**Mlflow Security Improvements Documentation**

**Introduction**

This document details the security enhancements implemented to strengthen the robustness of the MLflow project. We have addressed two critical vulnerabilities:

1. **Code Injection** in mlflow/projects/backend/local.py
2. **Directory Traversal** in mlflow/store/artifact/local\_artifact\_repo.py

By introducing input validation and sanitization mechanisms, we have mitigated these vulnerabilities, ensuring the integrity and security of the system.

**Overview of Changes**

**1. Code Injection Vulnerability Fix**

**Problem Addressed**

A **Code Injection** vulnerability was identified in the \_run\_entry\_point function within mlflow/projects/backend/local.py. The function passed user-supplied commands directly to subprocess.Popen without proper validation. This allowed attackers to inject arbitrary shell commands through crafted inputs, potentially compromising the system.

**Necessity of Improvements**

To prevent unauthorized code execution and enhance security, it was crucial to validate and sanitize the command strings before execution. Implementing input validation ensures that only safe and intended commands are executed, mitigating the risk of code injection attacks.

**2. Directory Traversal Vulnerability Fix**

**Problem Addressed**

A **Directory Traversal** vulnerability was found in mlflow/store/artifact/local\_artifact\_repo.py. Improper sanitization of user-supplied paths in functions like delete\_artifacts, \_download\_file, and download\_artifacts allowed attackers to manipulate input paths using traversal sequences (e.g., ../, %2e%2e%2f). This could enable unauthorized access to or deletion of arbitrary files and directories on the server.

**Necessity of Improvements**

To protect the system from unauthorized file access and deletion, strict input validation was necessary. By ensuring all user-supplied paths are sanitized and confined within the intended directory structure, we prevent exploitation of the directory traversal vulnerability.

**Implementation Details**

**1. Code Injection Vulnerability Fix**

**File Modified**

* **File Path:** mlflow/projects/backend/local.py

**New Function Added**

* **Function Name:** \_validate\_and\_sanitize\_command(command)

**Function Updated**

* **Function Name:** \_run\_entry\_point(command, work\_dir, experiment\_id, run\_id)

**Location of Changes**

1. **Added the New Function**
   * Placed immediately before the \_run\_entry\_point function.
   * Encapsulated within specific comment markers for clarity.
2. **Modified the Existing Function**
   * Inserted a call to \_validate\_and\_sanitize\_command(command) at the beginning of \_run\_entry\_point.
   * Wrapped the addition with comment markers.

**Comment Markers Used**

* **Start of Security Improvements:**

python

Copy code

## start of security improvements added on 2024-01-12

* **End of Security Improvements:**

python

Copy code

## end of security improvements added on 2024-01-12

**2. Directory Traversal Vulnerability Fix**

**File Modified**

* **File Path:** mlflow/store/artifact/local\_artifact\_repo.py

**New Function Added**

* **Function Name:** validate\_user\_supplied\_path(path, base\_directory)

**Functions Updated**

* Integrated validation into:
  + download\_artifacts
  + \_download\_file
  + delete\_artifacts

**Location of Changes**

1. **Added Validation Function**
   * Placed after the import statements and before the LocalArtifactRepository class definition.
   * Encapsulated within comment markers.
2. **Updated Existing Functions**
   * Inserted calls to validate\_user\_supplied\_path at the beginning of the affected methods.
   * Each insertion is wrapped with comment markers.

**Comment Markers Used**

* **Start of Security Improvements:**

python

Copy code

## start of security improvements added on 2024-01-12

* **End of Security Improvements:**

python

Copy code

## end of security improvements added on 2024-01-12

**Updated Code with Comments**

**1. Code Injection Vulnerability Fix**

python

Copy code

import subprocess

import os

from mlflow.exceptions import MlflowException

from mlflow.utils.os import is\_windows

# Other imports and code...

## start of security improvements added on 2024-01-12

def \_validate\_and\_sanitize\_command(command):

# Validates the command to prevent code injection attacks

dangerous\_chars = [';', '&', '|', '`', '$', '>', '<']

for char in dangerous\_chars:

if char in command:

raise MlflowException(f"Invalid character '{char}' detected in command.")

if not command.strip():

raise MlflowException("Command is empty or whitespace.")

## end of security improvements added on 2024-01-12

def \_run\_entry\_point(command, work\_dir, experiment\_id, run\_id):

## start of security improvements added on 2024-01-12

\_validate\_and\_sanitize\_command(command)

## end of security improvements added on 2024-01-12

env = os.environ.copy()

# Environment variable setup...

\_logger.info("=== Running command '%s' in run with ID '%s' === ", command, run\_id)

if not is\_windows():

process = subprocess.Popen(["bash", "-c", command], close\_fds=True, cwd=work\_dir, env=env)

else:

process = subprocess.Popen(["cmd", "/c", command], close\_fds=True, cwd=work\_dir, env=env)

return LocalSubmittedRun(run\_id, process)

# Rest of the code...

**2. Directory Traversal Vulnerability Fix**

python

Copy code

import os

import shutil

from typing import Any

from mlflow.store.artifact.artifact\_repo import (

ArtifactRepository,

try\_read\_trace\_data,

verify\_artifact\_path,

)

from mlflow.tracing.artifact\_utils import TRACE\_DATA\_FILE\_NAME

from mlflow.utils.file\_utils import (

get\_file\_info,

list\_all,

local\_file\_uri\_to\_path,

mkdir,

relative\_path\_to\_artifact\_path,

)

from mlflow.utils.uri import validate\_path\_is\_safe

## start of security improvements added on 2024-01-12

def validate\_user\_supplied\_path(path: str, base\_directory: str) -> str:

"""

Validates and sanitizes user-supplied paths to prevent directory traversal attacks.

Args:

path: The user-supplied path to validate.

base\_directory: The base directory that all paths should be confined to.

Returns:

The sanitized absolute path if valid.

Raises:

ValueError: If the path is invalid or attempts to traverse outside the base directory.

"""

# Ensure the input is a string

if not isinstance(path, str):

raise ValueError("The path must be a string.")

# Manually decode URL-encoded characters (e.g., '%2e' -> '.')

def manual\_unquote(s):

res = ''

i = 0

while i < len(s):

if s[i] == '%' and i + 2 < len(s):

hex\_value = s[i + 1:i + 3]

try:

res += chr(int(hex\_value, 16))

i += 3

except ValueError:

res += s[i]

i += 1

else:

res += s[i]

i += 1

return res

path = manual\_unquote(path)

# Normalize the path to remove traversal sequences

normalized\_path = os.path.normpath(path)

# Prevent paths that navigate to parent directories or are absolute

if normalized\_path.startswith("..") or os.path.isabs(normalized\_path):

raise ValueError("Invalid path: Path traversal is not allowed.")

if any(part == ".." for part in normalized\_path.split(os.sep)):

raise ValueError("Invalid path: Path traversal is not allowed.")

# Construct the absolute path

absolute\_path = os.path.abspath(os.path.join(base\_directory, normalized\_path))

# Ensure the path is within the base directory

base\_dir\_abs = os.path.abspath(base\_directory) + os.sep

if not absolute\_path.startswith(base\_dir\_abs):

raise ValueError("Invalid path: Access outside the allowed directory is prohibited.")

return absolute\_path

# Fix implemented on 2024-01-12: Added input validation to prevent directory traversal attacks.

## end of security improvements added on 2024-01-12

class LocalArtifactRepository(ArtifactRepository):

"""Stores artifacts as files in a local directory."""

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.\_artifact\_dir = local\_file\_uri\_to\_path(self.artifact\_uri)

@property

def artifact\_dir(self):

return self.\_artifact\_dir

# Existing methods...

def download\_artifacts(self, artifact\_path, dst\_path=None):

if dst\_path:

return super().download\_artifacts(artifact\_path, dst\_path)

## start of security improvements added on 2024-01-12

# Validate the user-supplied artifact\_path

artifact\_path = validate\_user\_supplied\_path(artifact\_path or "", self.artifact\_dir)

## end of security improvements added on 2024-01-12

local\_artifact\_path = os.path.join(self.artifact\_dir, os.path.normpath(artifact\_path))

if not os.path.exists(local\_artifact\_path):

raise OSError(f"No such file or directory: '{local\_artifact\_path}'")

return os.path.abspath(local\_artifact\_path)

def \_download\_file(self, remote\_file\_path, local\_path):

## start of security improvements added on 2024-01-12

# Validate the user-supplied remote\_file\_path

remote\_file\_path = validate\_user\_supplied\_path(remote\_file\_path, self.artifact\_dir)

## end of security improvements added on 2024-01-12

remote\_file\_path = os.path.join(self.artifact\_dir, os.path.normpath(remote\_file\_path))

shutil.copy2(remote\_file\_path, local\_path)

def delete\_artifacts(self, artifact\_path=None):

## start of security improvements added on 2024-01-12

# Validate the user-supplied artifact\_path

artifact\_path = validate\_user\_supplied\_path(artifact\_path or "", self.\_artifact\_dir)

## end of security improvements added on 2024-01-12

artifact\_path = local\_file\_uri\_to\_path(artifact\_path)

if os.path.exists(artifact\_path):

if os.path.isfile(artifact\_path):

os.remove(artifact\_path)

else:

shutil.rmtree(artifact\_path)

# Rest of the code...

**Explanation of Changes**

**1. Code Injection Vulnerability Fix**

**New Function: \_validate\_and\_sanitize\_command**

* **Purpose:** Ensures the command string does not contain characters that could be exploited for code injection.
* **Implementation Details:**
  + **Dangerous Characters:** Defined a list dangerous\_chars containing ;, &, |, `, $, >, <.
  + **Validation Logic:** Iterated over each character in dangerous\_chars to check for their presence in the command string.
    - Raised an MlflowException if any dangerous character is found.
  + **Empty Command Check:** Verified that the command is not empty or just whitespace.

**Modified Function: \_run\_entry\_point**

* **Change Made:** Added a call to \_validate\_and\_sanitize\_command(command) at the beginning.
* **Purpose:** Validates the command before execution to prevent malicious code from running.

**2. Directory Traversal Vulnerability Fix**

**Added Function: validate\_user\_supplied\_path**

* **Purpose:** Sanitizes user-supplied paths to prevent directory traversal attacks.
* **Implementation Details:**
  + **Type Check:** Ensured path is a string.
  + **Manual URL Decoding:** Decoded URL-encoded characters to neutralize encoded traversal sequences.
  + **Normalization:** Used os.path.normpath to resolve . and .. in paths.
  + **Traversal Checks:**
    - Rejected paths starting with ".." or absolute paths.
    - Ensured no segment of the path is "..".
  + **Path Construction:** Built the absolute path within base\_directory.
  + **Directory Constraint:** Verified the absolute path starts with base\_directory to prevent access outside the allowed directory.

**Updated Methods with Validation**

* **Functions Modified:**
  + download\_artifacts
  + \_download\_file
  + delete\_artifacts
* **Changes Made:**
  + Inserted calls to validate\_user\_supplied\_path at the beginning of each method.
* **Effect:** Ensured all file operations use validated paths, preventing directory traversal exploits.

**Conclusion**

By implementing these security improvements, we have effectively mitigated two critical vulnerabilities:

* **Code Injection Fix:** Input validation for command strings in \_run\_entry\_point prevents unauthorized code execution, ensuring only safe commands are executed.
* **Directory Traversal Fix:** Robust path validation in local\_artifact\_repo.py ensures file operations are confined within the authorized directory, protecting against unauthorized file access and deletion.

These enhancements significantly strengthen the security posture of the MLflow project, safeguarding it against potential exploitation.