Security | Langchain

Skip to main content LangChainDocsUse casesIntegrationsAPIMoreCommunityTutorialsContributingAlso by LangChainChat our docsLangSmithLangChain HubLangServePython DocsSearchCTRLKGet startedIntroductionInstallationQuickstartLangChain Expression LanguageInterfaceHow toCookbookWhy use LCEL?LangChain Expression Language (LCEL)ModulesModel I/ORetrievalChainsMemoryAgentsCallbacksModulesSecurityGuidesEcosystemSecurityOn this pageSecurityLangChain has a large ecosystem of integrations with various external resources like local and remote file systems, APIs and databases. These integrations allow developers to create versatile applications that combine the power of LLMs with the ability to access, interact with and manipulate external resources.Best PracticesWhen building such applications developers should remember to follow good security practices:Limit Permissions: Scope permissions specifically to the application's need. Granting broad or excessive permissions can introduce significant security vulnerabilities. To avoid such vulnerabilities, consider using read-only credentials, disallowing access to sensitive resources, using sandboxing techniques (such as running inside a container),

etc. as appropriate for your application.Anticipate Potential Misuse: Just as humans can err, so can Large Language Models (LLMs). Always assume that any system access or credentials may be used in any way allowed by the permissions they are assigned. For example, if a pair of database credentials allows deleting data, it’s safest to assume that any LLM able to use those credentials may in fact delete data.Defense in Depth: No security technique is perfect. Fine-tuning and good chain design can reduce, but not eliminate, the odds that a Large Language Model (LLM) may make a mistake. It’s best to combine multiple layered security approaches rather than relying on any single layer of defense to ensure security. For example: use both read-only permissions and sandboxing to ensure that LLMs are only able to access data that is explicitly meant for them to use.Risks of not doing so include, but are not limited to:Data corruption or loss.Unauthorized access to confidential information.Compromised performance or availability of critical resources.Example scenarios with mitigation strategies:A user may ask an agent with access to the file system to delete files that should not be deleted or read the content of files that contain sensitive information. To mitigate, limit the agent to only use a specific directory and only allow it to read or write files that are safe to read or write. Consider further sandboxing the agent by running it in a container.A user may ask an agent with write access to an external API to write malicious data to the API, or delete data from that API. To mitigate, give the agent read-only API keys, or limit it to only use endpoints that are already resistant to such misuse.A user may ask an agent with access to a database to drop a table or mutate the schema. To mitigate, scope the credentials to only the tables that the agent needs to access and consider issuing READ-ONLY credentials.If you're building applications that access external resources like file systems, APIs
or databases, consider speaking with your company's security team to determine how to best design and secure your applications.Reporting a VulnerabilityPlease report security vulnerabilities by email to security@langchain.dev. This will ensure the issue is promptly triaged and acted upon as needed.Enterprise solutionsLangChain offers enterprise solutions for customers who have

additional security requirements. Please contact us at sales@langchain.dev.

PreviousModulesNextGuidesBest PracticesReporting a VulnerabilityEnterprise solutionsCommunityDiscordTwitterGitHubPythonJS/TSMoreHomepageBlog