DESCRIPTION wy...@gmail.com created issue #1

Reproduceable on:
1. Dante SOCKS server for Ubuntu 12.0.4, setting to turn off authentication and allow all addresses
2. Galaxy Nexus running JWR66Y
3. JSOCKS library, a pure Java SOCKS client / server implementation

To enable SOCKS5 support in my app, I created a custom SocketImpl that attempts to wrap a JSOCKS Socket. The same code works on JVM 1.6 running on a desktop.

Include the code below in your project, and invoke urlTester.run(). In my application it is run under a newly spawn thread by onClick(View).

It crashes after printing "e0".

```
09-09 10:32:58.155 F/libc    ( 7592): Fatal signal 11 (SIGSEGV) at 0x00000008 (code=1), thread 7636 (Thread-71808)
09-09 10:32:58.296 I/DEBUG   (  123): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
09-09 10:32:58.296 I/DEBUG   (  123): Build fingerprint: 'google/yakju/maguro:4.3/JWR66Y/776638:user/release-keys'
09-09 10:32:58.296 I/DEBUG   (  123): Revision: '9'
09-09 10:32:58.296 I/DEBUG   (  123): pid: 7592, tid: 7636, name: Thread-71808  >>> com.test.app <<<
09-09 10:32:58.296 I/DEBUG   (  123): signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 00000008

09-09 10:32:58.874 I/DEBUG   (  123): backtrace:
09-09 10:32:58.874 I/DEBUG   (  123):    #00  pc 0004b802  /system/lib/libdvm.so
09-09 10:32:58.874 I/DEBUG   (  123):    #01  pc 00000923  /system/lib/libnativehelper.so (jniGetFDFromFileDescriptor+14)
09-09 10:32:58.874 I/DEBUG   (  123):    #02  pc 00022a51  /system/lib/libjavacore.so
09-09 10:32:58.874 I/DEBUG   (  123):    #03  pc 0001dc4c  /system/lib/libdvm.so (dvmPlatformInvoke+112)
09-09 10:32:58.874 I/DEBUG   (  123):    #04  pc 0004decf  /system/lib/libdvm.so (dvmCallJNIMethod(unsigned int const*, JValue*, Method const*, Thread*)+398)
09-09 10:32:58.874 I/DEBUG   (  123):    #05  pc 00027060  /system/lib/libdvm.so
09-09 10:32:58.874 I/DEBUG   (  123):    #06  pc 0002b5ec  /system/lib/libdvm.so (dvmInterpret(Thread*, Method const*, JValue*)+184)
09-09 10:32:58.874 I/DEBUG   (  123):    #07  pc 0005ff21  /system/lib/libdvm.so (dvmCallMethodV(Thread*, Method const*, Object*, bool, JValue*, std::__va_list)+292)
09-09 10:32:58.874 I/DEBUG   (  123):    #08  pc 0005ff4b  /system/lib/libdvm.so (dvmCallMethod(Thread*, Method const*, Object*, JValue*, ...)+20)
09-09 10:32:58.874 I/DEBUG   (  123):    #09  pc 00054ccb  /system/lib/libdvm.so
09-09 10:32:58.874 I/DEBUG   (  123):    #10  pc 0000ca58  /system/lib/libc.so (__thread_entry+72)
09-09 10:32:58.874 I/DEBUG   (  123):    #11  pc 0000cbd4  /system/lib/libc.so (pthread_create+208)
09-09 10:32:58.874 I/DEBUG   (  123):
09-09 10:32:58.874 I/DEBUG   (  123): stack:
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b58  00000000
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b5c  5f061b7c
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b60  41ed5a20  /dev/ashmem/dalvik-heap (deleted)
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b64  4096034d  /system/lib/libdvm.so
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b68  41ed5a20  /dev/ashmem/dalvik-heap (deleted)
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b6c  40963ffb  /system/lib/libdvm.so
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b70  5b358538
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b74  5f061b9c
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b78  5b358538
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b7c  4096032b  /system/lib/libdvm.so
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b80  5b358538
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b84  00000000
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b88  400045a8
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b8c  576b0a60  /dev/ashmem/dalvik-LinearAlloc (deleted)
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b90  df0027ad
09-09 10:32:58.874 I/DEBUG   (  123):         5f061b94  00000000
09-09 10:32:58.874 I/DEBUG   (  123):    #00  5f061b98  400045a8
09-09 10:32:58.882 I/DEBUG   (  123):         5f061b9c  5b358538
09-09 10:32:58.882 I/DEBUG   (  123):         5f061ba0  576b0a60  /dev/ashmem/dalvik-LinearAlloc (deleted)
09-09 10:32:58.882 I/DEBUG   (  123):         5f061ba4  400045a8
09-09 10:32:58.882 I/DEBUG   (  123):         5f061ba8  5b358538
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bac  400bf925  /system/lib/libnativehelper.so (jniGetFDFromFileDescriptor+16)
09-09 10:32:58.882 I/DEBUG   (  123):    #01  5f061bb0  fffffef0
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bb4  59caaa55  /system/lib/libjavacore.so
09-09 10:32:58.882 I/DEBUG   (  123):    #02  5f061bb8  0000a632
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bbc  00000000
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bc0  00000000
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bc4  0000001c
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bc8  0000a632
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bcc  32a6000a
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bd0  00000000
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bd4  00000000
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bd8  00000000
09-09 10:32:58.882 I/DEBUG   (  123):         5f061bdc  ffff0000  [vectors]
09-09 10:32:58.882 I/DEBUG   (  123):         5f061be0  8565960a
09-09 10:32:58.882 I/DEBUG   (  123):         5f061be4  00000000
```

```
09-09 10:32:58.882 I/DEBUG   (  123):        5f061be8  00000000
09-09 10:32:58.882 I/DEBUG   (  123):        5f061bec  00000000
09-09 10:32:58.882 I/DEBUG   (  123):        5f061bf0  00000000
09-09 10:32:58.882 I/DEBUG   (  123):        5f061bf4  00000000
```

```
// BEGINS CODE SNIPPET

   import socks.SocksSocket;    // the jsocks socket class
   import socks.CProxy;         // the jsocks proxy class

   public static final class ClientSocksSocketImpl extends SocketImpl {
      SocksSocket delegate = null;
      InetAddress host;
      int port;

      public void setOption(int optID, Object value) throws SocketException {
      }

      public Object getOption(int optID) throws SocketException {
         return null;
      }

      private boolean ensure(InetAddress host, int port) {
         // android-specific check: libcore would pass 0.0.0.0 to bind.  Assume it'll pass the "real" address at some point.
         // if this check is bypassed, crash is avoided but SOCKS server would reject the connection since it tries to connect to 0.0.0.0
         if (host.isAnyLocalAddress()) {
            android.util.Log.d(TAG, "e0"); // crashes!!!
            return false;
         }
         try {
            if (delegate == null) {
               this.host = host;
               this.port = port;
               delegate = new SocksSocket(host, port);
               return true;
            } else {
               return true;
            }
         } catch (Exception e) {
            android.util.Log.d(TAG, "e1", e);
            throw new RuntimeException(e);
         }
      }
      protected void accept(SocketImpl s) {
         throw new UnsupportedOperationException();
      }
      protected int available() {
         try {
            if (delegate != null) {
               return getInputStream().available();
            } else {
               return 0;
            }
         } catch (Exception e) {
            android.util.Log.d(TAG, "e2");
            throw new RuntimeException(e);
         }
      }
      protected void bind(InetAddress host, int port) {
         try {
            if (ensure(host, port)) {
               delegate.bind(new InetSocketAddress(host, port));
            }
         } catch (Exception e) {
            android.util.Log.d(TAG, "e3", e);
            throw new RuntimeException(e);
         }
      }
      protected void close() {
         try {
            if (delegate != null) {
               delegate.close();
            }
         } catch (Exception e) {
            android.util.Log.d(TAG, "e4");
            throw new RuntimeException(e);
         }
      }
      protected void connect(InetAddress address, int port) {
         try {
            if (ensure(address, port)) {
               delegate.connect(new InetSocketAddress(address, port));
            }
```

```
            } catch (Exception e) {
                android.util.Log.d(TAG, "e5");
                throw new RuntimeException(e);
            }
        }
        protected void connect(SocketAddress address, int timeout) {
            try {
                if (ensure(((InetSocketAddress)address).getAddress(), ((InetSocketAddress)address).getPort())) {
                    delegate.connect(address, timeout);
                }
            } catch (Exception e) {
                android.util.Log.d(TAG, "e6");
                throw new RuntimeException(e);
            }
        }
        protected void connect(String host, int port) {
            try {
                if (ensure(InetAddress.getByName(host), port)) {
                    delegate.connect(new InetSocketAddress(host, port));
                }
            } catch (Exception e) {
                android.util.Log.d(TAG, "e7");
                throw new RuntimeException(e);
            }
        }
        protected void create(boolean stream) {
            // stream is ignored, use datagramsocketimpl
        }
        protected InputStream getInputStream() {
            if (delegate == null) {
                return null;
            }
            return delegate.getInputStream();
        }
        protected OutputStream getOutputStream() {
            if (delegate == null) {
                return null;
            }
            return delegate.getOutputStream();
        }
        protected void listen(int backlog) {
            throw new UnsupportedOperationException();
        }
        protected void sendUrgentData(int data) {
            try {
                if (delegate != null) {
                    delegate.sendUrgentData(data);
                }
            } catch (Exception e) {
                android.util.Log.d(TAG, "e8");
                throw new RuntimeException(e);
            }
        }
    }

    public static final class ClientSocksSocketImplFactory implements SocketImplFactory {
        public SocketImpl createSocketImpl() {
            //java.net.SocksSocketImpl / java.net.PlainSocketImpl
            // conditionally return the default implmenetation depending on who calls it.  If ClientSocksSocketImpl is always returned, we'll run into infinite recursion since socks.SocksSocket uses pla
to use itself.

            boolean calledByJSocks = false;

            StackTraceElement[] stack = Thread.currentThread().getStackTrace();
            if (stack.length >= 6) {
                String cln = stack[5].getClassName();
                if ("socks.CProxy".equals(cln) || "socks.SocksSocket".equals(cln)) {
                    calledByJSocks = true;
                }
            }

            if (calledByJSocks) {
                // return new java.net.PlainSocketImpl();
                // a protected java.net class
                try {
                    Class klass = Class.forName("java.net.PlainSocketImpl");
                    Constructor ctor = klass.getDeclaredConstructor();
                    ctor.setAccessible(true);
                    return (SocketImpl)ctor.newInstance();
                } catch (ClassNotFoundException cnfe) {
                    android.util.Log.d(TAG, "cnfe");
                    return null;
                } catch (NoSuchMethodException nsme) {
                    android.util.Log.d(TAG, "nsme");
                    return null;
                } catch (InstantiationException ie) {
```

```
                android.util.Log.d(TAG, "ie");
                return null;
            } catch (IllegalAccessException iae) {
                android.util.Log.d(TAG, "iae");
                return null;
            } catch (InvocationTargetException ite) {
                android.util.Log.d(TAG, "ite");
                return null;
            }
        }

        return new ClientSocksSocketImpl();
    }
}

private final Runnable urlTester = new Runnable() {
    private static final String strUrl= "http://www.android.com:80/";

    private void setup() {
        try {
            CProxy.setDefaultProxy(PROXY_ADDR, PROXY_PORT);
            Socket.setSocketImplFactory(new ClientSocksSocketImplFactory());
        } catch (IOException ioe) {
            android.util.Log.d(TAG, "jsocks setting default proxy failed");
            throw new RuntimeException(ioe);
        }
    }

    private void javaNetTest() {
        try {
            android.util.Log.d(TAG, "BEGIN java.net Test");
            URL url = new URL(strUrl);
            HttpURLConnection urlConn = (HttpURLConnection) url.openConnection();
            android.util.Log.d(TAG, "java.net RESPONSE CODE ========================== " + urlConn.getResponseCode());
            android.util.Log.d(TAG, "END java.net Test");
        } catch (IOException e) {
            android.util.Log.d(TAG, "java.net: Error creating HTTP connection", e);
        } catch (RuntimeException re) {
            android.util.Log.d(TAG, "java.net: PROXY DOWN.  CONNECTION FAILED");
        }
    }

    @Override
    public void run() {
        setup();
        javaNetTest();
    }
};

// ENDS CODE SNIPPET
```

✓ Links (7)

“http://com.test.app”
“http://java.net”
“ private static final String strUrl= " http://www.android.com:80/ ";”
“You can follow the instructions at http://android-developers.blogspot.com/2011/07/debugging-android-jni-with-checkjni.html”
“...ubmitted https://android-review.googlesource.com/#/c/68897/ to prevent the VM crash. There's still a bug somewhere in our java layer that's passing a null file descriptor to the native code. I can't

See all related links

COMMENTS

All c

wy...@gmail.com <wy...@gmail.com> #2

PROXY_ADDR and PROXY_PORT are the IP addresses and port of the Dante Proxy in the LAN reachable from the phone through WiFi.

en...@google.com <en...@google.com>

Assigned to en...@google.com.

na...@google.com <na...@google.com> #3

Reassigned to na...@google.com.

wy2... : Can you confirm that this issue is 100% reproducible with the setup you've described ? (i.e, that it isn't a sporadic crash).

na...@google.com <na...@google.com> #4

So, I can't reproduce this issue locally.

I'm pretty sure the issue is that someone is passing in a null FileDescriptor to jniGetFdFromFileDescriptor, but I can't fix the issue without being able to reproduce it.

Do you mind turning on CheckJni and capturing a log from the crash ? (adb logcat)
You can follow the instructions at http://android-developers.blogspot.com/2011/07/debugging-android-jni-with-checkjni.html

---

na...@google.com <na...@google.com> #5

Ping ? I need more information to resolve this issue.

---

na...@google.com <na...@google.com> #6

Status: Won't Fix (Not Reproducible)

I submitted https://android-review.googlesource.com/#/c/68897/ to prevent the VM crash. There's still a bug somewhere in our java layer that's passing a null file descriptor to the native cod

---

wy...@gmail.com <wy...@gmail.com> #7

Yup, it is 100% reproduceable, and not a sporadic crash...

---

wy...@gmail.com <wy...@gmail.com> #8

Will try to run CheckJni.

---

ka...@gmail.com <ka...@gmail.com> #9

I am running into basically the same issue with using custom SocketImplFactory:

```
03-06 23:07:35.533 I/DEBUG   (  171): backtrace:
03-06 23:07:35.533 I/DEBUG   (  171):    #00  pc 0004ba5a  /system/lib/libdvm.so
03-06 23:07:35.533 I/DEBUG   (  171):    #01  pc 00001dad  /system/lib/libnativehelper.so (jniGetFDFromFileDescriptor+80)
03-06 23:07:35.533 I/DEBUG   (  171):    #02  pc 0001dbe5  /system/lib/libjavacore.so
03-06 23:07:35.533 I/DEBUG   (  171):    #03  pc 0001dbcc  /system/lib/libdvm.so (dvmPlatformInvoke+112)
03-06 23:07:35.533 I/DEBUG   (  171):    #04  pc 0004e123  /system/lib/libdvm.so (dvmCallJNIMethod(unsigned int const*, JValue*, Method const*, Thread*)+398)
03-06 23:07:35.533 I/DEBUG   (  171):    #05  pc 00026fe0  /system/lib/libdvm.so
03-06 23:07:35.533 I/DEBUG   (  171):    #06  pc 0002dfa0  /system/lib/libdvm.so (dvmMterpStd(Thread*)+76)
03-06 23:07:35.533 I/DEBUG   (  171):    #07  pc 0002b638  /system/lib/libdvm.so (dvmInterpret(Thread*, Method const*, JValue*)+184)
03-06 23:07:35.533 I/DEBUG   (  171):    #08  pc 00060581  /system/lib/libdvm.so (dvmCallMethodV(Thread*, Method const*, Object*, bool, JValue*, std::__va_list)+336)
03-06 23:07:35.533 I/DEBUG   (  171):    #09  pc 000605a5  /system/lib/libdvm.so (dvmCallMethod(Thread*, Method const*, Object*, JValue*, ...)+20)
03-06 23:07:35.533 I/DEBUG   (  171):    #10  pc 0005528b  /system/lib/libdvm.so
03-06 23:07:35.533 I/DEBUG   (  171):    #11  pc 0000d170  /system/lib/libc.so (__thread_entry+72)
03-06 23:07:35.533 I/DEBUG   (  171):    #12  pc 0000d308  /system/lib/libc.so (pthread_create+240)
03-06 23:07:35.533 I/DEBUG   (  171):
```

CheckJni isn't giving me any additional warnings, although I do see the message about it enabled at the start. 100% reproducible, as long as I am connecting to a valid host. If the connection

---

na...@google.com <na...@google.com> #10

Note that I fixed one cause of the hard crash in https://android-review.googlesource.com/#/c/68897 .

Given that I can't reproduce it locally, could you construct a simplified test case that demonstrates this problem ?

---

ka...@gmail.com <ka...@gmail.com> #11

created: http://hxbc.us/misc/issue59907.tgz

---

ka...@gmail.com <ka...@gmail.com> #12

Nevermind, the reason fd is null is because of a bug in the way I wrapped PlainSocketImpl. I'd say it's a bug in the way PlainSocketImpl and SocketImpl is coupled too, but that's beyond the sc

---

na...@google.com <na...@google.com> #13

Status: Won't Fix (Intended Behavior)

Yes, I expected that would be the problem (See my comment in #7) .

---

wy...@gmail.com <wy...@gmail.com> #14

Hello, I disagree it is a user error - the DVM should have thrown a Java Exception instead of dying itself.

Partially related - is there any plan for SOCKS5 socket support so Java code from other platforms can be ported to work on Android?

Thanks.

an...@yahoo.com <an...@yahoo.com> #15

For anyone who might get this same issue:

The reason kah..@gmail.com (and probably wy2..@gmail.com) got the crash is that the FileDescriptor field of SocketImpl is accessed directly by java.net.Socket rather than a method call. Th being wrapped around was initialized but not the one of the wrapper class (the one java.net.Socket sees).

The null FileDescriptor was then passed to native code causing a segfault.

This can be fixed as such:

```
public class MyWrapper extends SocketImpl {

SocketImpl impl;

public MyWrapper() {
  FileDescriptor fd = new FileDescriptor();
  this.impl = (SocketImpl) Class.forName("java.net.PlainSocketImpl").getConstructor(FileDescriptor.class).newInstance(fd);
  this.fd = fd;
}
}
```