data.

In addition, the application may have a broader authorization scope than the use case demands.

Business User Description

When writing an application, it’s critical to verify the identity of a user (Authentication), and define the level of access privileges a user has when using that application (Authorization). Authorization misuse occurs when the application incorrectly defines what a user can do in a system.

Low-code/no-code development does not happen in siloes.

The impact is based on integrations across the organization’s stack through on-premise, SaaS, PaaS, and cloud platforms.

Most low-code/no-code platforms come built-in with a large set of connectors, i.e., wrappers around APIs, which allow quick and easy connectivity.

Connectors and user credentials form connections, which are predominantly first-class objects in most low-code/no-codeplatforms.

This means that connections can be shared among applications, with other users, or with entire organizations.

Many low-code/no-code platforms abuse OAuth authorization flows by querying and storing user refresh tokens, and re-using them at will to increase productivity and reduce time-to-deliver.

This allows business users to quickly set up connections without thinking about secrets or permissions; at the same time, connections are embedded with user identities that are difficult to monitor or revoke.

Even though OAuth refresh tokens are designed to be short-lived, they are most frequently valid for a few months or even years.

Hence, a connection created by a business user in under a minute could persist in the low-code/no-code platform for an extended period and often get used by other users for different purposes than the original intention.

An authorization scope controls the access to resources and assets of an organization.

Low-code/no-code developers prefer broad authorization scope for their applications to make them as generic as possible.

For organizations, this allows quick and easy setup of applications, which they can later use for other use cases without needing another application.

The broad authorization scope does come at the cost of unnecessary risk of authorization misuse.

Example Attack Scenarios

A developer creates a connection to their corporate email account.

They inadvertently click the “share with everyone” option, granting either usage permissions or full ownership.

Every user in the organization, including contractors and vendors, gains access to their corporate email account.

A malicious user triggers a “forgot password” flow and uses the connection to follow through with the process and gain control over the account.

A developer creates a simple application to view records from a database.

The application is configured to ensure each user can only view related records.

However, the application is configured in such a way that the underlying database connection is implicitly shared with its user.

An application user can use the database connection directly, gaining full access to all records.

Admin connects an application to their source code management system (e.g., Bitbucket) using a service or application account.

The provisioned service or application account has unrestricted access to all repositories to enable seamless integration.

Any internal user can abuse this connection to access restricted repositories they usually don’t have access to.

A developer creates a simple application to submit forms from one platform to another.

The application, however, is configured to require authorization to edit and delete form submissions when just creating form submissions should have been enough.

Example Attack & Misuse Scenarios - Business Users

A developer is asked to automate the entry of expense items into a finance system, which automates the process of entry and approval. Finance processes should have privilege separation between the user entering the transaction and the user approving the actions. Because this automation was built with incorrect Authorization, any user running this process can perform both tasks.

In an insurance company, a developer is asked to build an application to simplify the intake process of a new Consumer Auto insurance policy. The application is a big success, and users in the Commercial Auto line start using the application.

The Commercial Auto process has additional validations that were not planned for in the build of this application, causing major pricing errors later in the process.

The developer did not anticipate other departmental usage, and therefore never designed the application with appropriate Authorization settings.

A Business Intelligence (BI) report is created to analyze the experience and salaries of employees across departments. This report is intended only for senior management but it has been set with a wide authorization scope, allowing anybody in any department to view detailed information about employees.

A senior manager shares it with their team, who now can see all salaries across the company, including their salary compared to each other.

Disable or monitor the use of implicitly shared connections

Adhere to the principle of least privilege when providing access to environments that can contain shared connections

Monitor no-code/low-code platforms for over-shared connections

Educate business users on the risks of connection sharing and its relation to credential sharing

Explicitly refresh OAuth tokens on a regular basis by re-authenticating connections.

Carefully review the scope an application requires and adhere to the principle of least privilege

Credential Sharing as a Service: The Hidden Risk of Low-Code/No-Code

Low-Code Platforms Are the New Holy Grail for Hackers

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

LCNC-SEC-03: Data Leakage and Unexpected Consequences

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-03: Data Leakage and Unexpected Consequences

Low-code/no-code development legitimately accesses data from underlying services but can also serve as a conduit to those backend systems for actions that were not anticipated or approved of.

This includes unintended negative side effects such as data leakage beyond the application/security boundary, triggering create/read/update/delete operations on the data, and accidental/malicious data exfiltration.

Business User Description

Data leakage occurs when data moves outside of the organization’s control, and ends up in locations not intended or appropriate, which can lead to unexpected consequences.

Low-code/no-code development often accesses data or performs operations on underlying services.

As data operators, low-code/no-code applications can easily cause data leakage by moving data outside its designated storage or even the organizational boundary.

In some cases, low-code/no-code development can be used to sync data between multiple systems or trigger operations on one system due to a change in another.

As operation triggers, these resources can result in unexpected consequences by implicitly coupling an operation within one system with a change in another.

Furthermore, multiple applications can be connected to and triggered by a single data source, resulting in chained data movement or operation triggers, which are difficult to predict or fully map.

Example Attack Scenarios

A developer grants an application that was built using low-code/no-code access to a corporate database.

The application is shared with other users, granting them implicit access to the database without going through an approval or access request process.

A developer configures an automation that triggers each time they receive an email in their corporate mailbox.

The automation sends a new email to the developer’s personal email account, copying the recipients, subject, and body from the original email received in the corporate mailbox.

Since data is copied to a separate mailbox rather than emails being forwarded from the corporate mailbox, the automation bypasses DLP controls.

A developer builds an automation that syncs changes between two SharePoint sites, so every new file on site A is copied to site B.

A business user accidentally writes a sensitive document to site A, not knowing that it is replicated to site B.

Even if the business user deletes the document from site A, the document is still available on site B.

Example Attack & Misuse Scenarios - Business Users

A developer builds a process to copy emails with specific words to their personal email account. The content and its attachments, are all now outside of corporate control creating a risk of that data being compromised or being used inappropriately.

A user writes a process to automatically store data from a SharePoint site to a network drive. After loading a file to SharePoint, a user realizes the content is sensitive and deletes it from the

SharePoint. They don’t realize the file has been copied to the network drive, and as a result, that sensitive data remains available.

Limit connectors to an approved services list

Limit creation of custom connectors to dedicated personnel

Monitor platforms for data flow outside the organizational boundary, including multi-hop paths

3 Ways No-Code Developers Can Shoot Themselves in the Foot

Low-Code Security and Business Email Compromise via Email Auto-Forwarding

Hackers Abuse Low-Code Platforms And Turn Them Against Their Owners

Full Operational Shutdown—another cybercrime case from the Microsoft Detection and Response Team

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-04: Authentication and Secure Communication Failures

Low-code/no-code development typically interacts with business-critical data via connections set up by business users, which can often result in insecure communications.

Business User Description

Many applications need to access data and/or move data to fulfill their design. When the connection to access or move the data is created, “improper” settings can lead to the data being intercepted, blocked, or users having over-privileged access to data, services and more.

Low-code/no-code development typically connects to business-critical data across on-premise, Saas, PaaS, and cloud platforms.

Applications use built-in connectors, which allow for easy connections to various services.

Connections offer a variety of security configurations, including communication protocols, authentication flows, and types of credentials used.

In many cases, business users set up connections that often result in deviations from best practices and corporate data security policies.

Example Attack Scenarios

A developer creates an application that uses an FTP connection but doesn’t check the box that turns on encryption.

Users of that app have no way to know that their data is being transferred unencrypted since the communication between the app and its users is encrypted.

A developer uses administrator credentials to create a database connection.

They build an application that uses that connection to show data to its users. Even though they intended to allow read-only operations through the app, users can use the over-privileged connection to write or delete records from the database.

Example Attack & Misuse Scenarios - Business Users

An application is created to accept payments from users through a website, which sends credit card details to a credit card processor for processing. When sending the data to the credit card processor, the data is not encrypted.

A malicious user is intercepting the web traffic, and since the data is not encrypted, they can read all of the data, including the credit card information.

In production environments, limit the creation of connections to dedicated personnel

Monitor platforms for connections that do not comply with best practices

Educate business users on the risks of insecure communication and the need to involve security teams when setting them up

The CISO View: Protecting Privileged Access in RPA

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter

LCNC-SEC-05: Security Misconfiguration

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-05: Security Misconfiguration

Misconfigurations can often result in security risks like anonymous access to sensitive data or operations, unprotected public endpoints and secrets, and oversharing.

Business User Description

In building a solution, platforms provide a variety of features, each having an impact on security. They can range from how/where sensitive data is stored, encryption settings, and more. Users need to consider productivity vs risk when making the right choices to combine these features.

Low-code/no-code development platforms provide a wide range of features, some of which control the balance between security and support of specific use cases.

However, misconfigurations can often result in anonymous user access to sensitive data or operations, unprotected public endpoints, secrets, and oversharing.

Furthermore, many configurations are set on the application level rather than the tenant level, meaning they can be set by business users themselves, rather than administrators.

Example Attack Scenarios

A developer creates an application that exposes an API endpoint and fails to configure that endpoint to deny anonymous access.

Attackers scan the low-code/no-code platform’s subdomains, locate the app, and steal its underlying data.

A developer creates an automation that is triggered by a webhook, but fails to protect that webhook with a secret.

Attackers identify the webhook and can now trigger the automation at will.

The automation could be used to modify or delete data.

Example Attack & Misuse Scenarios - Business Users

A new application is built that requires a password to be set by a user. The “rules” configured for this new application are less secure than the company’s password policy. The less secure password makes it easier for a malicious user to guess the users password and access the system.

An application is created that allows users to upload documents to a server. The upload process has not been configured to scan the document upon upload for security risks. A malicious user uses this process to upload a document containing malware that then locks all computers in the company unless a payment is made (ransomware).

Read low-code/no-code platform vendor documentation and follow best practices guides

Ensure configurations align with industry best practices

Monitor configuration for drifts

Implement a change management system for tenant-level configuration

By Design: How Default Permissions on Microsoft Power Apps Exposed Millions

Microsoft Internal Security Best Practices: Secure Power Platform Development

Power Platform Landing Zones Reference Implementation

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

LCNC-SEC-06: Injection Handling Failures

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-06: Injection Handling Failures

Resources built using low-code/no-code development platforms ingest user-provided data in multiple ways, including direct input or retrieving user-provided content from various services.

Such data can contain malicious payloads that may introduce risk to the application, automation, connection, workflow, bot, or integration.

Business User Description

When user's input data into a system, that data can be interpreted as "code to run" rather than "data to be stored or processed".

Low-code/no-code applications ingest user-provided data in multiple ways, including direct input or retrieving user-provided content from various services.

Since applications often query data dynamically based on user input, they are exposed to a significant risk of injection-based attacks.

Moreover, since applications can use user-provided content in various ways, including querying a database, parsing a document, and so forth, protection from injection-based attacks must consider the specific application and its use of user data.

Example Attack Scenarios

A developer sets up an automation that triggers every time a new publication is made to an RSS feed and stores it into a SQL database.

An attacker controlling the feed uses this flow to inject commands into the database that delete important records.

A developer creates an application that allows users to fill out forms.

The app encodes form data as CSV files and stores them on a shared drive.

Even though the platform sanitizes form inputs for SQL injection attacks, security teams need to think about and label all the different things that are inputting data into, which is likely to result in something being missed, like an Office macro attack.

An attacker takes advantage of this and inputs a macro that gets written into the CSV file.

A user opens the CSV file to analyze user forms, and the macro gets executed.

Example Attack & Misuse Scenarios - Business Users

A user builds an online feedback form for an ecommerce store. A malicious user fills out the form using their knowledge of database commands that updates the prices of an item from $1,000 to $1. The hacker then buys the $1,000 item for $1, costing the company $999.

A user builds a registration form for new users. A malicious user uses their knowledge of database commands to craft a specific input into the user field, which deletes all users from the database. Users with pending orders try to check on the status of their orders, to be told their account doesn't exist.

Sanitize user input, taking into account the operations that will be performed on that input by the application

For database interaction via SQL, also use query parameterization, stored procedures, or escaping

Educate business users on the risk of unsanitized user input. Platforms cannot make this problem go away on their own

A03:2021 – Injection, OWASP Top 10

Robot Framework / RPA Framework SQL injection example & prevention

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter

LCNC-SEC-07: Vulnerable and Untrusted Components

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-07: Vulnerable and Untrusted Components

Low-code/no-code development relies heavily on ready-made components out of marketplaces, the web, or custom connectors built by developers.

These components are often unmanaged, lack visibility, and expose organizations to supply chain-based risks.

Business User Description

To accelerate building a solution, many LCNC platforms have the equivalent of an “app store” where a developer can use pre-made parts of a solution created by third parties. These pre-made components can come with any of the risks below, as well as being created with malicious intent.

In many cases, entire applications are built by vendors leveraging pre-built components, data connectors, widgets, and sub-services.

Third-party components and applications are often a target for attackers who wish to compromise a large number of customers.

Moreover, low-code/no-code development often enables extensibility through custom code.

These pieces of code are embedded into the application and, in some cases, are not held up to the same level of security vigilance as with other profesionally developed applications.

Example Attack Scenarios

Developers across an organization use a vulnerable component from the marketplace.

Every app that uses the component is exposed to exploitation. Admins can find it difficult to locate apps affected by the vulnerable component.

A developer creates a custom connector that allows developers to connect to an internal business API.

The custom connector passes the authentication token on the URL, exposing the authentication secrets to app users.

Example Attack & Misuse Scenarios - Business Users

A user is building an application that connects to their human resources systems to update salary information. The user finds a third party created “connector” to add to their HR system that greatly simplifies their task. The connector was created without properly securing the transmission (LCNC-SEC-04-Authentication-and-Secure-Communication-Failures), allowing a malicious user to view the data in-transit. The malicious user is able to use this to obtain and publicly publish the salaries of all employees.

A user builds a tablet based application to allow sales to be processed at a conference. They find a component that processes credit card data and decides to add it to their solution. The component was maliciously designed to send the credit card data to its creator. As a result, all of the conference users who purchased products had their credit card data stolen.

A new application has been created using a third party component for login. The third party component comes with default credentials of admin/admin, which is publicly documented. The developer is unaware of this default setting and does not change it. A malicious user is able to access the application using these default settings, leading to unauthorized access to the application.

Remove unused dependencies, unnecessary features, components, and files

Continuously inventory versions of applications and components used by those applications and scan that inventory for deprecated or vulnerable components

Limit use to pre-approved marketplace components

Monitor for components that are unmaintained or do not create security patches for older versions

No-Code Malware - Windows 11 at Your Service, DEF CON 2022

CVE-2021-44228: Incident Report for Mendix Technology B.V.

A06:2021 – Vulnerable and Outdated Components, OWASP Top 10

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter

LCNC-SEC-08: Data and Secret Handling Failures

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-08: Data and Secret Handling Failures

Low-code/no-code development often stores data or secrets as part of their “code,” or on managed databases offered by the platform, which must be stored adequately in compliance with regulation and security requirements.

Business User Description

Most applications store or use data, and some of that data is more sensitive than others. If a developer is unaware of what data is considered “sensitive data”, or the processes to protect that data, the sensitive information could be exposed.

This can include data such as username/password, credentials to access other systems, personal identifiable information (PII) and more based upon the specific business.

Data stored in databases managed by low-code/no-code vendors often include sensitive data, including PII and financial data.

Low-code/no-code builders can decide for themselves how to store data, and administrators often lack visibility into such managed databases.

In many cases, sensitive data is stored unencrypted and moved between geo-locations without considering regulatory requirements.

Furthermore, builders have many opportunities to hard-code secrets into their “code”.

Whether it’s through environment variables, configuration, or code, applications can often rely on hard-coded secrets to access other services.

Hard-coded secrets are available to all users with write permissions to the applications and might also leak to application readers or anonymous users via client-side code.

Moreover, many native log streams mix application logs, metrics, and sensitive data being passed through the application.

In many platforms, logs will contain actual data points that the application uses by default.

Example Attack Scenarios

A developer creates a business application that asks users to fill out a form with sensitive data.

They use the managed database provided by the platform to store results.

Since the managed database is stored with every other developer by default, they all gain access to the sensitive data.

A developer creates an application using a custom API and hard-codes the API key in the code.

Other developers can access the API key directly.

Moreover, the API key might leak to the app’s client code allowing users to gain direct access to the key.

Example Attack & Misuse Scenarios - Business Users

An application is developed, which stores credentials in a database. The database stores passwords in a human readable format, allowing anybody to view them. An employee with access to this database can view the password of every employee within the company, including highly privileged administrative accounts.

A new application is built that stores personal data about EU residents. The developer is unaware of the GDPR rules that allows a user to have their personal data deleted upon request. Rather than building a process to delete data upon request, the business now has to process these requests manually, resulting in inconsistent adherence to the GDPR rules. That inconsistent behavior can

lead to fines, audit findings and other potential penalties.

Educate business users on the compliance, privacy, and security risks related to data storage

Monitor managed databases, environment variables, and configuration provided by no-code/low-code vendors for sensitive data

Ensure security teams are involved with applications having access to sensitive data

3 Ways No-Code Developers Can Shoot Themselves in the Foot

Establishing a DLP strategy

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter

LCNC-SEC-09: Asset Management Failures

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-09: Asset Management Failures

The purpose of low-code/no-code development is that it makes it easy for anyone to create applications with relatively low maintenance costs.

This also unfortunately makes them prone to abandonment while the apps still remain active.

Furthermore, internal applications can gain popularity rapidly without addressing business continuity concerns as they scale in usage.

Business User Description

Keeping a clear inventory of applications in an organization is critical to maintaining a healthy ecosystem. Since LCNC applications can be developed rapidly and with ease, keeping track of what applications exist, how to maintain them, and who “owns” them becomes challenging.

The ease of creating new applications, their relatively low maintenance costs, and the fact that they are often managed by SaaS services are all contributing factors.

This means that the number of active applications in an organization tends to grow rapidly and that popular business applications can often find themselves without an owner.

Furthermore, the viral nature of useful internal business applications within large organizations can result in applications developed by a single business user and being relied on by many users throughout the organization.

Low-code/no-code development can lack business continuity measures expected from important business applications, for example, having multiple owners, being monitored by IT, or offering an SLA.

Example Attack Scenarios

A developer creates an application that becomes popular as an internal tool.

The developer leaves the organization or moves to another role.

Due to an API change, the application breaks, disrupting business.

IT is not aware of the app and cannot help fix it and maintain productivity.

A developer browses through the platform marketplace and explores application templates.

Each click creates an app with an external-facing URL.

The user forgets about these apps, even though they might expose business data.

This scenario is multiplied by the number of developers, resulting in an ever-growing number of stale apps.

Example Attack & Misuse Scenarios - Business Users

A developer builds an automated process to load data from a network location into the financial system to complete order processing. A manager is alerted the process isn’t running. The developer is now working for another company and has left no documentation. What should have been a quick fix takes weeks, and results in many thousands of overtime to manually process the transactions.

The security team notices that an account is being used to store multiple records simultaneously by a single user. This pattern indicates one of the vulnerabilities (LCNC-SEC-01-Account-Impersonation) may have occurred. The security team wants to identify the application and implement a change to record the individual user’s actions to their credentials. Due to poor inventory management, IT is unable to identify what application is causing this and therefore unable to enforce an appropriate change.

The Security team is planning a Vulnerability test of all applications. Due to poor inventory management, several LCNC developed applications are missed during this test. These applications have vulnerabilities that later led to a data breach. This breach should have been prevented if the security team were able to test the application.

Maintain a comprehensive inventory that lists applications, components, and users

Remove or disable unused dependencies, unnecessary features, components, files, and documentation

3 Ways No-Code Developers Can Shoot Themselves in the Foot

Why So Many Security Experts Are Concerned About Low-Code/No-Code Apps

You Can’t Opt Out of Citizen Development

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter

LCNC-SEC-10: Security Logging and Monitoring Failures

For full functionality of this site it is necessary to enable JavaScript. Here are the instructions how to enable JavaScript in your web browser.

This website uses cookies to analyze our traffic and only share that information with our analytics partners.Accept

LCNC-SEC-10: Security Logging and Monitoring Failures

Low-code/no-code applications often lack a comprehensive audit trail, produce insufficient or non-existent logs, or overshare access to sensitive logs.

Business User Description

When an application runs, it's critical to know what ran and the actions it or a user took (also known as an audit trail). This is often done using logging of specific actions. In case of a breach or malicious activity, the audit log is key to understanding what happened and how to recover. This risk occurs when running systems log too much or too little information, which needs to be balanced with cost and quality of information.

Low-code/no-code applications often rely on vendors to generate logs and monitoring data.

In many cases, logs are either insufficient or not being collected, which impedes security investigations and can lead to a failed audit stemming from compliance requirements.

Furthermore, individual resources often lack a comprehensive audit trail, preventing change management processes and inquiries.

Finding out who introduced a change becomes an intractable challenge.

Example Attack Scenarios

Application logs are turned off.

When a breach attempt occurs, security teams are unable to determine who accessed the app and what they tried to do.

A business-critical application stops functioning following a change.

Since multiple changes have occurred, each resulting in an application update, it is challenging to find which developer introduced the particular change that caused the issue.

Developers have to review each application version manually to locate the problematic version.

Since each application "save" translates to an update, the number of updates would make a manual process prohibitively expensive and time-consuming.

On some platforms, developers can only review the application's current version, so they won't be able to find or revert to a stable version.

Example Attack & Misuse Scenarios - Business Users

A developer builds an automated process to load data into a financial system to complete order processing. The process handles around one thousand transactions per week with a 97% success rate. The 3% that fail are processed manually off of a daily Failed Transaction email.

The Failed Transaction email fails to get sent for a few days before its absence is noticed. Due to poor logging there is no easy way to find the failed records. Instead 100% of all transactions have to be investigated to find the failures, resulting in a significantly larger task.

An application to process credit card payments at a conference is created. As part of its creation, a detailed log file is created to track the transactions and stored on a shared network drive. The logging includes records of the credit card details. A user browsing the network drive discovers this file and is able to obtain all of the credit card data.

Leverage platform built-in capabilities to collect user access and platform audit logs

Where applicable, instrument applications with logging mechanisms to provide extra visibility

Ensure logs are not contaminated with sensitive data by configuring the platform to avoid logging raw application data

No-Code Malware - Windows 11 at Your Service, DEF CON 2022

Full Operational Shutdown—another cybercrime case from the Microsoft Detection and Response Team

A08:2021 – Software and Data Integrity Failures, OWASP Top 10

The OWASP® Foundation works to improve the security of software through its community-led open source software projects,

hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Top 10 Low-Code/No-Code Security Risks

LCNC-SEC-01: Account Impersonation

LCNC-SEC-02: Authorization Misuse

LCNC-SEC-03: Data Leakage and Unexpected Consequences

LCNC-SEC-04: Authentication and Secure Communication Failures

LCNC-SEC-05: Security Misconfiguration

LCNC-SEC-06: Injection Handling Failures

LCNC-SEC-07: Vulnerable and Untrusted Components

LCNC-SEC-08: Data and Secret Handling Failures

LCNC-SEC-09: Asset Management Failures

LCNC-SEC-10: Security Logging and Monitoring Failures

Upcoming OWASP Global Events

Become a corporate supporter