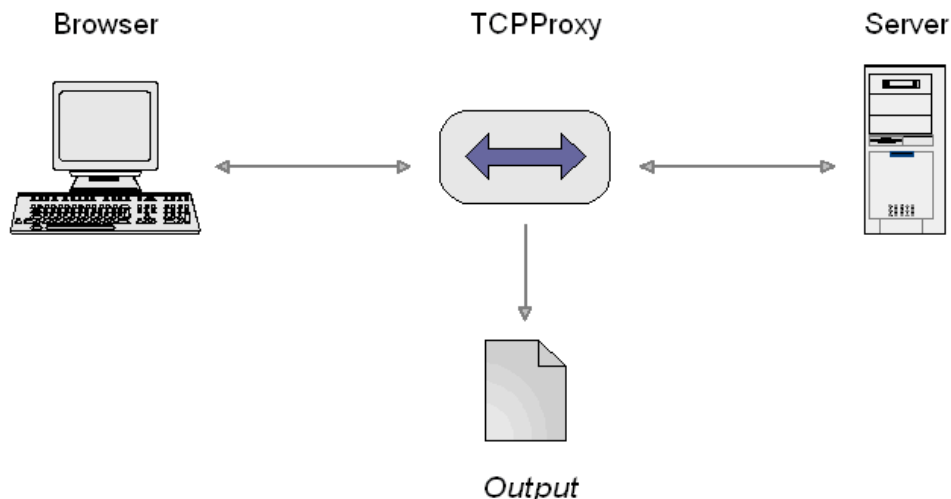


The TCPProxy

Table of contents

1 Starting the TCPProxy.....	2
2 Preparing the Browser.....	3
3 Using the EchoFilter.....	4
4 Using the HTTP TCPProxy filters.....	5
4.1 Altering the output with custom stylesheet.....	8
4.2 How to offset test numbers.....	8
5 SSL and HTTPS support.....	9
6 Using the TCPProxy with other proxies.....	10
7 Using the TCPProxy as a port forwarder.....	10
8 Summary of TCPProxy options.....	10

The TCPProxy is a proxy process that you can place in a TCP stream, such as the HTTP connection between your browser and a server. It filters the request and response streams, sending the results to the terminal window (`stdout`). You can control its behaviour by specifying different filters.



The TCPProxy's main purpose is to automatically generate HTTP test scripts that can be replayed with The Grinder's HTTP plugin. Because the TCPProxy lets you see what's going on at a network level it is also very useful as a debugging tool in its own right.

1. Starting the TCPProxy

You start the TCPProxy with something like:

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

```
java net.grinder.TCPProxy
```

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

With no additional options, the TCPProxy will start and display the following information:

```
Initialising as an HTTP/HTTPS proxy with the parameters:
  Request filters: EchoFilter
  Response filters: EchoFilter
  Local address: localhost:8001
Engine initialised, listening on port 8001
```

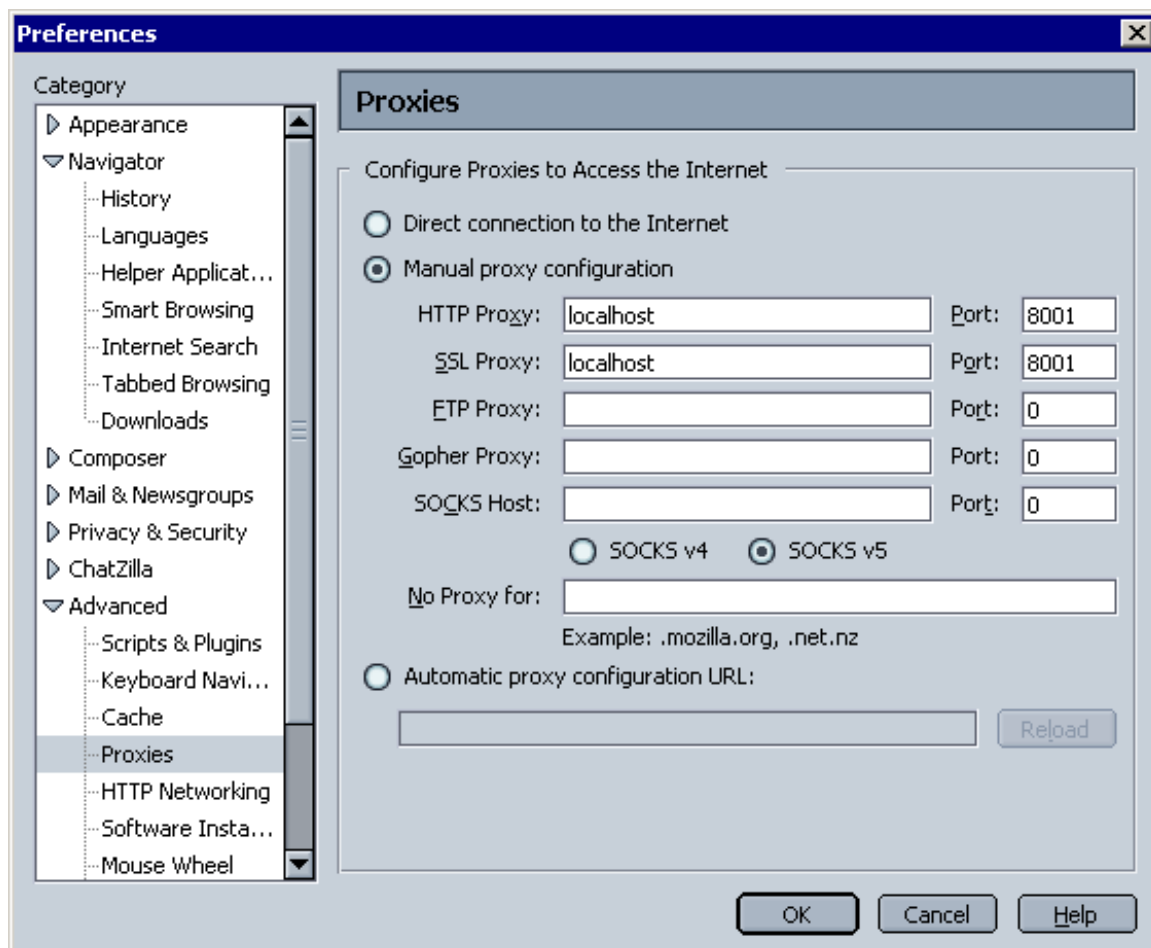
This indicates that the TCPProxy is listening as an HTTP proxy on port 8001 (the default, but you can change it with `-localPort`).

The TCPProxy appears to your browser just like any other HTTP proxy server, and you can use your browser as normal. If you type `http://grinder.sourceforge.net` into your browser it will display The Grinder home page and the TCPProxy will output all of the HTTP interactions between the browser and the SourceForge site.

The TCPProxy will proxy both HTTP and HTTPS. See [below](#) for details on customising the SSL configuration.

2. Preparing the Browser

You should now set your browser connection settings to specify the TCPProxy as the HTTP proxy. In the browser options dialog, set the proxy host to be the host on which the TCPProxy is running and proxy port to be 8001).



The relevant options dialog can be accessed by the following steps:

MSIE: Tools -> Internet Options -> Connections -> Local Area Network Settings.

Mozilla/Netscape: Edit -> Preferences -> Advanced - Proxies.

Mozilla/Firefox: Tools -> Options -> General -> Connection Settings.

Opera: Tools -> Preferences -> Advanced -> Network -> Proxy Servers.

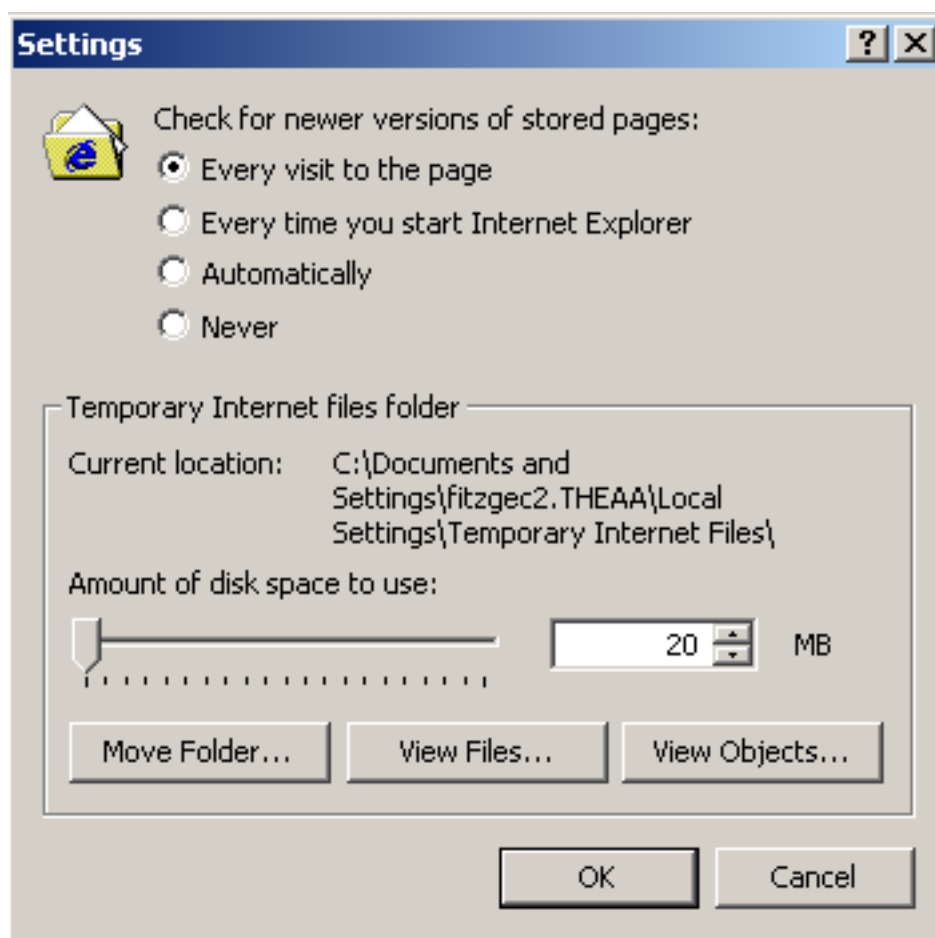
It is important to remember to remove any "bypass proxy server" or "No Proxy for" settings that you might have so that all the traffic flows through the TCPProxy and can be captured.

Note:

[Firefox](http://www.mozilla.com/) (<http://www.mozilla.com/>) users will find Jeremy Gillick's [SwitchProxy](https://addons.mozilla.org/quicksearch.php?q=switchproxy§ion=A&application=firefox) (<https://addons.mozilla.org/quicksearch.php?q=switchproxy§ion=A&application=firefox>) extension useful for switching to and from the TCPProxy.

It might also be a good idea to clear out any cache/temporary Internet files that might be on your workstation. On the other hand, you might decide not to do this if you want to

record a script representing a frequent user to your site who has images are [resources in their browser cache](#) (./faq.html#http-caching) . Also for IE users, changing the temporary Internet files settings to check for a newer version on every visit to a page can be useful.



3. Using the EchoFilter

The EchoFilter is the default filter used by the TCPProxy if no options are specified in the startup command. The EchoFilter outputs the stream activity to the terminal. It can be very useful for debugging as described in [this FAQ](#) (./faq.html#use-the-tcp-proxy) .

Bytes that do not have a printable ASCII representation are displayed in hexadecimal between square brackets. Here's some example output:

```
----- 127.0.0.1:2263->ads.osdn.com:80 -----
GET
/?ad_id=5839&alloc_id=12703&site_id=2&request_id=8320720&1102173982760
HTTP/1.1
Host: ads.osdn.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7.5)
Gecko/20041107 Firefox/1.0
Accept: image/png,*/*;q=0.5
Accept-Language: en-gb,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://sourceforge.net/projects/grinder
```

```
--- ads.osdn.com:80->127.0.0.1:2263 opened --
----- ads.osdn.com:80->127.0.0.1:2273 -----
HTTP/1.1 200 OK
Date: Sat, 04 Dec 2004 15:26:27 GMT
Server: Apache/1.3.29 (Unix) mod_gzip/1.3.26.1a mod_perl/1.2.9
Pragma: no-cache
Cache-control: private
Connection: close
Transfer-Encoding: chunked
Content-Type: image/gif

----- ads.osdn.com:80->127.0.0.1:2273 -----
80B
GIF89ae[00])[00D50000C3C3C3FEFDFD]hhhVVVyyy[F5CCD2D4D4D4CBCBCBD7]'F
```

Information lines are displayed to indicate the end point addresses and direction of the information flow and also whether a connection has just been opened or closed.

4. Using the HTTP TCPProxy filters

You can use the TCPProxy to generate an HTTP script suitable for use with The Grinder. The Grinder provides a pair of HTTP filters for this purpose. These filters are enabled by the `-http` command line option.

The first step is to start the TCPProxy with an HTTP filter:

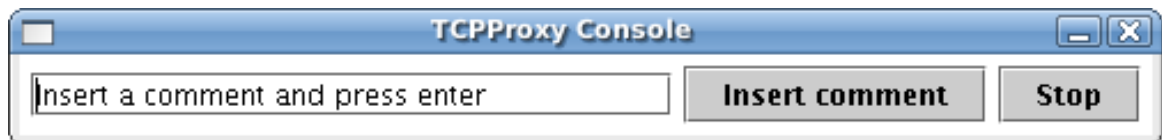
```
java net.grinder.TCPProxy -console -http > grinder.py
```

The `> grinder.py` part of the line sends the script to a file called `grinder.py`.

The terminal output of the TCPProxy looks like:

```
14/03/06 17:04:25 (tcpproxy): Initialising as an HTTP/HTTPS proxy with
the
parameters:
  Request filters:   HTTPRequestFilter
  Response filters: HTTPResponseFilter
  Local address:    localhost:8001
14/03/06 17:04:27 (tcpproxy): Engine initialised, listening on port 8001
```

The console (initiated by `-console`) displays a simple control window that allows the TCPProxy to be shut down cleanly. This is needed because some terminal shells, e.g. Cygwin bash, do not allow Java processes to be interrupted cleanly, so filters cannot rely on standard shutdown hooks. The console also allows a user to add ad-hoc commentary to the script during the recording. The console looks like this:



The TCPProxy console will be incorporated into the main [console](#) (`../g3/console.html`) in a future release.

Set your browser to use the TCPProxy as the HTTP proxy as [described earlier](#)), and run through your test scenario on your website.

Having finished your run through, press "Stop" on the TCPProxy console and the generated script will be written to `grinder.py`.

The `grinder.py` file contains headers, requests and a logical grouping of requests into pages, of the recorded tests.

For example, the headers section:

```
# The Grinder 3.0-beta29
# HTTP script recorded by TCPProxy at 14-Feb-2006 16:25:38

from net.grinder.script import Test
from net.grinder.script.Grinder import grinder
from net.grinder.plugin.http import HTTPPluginControl, HTTPRequest
from HTTPClient import NVPair
connectionDefaults = HTTPPluginControl.getConnectionDefaults()
httpUtilities = HTTPPluginControl.getHTTPUtilities()

# These definitions at the top level of the file are evaluated once,
# when the worker process is started.

connectionDefaults.defaultHeaders = \
    ( NVPair('User-Agent', 'Mozilla/5.0 (Windows; U; Windows NT 5.0;
en-US; rv:1.8.0.1) Gecko/20060111 Firefox/1.5.0.1'),
      NVPair('Accept-Encoding', 'gzip,deflate'),
      NVPair('Accept-Language', 'en-us,en;q=0.5'),
      NVPair('Accept-Charset', 'ISO-8859-1,utf-8;q=0.7,*;q=0.7'), )

headers0= \
    ( NVPair('Accept',
'text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,ima
)

headers1= \
    ( NVPair('Accept', 'text/css,*/*;q=0.1'),
      NVPair('Referer', 'http://localhost/'), )

headers2= \
    ( NVPair('Accept', 'text/css,*/*;q=0.1'),
      NVPair('Referer', 'http://localhost/css/product_styles.css'), )

headers3= \
    ( NVPair('Accept', 'image/png,*/*;q=0.5'),
      NVPair('Referer', 'http://localhost/css/colours.css'), )

headers4= \
    ( NVPair('Accept', 'image/png,*/*;q=0.5'),
      NVPair('Referer', 'http://localhost/'), )

#....
```

In the requests section each unique request is captured:

```
url0 = 'http://localhost:80'

# Create an HTTPRequest for each request, then replace the
# reference to the HTTPRequest with an instrumented version.
# You can access the unadorned instance using request101.__target__.
request101 = HTTPRequest(url=url0, headers=headers0)
request101 = Test(101, 'GET /').wrap(request101)

request102 = HTTPRequest(url=url0, headers=headers1)
request102 = Test(102, 'GET product_styles.css').wrap(request102)

request103 = HTTPRequest(url=url0, headers=headers2)
request103 = Test(103, 'GET tools.css').wrap(request103)
```

```
request104 = HTTPRequest(url=url0, headers=headers2)
request104 = Test(104, 'GET colours.css').wrap(request104)
```

```
request105 = HTTPRequest(url=url0, headers=headers2)
request105 = Test(105, 'GET forms.css').wrap(request105)
```

```
# ...
```

Finally the TestRunner class. This section groups the requests into pages and defines each page as a method, sets sleep interval between requests and provides an instrumented method for the return of data from the tests:

```
class TestRunner:
    """A TestRunner instance is created for each worker thread."""

    # A method for each recorded page.
    def page1(self):
        """GET / (requests 101-121)."""
        result = request101.GET('/', (),
            ( httpUtilities.basicAuthorizationHeader('user', 'pwd'), ))
        self.token_database = \
            httpUtilities.valueFromBodyURI('database') # 'B'

        grinder.sleep(171)
        request102.GET('/css/product_styles.css', (),
            ( httpUtilities.basicAuthorizationHeader('user', 'pwd'), ))

        grinder.sleep(50)
        request103.GET('/css/tools.css', (),
            ( httpUtilities.basicAuthorizationHeader('user', 'pwd'), ))

        request104.GET('/css/colours.css', (),
            ( httpUtilities.basicAuthorizationHeader('user', 'pwd'), ))

        request105.GET('/css/forms.css', (),
            ( httpUtilities.basicAuthorizationHeader('user', 'pwd'), ))

    #.....

    return result

    def page2(self):
    #.....

    def __call__(self):
        """This method is called for every run performed by the worker
        thread."""
        self.page1()          # GET / (requests 101-121)

        grinder.sleep(100)
        self.page2()          # GET logging.html (requests 201-217)

    #.....

    def instrumentMethod(test, method_name, c=TestRunner):
        """Instrument a method with the given Test."""
        unadorned = getattr(c, method_name)
        import new
        method = new.instancemethod(test.wrap(unadorned), None, c)
        setattr(c, method_name, method)

    # Replace each method with an instrumented version.
    # You can call the unadorned method using self.page1.__target__().
    instrumentMethod(Test(100, 'Page 1'), 'page1')
    instrumentMethod(Test(200, 'Page 2'), 'page2')
```

#.....

Once you've recorded your script you have two methods that you can use to replay your script:

1. You can create a simple [grinder.properties](#) (../g3/properties.html) file and you can replay the recorded scenario with The Grinder. Your properties file should at least set `grinder.script` to `grinder.py`.
2. Alternately you can use the console to [distribute your script to an agent and set it as the script to run](#) (../g3/console.html#Script+tab) . Each agent will still need a simple [grinder.properties](#) (../g3/properties.html) file containing the console address, though you will not need to set the `grinder.script` property.

The recorded script `grinder.py` can be edited by hand to suit your needs.

4.1. Altering the output with custom stylesheet

The recommended TCPProxy HTTP filter, `-http`, modifies the output of its request and response filters by way of an XSLT stylesheet when producing the recorded script.

The standard/default stylesheet can be found in `etc/httpToJythonScript.xsl`. Replacing the standard stylesheet with one of your own making allows you to customise the output of the filter. You should pass the file name of your custom stylesheet as a command line argument directly after `-http`.

If you want to see the intermediate XML model you can use:

```
java net.grinder.TCPProxy -http etc/httpToXML.xsl -console
```

The model conforms to the XML schema `etc/tcpproxy-http.xsd`.

4.2. How to offset test numbers

It is sometimes useful to offset test numbers for a test script when running several different scripts together, perhaps using the [sequence](#) (../g3/script-gallery.html#sequence) , or [parallel](#) (../g3/script-gallery.html#parallel) examples from the script gallery. This gives the tests contributed by each script a distinct range of test numbers, which is important because the test number uniquely identifies the test in the console and the data logs.

The HTTP TCPProxy filter doesn't directly support off-setting test numbers. However, it is simple to do in the script.

Edit the recorded script to replace:

```
from net.grinder.script import Test
```

with:

```
from net.grinder.script import Test as StandardTest
```

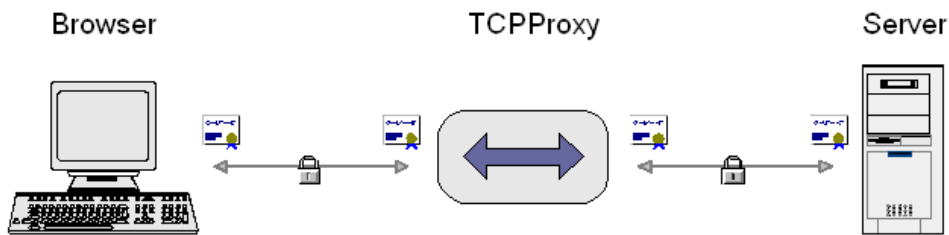
```
def Test(number, description):
    # Adjust the 1000 to the appropriate offset.
    return StandardTest(number + 1000, description)
```

This technique doesn't allow different test scripts to be merged together into one. If you want to do this, you might consider a [custom stylesheet](#); but be aware that you might also have to alter the identifiers used for headers, URLs, pages, tokens, and so on.

5. SSL and HTTPS support

The TCPProxy has SSL support based on the [JSSE](http://java.sun.com/products/jsse/) (<http://java.sun.com/products/jsse/>) . The JSSE is required by The Grinder, and is now part of Java 2 Standard Edition 1.4.1, see the [Download page](#) ([../download.html](#)) for further details.

SSL relationships are necessarily point to point. When you interpose the TCPProxy in SSL communications between a browser and a server you end up with two SSL connections. Each SSL connection has its own set of client and server certificates (both of which are optional).



The TCPProxy will negotiate appropriate certificates for both connections using built-in certificates or in a user-specified Java key store. In particular, the TCPProxy needs a self-signed server certificate for the connection from the browser. By default, the TCPProxy will use a built-in certificate.

When first establishing a connection, your browser will present a warning and confirmation dialog. This is because the built-in certificate isn't authorised by any of the certificate authorities that the browser trusts. Additionally, the built-in certificate authorises localhost so if your server doesn't listen at this address the browser will complain. Choose the "accept this certificate for this session" option.

Warning:

The Grinder deliberately accelerates SSL initialisation by using a random number generator that is seeded with a fixed number. This does not hinder SSL communication, but theoretically makes it less secure. No guarantee is made as to the cryptographic strength of any SSL communication using The Grinder.

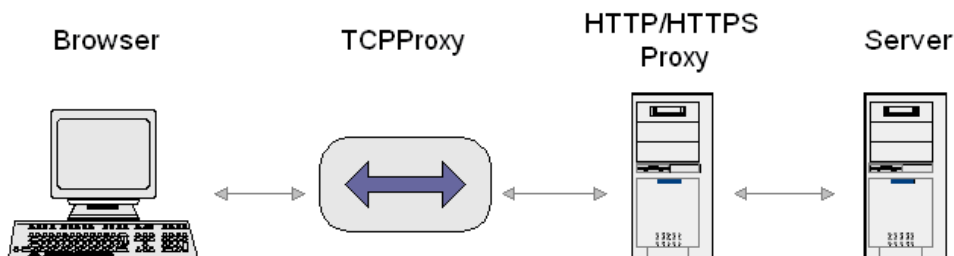
Advanced users can specify their own server certificate for the connection from the browser, or add client certificates for the connection to the server, using the `-keystore`, `-keystorepassword`, and `-keystoretype` options. See the J2SE/JSSE documentation for how to set up a key store.

If you fail to provide a key store with a valid server certificate (hard to do now The Grinder has a built in certificate), you may get a *No available certificate corresponds to the SSL cipher suites which are enabled* exception, and your browser may report that it cannot communicate as it has no common encryption algorithms. Internet Explorer likes to be different. If start the TCPProxy without a valid server certificate and then connect through it using Internet Explorer, the TCPProxy will report "SSL peer shut down incorrectly. The browser will just spin away until it times out. The easiest way to provide a server certificate is to copy the *testkeys* file from the [JSSE samples distribution](#) (<http://java.sun.com/j2se/1.4.1/docs/guide/security/jsse/samples/>) and start the proxy using:

```
java net.grinder.TCPProxy -keyStore testkeys -keyStorePassword
passphrase
```

6. Using the TCPProxy with other proxies

The TCPProxy can be used with other HTTP/HTTPS proxies.



Use the `-httpproxy` option to specify the host name and port of the proxy. Use the `-httpsproxy` option if your HTTPS proxy requires separate settings.

7. Using the TCPProxy as a port forwarder

It is normally most useful to use the TCPProxy in its HTTP Proxy mode as described above.

When using the TCPProxy as a debugging tool it occasionally is useful to use it in *port forwarding* mode. This mode is enabled when one or more of `-remotehost` and `-remoteport` are specified. In port forwarding mode, the TCPProxy simply listens on `localhost:localport` and forwards to `remotehost:remoteport`.

To understand why HTTP Proxy mode is usually better than port forwarding mode when using a browser, consider what happens if the remote server returns a page with an absolute URL link back to itself. If you click on the link, the browser will contact the server directly, bypassing the TCPProxy. Another disadvantage is that you can't use the TCPProxy with more than one remote sever.

8. Summary of TCPProxy options

Option	Description
Common	
<code>-console</code>	Display a simple console that has a control button that allows The TCPProxy to be shutdown cleanly. This can help in certain situations where a hard kill of the TCPProxy process would loose output that is still buffered in memory.
<code>-http [stylesheet]</code>	Adds the recommended request filter and a response filter to produce a script for The Grinder suitable for use with the HTTP plugin. The output can be customised by specifying the file name of an alternative XSLT style sheet.
<code>-requestfilter filter</code>	Add a request filter. <code>filter</code> can be the name of

	a class that implements <code>net.grinder.tools.tcpproxy.TCPProxyFilter</code> or one of <code>NONE</code> , <code>ECHO</code> . The option can be specified multiple times, in which case the filters are invoked one after another. If the not specified, the default <code>ECHO</code> filter is used.
<code>-responsefilter filter</code>	Add a response filter. <code>filter</code> can be the name of a class that implements <code>net.grinder.tools.tcpproxy.TCPProxyFilter</code> or one of <code>NONE</code> , <code>ECHO</code> . The option can be specified multiple times, in which case the filters are invoked one after another. If the not specified, the default <code>ECHO</code> filter is used.
<code>-localhost host</code>	Set the host name or IP address to listen on. This must correspond to an interface of the machine the TCPProxy is started on. The default is <code>localhost</code> .
<code>-localport port</code>	Set the port to listen on. The default is <code>8001</code> .
<code>-keystore file</code>	Specify a custom key store. Usually the built-in keystore is good enough so <code>-keystore</code> does not need to be specified.
<code>-keystorepassword password</code>	Set the key store password. Only used if <code>-keystore</code> is set. Optional for some key store types.
<code>-keystoretype type</code>	Set the key store type. Only used if <code>-keystore</code> is set. If not specified, the default value depends on JSSE configuration but is usually <code>jks</code> .
Additional	
<code>-properties file</code>	Specify a file containing properties that are passed on to the filters.
<code>-remotehost host</code>	Set the host name or port the TCPProxy should connect to in port forwarding mode . The TCPProxy starts in port forwarding mode if either <code>-remotehost</code> or <code>-remoteport</code> is set. The default is <code>localhost</code> .
<code>-remoteport port</code>	Set the port the TCPProxy should connect to in port forwarding mode . The TCPProxy starts in port forwarding mode if either <code>-remotehost</code> or <code>-remoteport</code> is set. The default is <code>7001</code> .
<code>-timeout seconds</code>	Set an idle timeout. This is how long the TCPProxy will wait for a request before timing out and freeing the local port. The TCPProxy will not time out if there are active connections.
<code>-httpproxy host port</code>	Specify that output should be directed through another HTTP/HTTPS proxy . This may help you reach the Internet. This option is not supported in port forwarding mode .
<code>-httpsproxy host port</code>	Specify that output should be directed through a HTTPS proxy. Overrides any <code>-httpproxy</code>

	setting. This option is not supported in port forwarding mode .
-ssl	Use SSL in port forwarding mode . This will make both the TCPProxy's local socket and the connections to the target server use SSL. The default <i>HTTP Proxy mode</i> ignores this option and always listens as an HTTP proxy and an HTTPS proxy.
-colour	Specify that a simple colour scheme should be used to distinguish request streams from response schemes. This uses terminal control codes that only work on ANSI compliant terminals.
-component class	Register a component class with the filter PicoContainer.
-debug	Make PicoContainer chatty.