Runtime Infrastructure - Bug #2723

intermittent failure in reading loop on P2J server startup in Java 8

09/17/2015 02:24 PM - Eric Faulhaber

Status:	New	**Start date:**	09/15/2015
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 09/17/2015 02:29 PM - Eric Faulhaber

This is something I've only seen since moving to Java 8. It happens intermittently on server startup, when launching app server agents (possibly a race condition?):

```
Sep 15, 2015 11:54:34 AM Protocol.Reader.run()
WARNING: {Reader} failure in reading loop
javax.net.ssl.SSLProtocolException: Data received in non-data state: 6
    at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:1097)
    at sun.security.ssl.SSLSocketImpl.readDataRecord(SSLSocketImpl.java:918)
    at sun.security.ssl.AppInputStream.read(AppInputStream.java:105)
    at sun.security.ssl.AppInputStream.read(AppInputStream.java:71)
    at java.io.ObjectInputStream$PeekInputStream.peek(ObjectInputStream.java:2296)
    at java.io.ObjectInputStream$BlockDataInputStream.peek(ObjectInputStream.java:2589)
    at java.io.ObjectInputStream$BlockDataInputStream.peekByte(ObjectInputStream.java:2599)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1319)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:371)
    at com.goldencode.p2j.net.Protocol$Reader.run(Protocol.java:342)
    at java.lang.Thread.run(Thread.java:745)
```

The AppServerManager seems to recover by launching another agent, so we always have the right number of agents running by the time the server is done initializing. I haven't yet seen it happen with an interactive client.

In the configuration in which I've seen this, I've got some batch processes running. I don't yet know if it has to do with those.

#2 - 09/17/2015 04:18 PM - Eric Faulhaber

Looking into the sun.security.ssl.SSLSocketImpl source, state 6 is cs_CLOSED. This connection state is set in two places: a private closeInternal method and a package private fatal method. I think our case is going through closeInternal.

Inside SSLSocketImpl.readRecord(InputRecord r, boolean needAppData), there is a loop that reads the InputRecord while the connection state does not indicate an error or closed socket. closeInternal is called from within a catch(EOFException eof) block:

```
while (((state = getConnectionState()) != cs_CLOSED) &&
        (state != cs_ERROR) && (state != cs_APP_CLOSED)) {
    /*
     * Read a record ... maybe emitting an alert if we get a
     * comprehensible but unsupported "hello" message during
     * format checking (e.g. V2).
     */
    try {
        r.setAppDataValid(false);
        r.read(sockInput, sockOutput);
    } catch (SSLProtocolException e) {
        ...
    } catch (EOFException eof) {
        boolean handshaking = (getConnectionState() <= cs_HANDSHAKE);
        boolean rethrow = requireCloseNotify || handshaking;
        ...
        if (rethrow) {
            ...
        } else {
            // treat as if we had received a close_notify
            closeInternal(false);
            continue;
        }
    }
    ...
```

I think that's what is happening here. Further down in readRecord, the record is decrypted and then there is a switch on the content type. We end up in this logic:

```
case Record.ct_application_data:
    // Pass this right back up to the application.
    if (connectionState != cs_DATA
            && connectionState != cs_RENEGOTIATE
            && connectionState != cs_SENT_CLOSE) {
        throw new SSLProtocolException(
            "Data received in non-data state: " +
            connectionState);
    }
    ...
```

This is the exception which creates the above stack trace. So, it seems we have an input record which is supposed to have application data in it, but is hitting EOF when read.

EOFException is thrown from two places within InputRecord.read. In one case (a very early check to see if the SSL peer is gone), it's thrown if the existing record length is less than the SSLv3 header length of 5. In the other case, it's thrown when reading a record of unknown type, if the existing record length is less than the header length plus the buffer length, as read from the first byte of the buffer. I don't know which one is causing this situation. I'm setting breakpoints in each to see if I can catch it.

#3 - 09/17/2015 05:37 PM - Eric Faulhaber

Well, eclipse lets me set the breakpoints, but they're never hit, despite triggering the error. I guess the classes in rt.jar are compiled without debug information (-g:none).

Some external info on this subject suggests this is a race condition with the garbage collector:

- http://stackoverflow.com/questions/17979484/program-never-throws-exception-when-debugger-is-attached
- https://forums.aws.amazon.com/thread.jspa?messageID=438171