The TCPSniffer

Table of contents

The TCPSniffer is misnamed: its not a sniffer (like `snoop` or *Ethereal*) at all, but rather a *proxy* that you can place between in a TCP stream. It filters the request and response streams, sending the results to `stdout`. You can control its behaviour by specifying different filters. Whilst the TCPSniffer is very useful in its own right, its main purpose as far as The Grinder goes is to automatically generate scripts for the HTTP plugin.

> **Note:**
> If you are not interested in the ability to record scripts for The Grinder 2, use the The Grinder 3's TCPProxy (../g3/tcpproxy.html) instead. It has more features and fixes.

1 Starting the TCPSniffer

You invoke the TCPSniffer with something like:

```
CLASSPATH=/opt/grinder/lib/grinder.jar
export CLASSPATH

java net.grinder.TCPSniffer
```

Say `java net.grinder.TCPSniffer -?` to get a list of the command line options.

Suppose you want to capture a conversation with a server on host *server*, port *7001*, you should say something like:

```
java net.grinder.TCPSniffer -remoteHost server
```

The TCPSniffer will start and display the following information:

```
 class="text">
Initialising standard sniffer engine with the parameters:
   Request filter:  net.grinder.tools.tcpsniffer.EchoFilter
   Response filter: net.grinder.tools.tcpsniffer.EchoFilter
   Local host:       localhost
   Local port:       8001
   Remote host:      localhost
   Remote port:      7001
Engine initialised, listening on port 8001
```

You can then point your web browser at `http://localhost:8001/` and exercise the application through the browser. The TCPSniffer will echo your requests to the terminal and forward the requests to `localhost:7001`, as well as echoing response from the server the terminal and returning them to the browser.

2 Using the TCPSniffer as an HTTP proxy

One problem of running the TCPSniffer as described above is that it only forwards to a single remote host. Any links or redirects to other hosts that the application returns to the browser will bypass the TCPSniffer, meaning that they will not feature in the test script. This also applies to absolute URLs to the server.

When recording browser traffic, a much better way to use the TCPSniffer is to run it as an HTTP proxy:

```
java net.grinder.TCPSniffer -proxy
```

This will make it listen as an HTTP proxy on port `8001` (the default, you can change it with `-localPort`), and forward requests onto the relevant remote host, while echoing out the HTTP interactions.

You should set your browser connection settings to specify the TCP sniffer as the HTTP proxy (set *host* to be the host on which the TCPSniffer is running and *port>* to be `8001`). You then use your browser as normal, e.g. in the example in the previous section you should use the direct address `http://localhost:7001` in your browser.

The TCPSniffer will run as a proxy for both HTTP and HTTPS if you specify `-ssl`.

3 Using the HttpPluginSnifferFilter

You can use the TCPSniffer to generate an HTTP plugin script segment suitable for use with The Grinder.

```
java net.grinder.TCPSniffer -proxy -httpPluginFilter
```

The output of the HttpPluginSnifferFilter looks like:

```
Initialising standard sniffer engine with the parameters:
   Request filter: net.grinder.plugin.http.HttpPluginSnifferFilter
   Response filter: net.grinder.tools.tcpsniffer.NullFilter
   Local host:       localhost
   Local port:       8001
   Listening as an HTTP proxy
   Engine initialised, listening on port 8001

  #
  # The Grinder version 2.8.3
  #
  # Script generated by the TCPSniffer at 25-Apr-02 08:17:57
  #

  grinder.processes=1
  grinder.threads=1
  grinder.cycles=0

  grinder.test0.sleepTime=11336
  grinder.test0.parameter.url=http://localhost:7001/
  grinder.test1.sleepTime=12168
  grinder.test1.parameter.url=http://localhost:7001/lah.html
  grinder.test2.sleepTime=411
  grinder.test2.parameter.url=http://localhost:7001/test.gif
  grinder.test3.sleepTime=4786
  grinder.test3.parameter.url=http://localhost:7001/lah.html
  grinder.test3.parameter.header.If-Modified-Since=Tue, 16 Jan 2001 16:26:42 GMT
  grinder.test4.sleepTime=311
  grinder.test4.parameter.url=http://localhost:7001/test.gif
  grinder.test4.parameter.header.If-Modified-Since=Mon, 06 Nov 2000 08:35:58 GMT
```

The script part of this is sent to the `stdout` stream, whereas the information messages are sent to `stderr`. You can redirect the script part to a file if you wish:

```
java net.grinder.TCPSniffer -proxy -httpPluginFilter > grinder.properties
```

You can then use this file with The Grinder.

4 SSL and HTTPS support

The TCPSniffer has SSL support. If you are using an old JVM, (earlier than Java SE 1.4.1) you must first install the JSSE (http://www.oracle.com/technetwork/java/jsse-136410.html) .

SSL relationships are necessarily point to point. When you interpose the TCPSniffer you end up with:

Client <--- ssl1 ---> TCPSniffer <--- ssl2 ---> Server

Where *ssl1* and *ssl2* are two separate SSL connections. Each SSL connection has its own set of client and server certificates (both of which are optional).

The TCPSniffer will negotiate appropriate certificates for both connections using certificates specified in a key store. See the JSSE documentation for how to set up a key store. There are three parameters you can pass as command line options to the TCPSniffer to specify key store details:

`-keyStore` *file*	The key store file.
`-keyStorePassword` *password*	The password for the key store.
`-keyStoreType` *type*	The type, defaults to `jks`.

You can also specify these with the corresponding `javax.net.ssl.XXX` properties.

Here's an example of starting the TCPSniffer as an HTTP/HTTPS proxy using the *testkeys* key store provided with the JSSE samples:

```
java net.grinder.TCPSniffer -ssl -proxy -keyStore testkeys -keyStorePassword passphrase
```

Even if you are not using client certificates, you probably need to specify a key store. This is because the proxy needs a server certificate of its own:

Browser -----> [ServerCert] Proxy ----> [ServerCert2] Target

You need to start the proxy with a key store containing a self-signed server certificate. This is the certificate that the browser will be presented with. If you fail to provide a server certificate, you will get a *No available certificate corresponds to the SSL cipher suites which are enabled* exception. The easiest way to provide a certificate is to copy the *testkeys* file from the JSSE samples distribution and start the sniffer using:

```
java net.grinder.TCPSniffer -ssl -proxy -keyStore testkeys -keyStorePassword passphrase
```

Alternatively you might want to generate your own. Here's an example:

```
PASTON:philipa% keytool -genkey -keystore testkeys -storepass passphrase -keyalg rsa
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:  Engineering
What is the name of your organization?
  [Unknown]:  Grinders Inc
What is the name of your City or Locality?
  [Unknown]:  Grindsville
What is the name of your State or Province?
  [Unknown]:  Grindshire
What is the two-letter country code for this unit?
  [Unknown]:  GR
```

```
  Is <CN=localhost, OU=Engineering, O=Grinders Inc, L=Grindsville, ST=Grindshire, C=GR
%gt; correct?
    [no]:  yes

  Enter key password for <mykey>
          (RETURN if same as keystore password):
  PASTON:philipa%
```

The *first and last name* ought to match the server which you run the proxy on, and you must specify `-keyalg rsa` to generate a certificate that works with common browsers. See the `keytool` notes in the JDK documentation for how to do more tricks.

You may find `NullPointerException`s when using PKCS12 files generated by some tools, e.g. `openssl`. Use the Java `keytool` utility to maintain key stores and you'll be all right.

5 The Sniff 'n' Grind web application

Sniff 'n; Grind is Paddy Spencer's J2EE web application that automates the tasks involved using the TCPSniffer to record and replay HTTPPlugin scripts. This section contains rather minimal notes on its use.

5.1 Setup page

In the setup page enter the starting URL; this is usually the front page of the application you're testing. You need to include the protocol in the URL, only HTTP and HTTPS are currently supported. Click on the button to go to...

5.2 Start TCPSniffer page

This page tells you what you need to change your browser proxy settings to. You need to so this before clicking on the link to start the page, otherwise your requests will go through the proxy you normally use and the sniffer won't pick them up. Once you've set the proxy, click on the *Click here to go to...* link, and the start URL will be returned in a new window.

If you specify a certificate in the `web.xml` file, and a secure starting URL (an HTTPS rather than HTTP one) then the web application will let you sniff and grind your ssl-using web application. The only difference you'll notice is that you will be asked to accept an untrusted certificate from the server. This is because the web application uses the certificate you give it as a server certificate when you connect to it and as a client certificate when it connects to your web application. So you'd better make sure that it's one your app will accept!

Do not close the sniffer window. If you do that you will not be able to end the test and get your results.

In order not to be a resource hog, the sniffer proxy process will timeout after a given number of seconds (configurable via the `web.xml`), so if you don't do anything for a while, you may find your proxy isn't there any more.

When you've finished the test, close the test window and rest your browser proxy to it's original settings (you DID note down those settings, didn't you?) and then click on the *stop* button. You will be taken to...

5.3 TCPSniffer results page

The Sniff 'n' Grind generates a number of files. As a minimum there are two:

- `httpsniffer.err` - which contains the initial startup information as well as any runtime errors that might have occurred.
- `httpsniffer.out` - which contains the details of the test(s).

If you wish to run the test manually, you need to copy these into your `grinder.properties` file and run The Grinder in the normal way. You will also need to cut and paste the various `http-plugin-sniffer-post` files, if any.

If you instead want to run The Grinder right now, with your recorded results, then click on the link and go to...

5.4 The Grinder setup page

Despite, or perhaps because of, the vast, bewildering array of properties that can be set to control a Grinder session, the web application (in its current incarnation) only allows you to set the numbers of processes, threads and cycles from within the browser. These default to one of each and have maximum settings (currently hard-coded) of 5 processes, 25 threads and 50 cycles.

The *reset* button resets the values in the form (as you'd expect) and *Grind me, baby!* does what it says, leading to...

5.5 Wait page

The patience page. If I was a real `31337 h4x0r d00d` I'd have written some kewl applet which would keep you entertained with graphical and highly amusing pr0n animations to keep you entertained while sneakily querying the server for whether The Grinder has finished. However, I'm not and so you've got a rather dull page with a five second refresh on it and a note saying, "Wait."

5.6 The Grinder results page

The results page simply presents the contents of the files in the `log` directory (so if you've used some huge number of threads and cycles, this page will be BIG) and gives you the options to re-run the grinder against the same test, but with different properties, or to record another test.