MockParser

MockParser

Background

So, you've integrated Apache Tika into your framework, tried it on a couple of thousand files and all works well. Problem solved!

No.

In very, very rare cases, Tika can do some really bad things. We try to fix these problems when we can, but if history is any indication (e.g. TIKA-1132 and TIKA-2040 to name a few), if you are processing millions/billions of files from the wild, you'll need to defend against:

1. Regular catchable exceptions
2. OutOfMemory errors which can put the jvm in an unreliable state
3. Permanent hangs (Tika can chew up massive amounts of resources and go *forever*)
4. Security vulnerabilities (e.g. CVE-2016-6809 and CVE-2016-4434)

Please note that for 3., permanent hangs – you cannot terminate the Thread. Thread's *stop*, *suspend*, *destroy* sound like they'll do the trick, but they won't. **You need to kill the entire process.** See TIKA-456.

As of Tika 1.15, we added a MockParser in the tika-core-tests.jar that will allow you to test your framework against items 1-3. Simply add that jar to your class path and then include a <mock> xml file in your set of test documents, and crash, crash away.

If you'd like to test 4., you can do that too! While you should be protected from an XXE (let us know if you're not!), you could create a deserialization attack...just create your own malicious Throwable class, add it to the classpath and send in a mock file that includes:

```
<throw class="my.evil.DeserializationAttack">bwahahaha</throw>
```

Usage

Below are several options for adding the dependency.

Including the tika-core-tests dependency in your project

```
<dependency>
  <groupId>${project.groupId}</groupId>
  <artifactId>tika-core</artifactId>
  <version>${project.version}</version>
  <type>test-jar</type>
  <scope>test</scope>
</dependency>
```

Tika-app

Place the tika-app.jar and the tika-core-tests.jar in a "bin" directory.

```
java -cp "bin/*" org.apache.tika.TikaCLI mock_example.xml
```

Tika-server

Place the tika-server.jar and the tika-core-tests.jar in a "bin" directory.

```
java -cp "bin/*" org.apache.tika.server.TikaServerCli
```

Then curl away:

```
curl -T mock_example.xml http://localhost:9998/rmeta/text
```

Your Framework

Place the tika-core-tests.jar on your class path (NOT IN PRODUCTION!!!) and then add some mock.xml files to your batch of documents.

Mock options

See the mock example.xml file in tika-parsers/src/test/resources/test-documents/mock.

This shows all of the examples of what you can do.

```xml
<?xml version="1.0" encoding="UTF-8" ?>

<mock>
    <!-- this file offers all of the options as documentation.
    Parsing will stop at an IOException, of course
    -->

    <!-- action can be "add" or "set" -->
    <metadata action="add" name="author">Nikolai Lobachevsky</metadata>

    <!-- element is the name of the sax event to write, p=paragraph
        if the element is not specified, the default is <p> -->

    <write element="p">some content</write>

    <!-- write something to System.out -->
    <print_out>writing to System.out</print_out>

    <!-- write something to System.err -->
    <print_err>writing to System.err</print_err>

    <!-- hang
        millis: how many milliseconds to pause.  The actual hang time will probably
            be a bit longer than the value specified.
        heavy: whether or not the hang should do something computationally expensive.
            If the value is false, this just does a Thread.sleep(millis).
            This attribute is optional, with default of heavy=false.
        pulse_millis: (required if "heavy" is true), how often to check to see
            whether the thread was interrupted or that the total hang time exceeded the millis
        interruptible: whether or not the parser will check to see if its thread
            has been interrupted; this attribute is optional with default of true
    -->
    <hang millis="100" heavy="true" pulse_millis="10" interruptible="true" />

    <!-- As of Tika 1.27/2.0, we've integrated FakeLoad (https://github.com/msigwart/fakeload)
         which enables much more precision for resource consumption than does
         "hang"

         millis = milliseconds to run, cpu is an integer % of the cpu to peg
         and mb is the integer amount of memory to consume in megabytes -->
    <fakeload millis="100" cpu="10" mb="10"/>

    <!-- throw an exception or error; optionally include a message or not -->
    <throw class="java.io.IOException">not another IOException</throw>

    <!-- perform a genuine OutOfMemoryError -->
    <oom/>

    <!-- perform a system exit...what parser would do that?!  We had one once... -->
    <system_exit/>

    <!-- interrupt the thread -->
    <thread_interrupt/>

    <!-- print to stdout -->
    <print_out>some junk</print_out>

    <!-- print to stderr -->
    <print_err>some junk</print_err>

</mock>
```

References

1. Tika to Ride 2. Evaluating Text Extraction