Authenticated Remote Code Execution in latest OpenNMS over MQ channel

Installing the latest version of **OpenNMS** in a Docker container (see Installation guide) allowed for an easy security testing setup in a controlled environment. After reading the installation manual etc., it became clear that besides the already known web-interface (and associated *REST* services) another communication channel was used by special kind of *OpenNMS clients* (so called Minions): **MQ channels** for *ActiveMQ* bidirectional client communication.

According to the installation manual, a minion user with the security role *ROLE_MINION* (Minimal amount of permissions required for a Minion to operate) has to be added. This account is then used for authentication by the remote minion station to communicate primarily via an MQ channel listening on port **tcp/61616** at the Horizon base station.

Based on research of Matthias Kaiser, a tool was published 4 years ago at Blackhat USA 2016 (JMET - Java Message Exploitation Tool):

> JMET was released at Blackhat USA 2016 and is an outcome of Code White's research effort presented in the talk "Pwning Your Java Messaging With Deserialization Vulnerabilities". The goal of JMET is to make the exploitation of the Java Message Service (JMS) easy. In the talk more than 12 JMS client implementations where shown, vulnerable to deserialization attacks. The specific deserialization vulnerabilities were found in ObjectMessage implementations (classes implementing javax.jms.ObjectMessage). The following more or less complete list shows the vulnerable JMS broker client libraries...

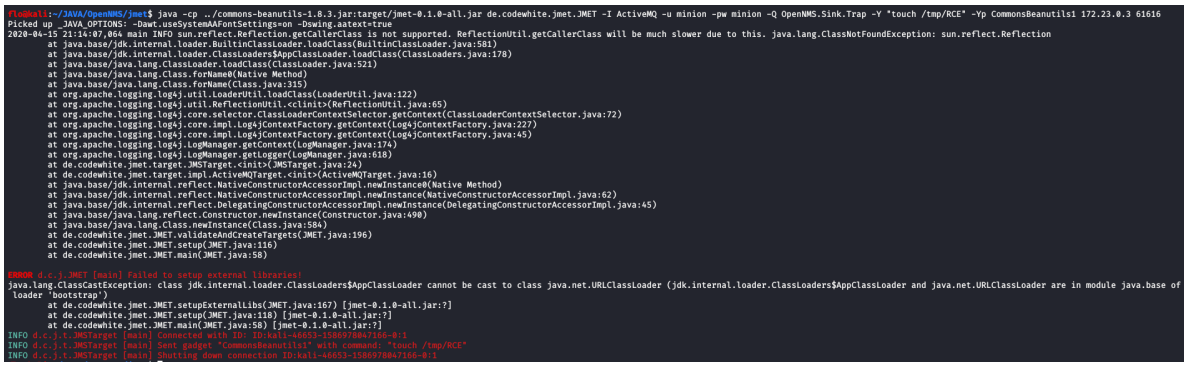
One of these vulnerable clients was *Apache ActiveMQ* which indeed is used by OpenNMS for the bidirectional communication mentioned above.

Since *JMET* has to be weaponized with serialized payloads, these are created by another famous tool called ysoserial. Since JMET was released quite some time ago, several *current gadgets* were missing. So a current ysoserial version was compiled into JMET to support all these gadgets. As it turned out, one well-known gadget named **CommonsBeanUtils1** was sufficient to trigger a *Remote Code Execution (RCE)* at the receiver of an MQ message taken from the broker.

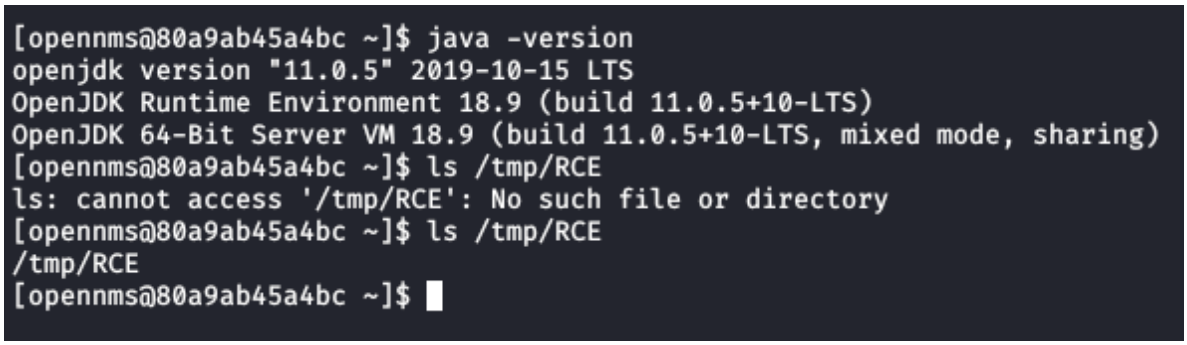
Using the off-the-shelf ysoserial payload did not succeed since the *serialVersionUID* wasn't matching which could be observed in Horizon's server logs. Since ysoserial uses *commons-beanutils* in version 1.9.2 compared to the deployed version 1.8.3 at the Horizon backend, a simple replacement of the library during payload generation was the quick fix.

After trying several *queue names and topics* known to be existing at the MQ broker of Horizon, finally an RCE was possible.

Execution of the JMET tool using the minion user credentials and a proper queue name.



Shell on Horizon base station (dockerized), showing the JDK version, and showing the file path proving RCE (before and) after sending the serialized payload via MQ.



Of course from now on, remote code (and therefore command) execution was possible to establish a reverse shell, alter the Horizon configuration (e.g. upgrade to admin) etc.