SSL Support

Table of contents

The Grinder 3 supports the use of SSL by scripts. The Grinder 3 implements SSL using the Java Secure Socket Extension (JSSE) included in the Java run time. When used with the HTTP Plug-in, this is as simple as using `https` instead of `http` in URIs. Scripts can obtain a suitable `SSLContext` and hence a `SSLSocketFactory` for non-HTTP use cases, and can control the allocation of SSL sessions to worker threads.

1 Before we begin

1.1 Performance

Simulating multiple SSL sessions on a single test machine may or may not be realistic. A typical browser running on a desktop PC has the benefit of a powerful CPU to run the SSL cryptography. Be careful that your results aren't constrained due to inadequate test client CPU power.

1.2 The Grinder's SSL implementation is not secure

To reduce the client side performance overhead, The Grinder deliberately accelerates SSL initialisation by using a random number generator that is seeded with a fixed number. Further, no validation of server certificates is performed. Neither of these hinder SSL communication, but they do make it less secure.

Warning:

No guarantee is made as to the cryptographic strength of any SSL communication using The Grinder.

This acceleration affects initialisation time only and should not affect timing information obtained using The Grinder.

2 Controlling when new SSL sessions are created

By default The Grinder creates a new SSL session for each run carried out by each worker thread. This is in line with the usual convention of simulating a user session with a worker thread executing the part of the script defined by `TestRunner.__call__()`.

Alternatively, scripts may wish to have an SSL session per worker thread, i.e. for each thread to reuse SSL sessions on subsequent executions of `TestRunner.__call__()`. This can be done with the `SSLControl.setShareContextBetweenRuns()` method:

```
from net.grinder.script.Grinder import grinder
grinder.SSLControl.shareContextBetweenRuns = 1
```

This will cause each worker thread to reuse SSL sessions between runs. SSL sessions will still not be shared between worker threads. Calling `setShareContextBetweenRuns()` affects all of the worker threads.

3 Using client certificates

If a server requests or requires a client certificate, The Grinder must have some way of providing one - this involves specifying a key store.

```
from net.grinder.script.Grinder import grinder

class TestRunner:
    def __call__(self):
        grinder.SSLControl.setKeyStoreFile("mykeystore.jks", "passphrase")
```

It is only valid to use `setKeyStoreFile` from a worker thread, and it only affects that worker thread.

There is also a method called `setKeyStore` which takes a `java.io.InputStream` which may be useful if your key store doesn't live on the local file system. Both methods have an overloaded version that allows the key store type to be specified, otherwise the default type is used (normally `jks`).

Whenever `setKeyStoreFile`, `setKeyStore`, or `setKeyManagers` (see below) is called, the current SSL session for the thread is discarded. Consequently, you usually want to call these methods at the beginning of your `__call__()` method or from the `TestRunner.__init__()` constructor. Setting the thread's key store in `TestRunner.__init__()` is especially recommended if you calling `setShareContextBetweenRuns(true)` to share SSL sessions between runs.

4 FAQ

The astute reader who is familiar with key stores may have a few questions. Here's a mini FAQ:

1. *If I have several suitable certificates in my key store, how does The Grinder chose between them?*

   The Grinder relies on the JVM's default `KeyManager` implementations. This picks a certificate from the store based on SSL negotiation with the server. If there are several suitable certificates, the only way to control which is used is to provide your own `KeyManager`.
2. *`setKeyStoreFile` has a parameter for the key store password. What about the pass phrase that protects the private key in the key store?*

   The pass phrases for keys must be the same as the key store password. This is a restriction of the default `KeyManager`s. If you don't like this, you can provide your own `KeyManager`.
3. *Shouldn't I need to specify a set of certificates for trusted Certificate Authorities?*

   No. The Grinder does not validate certificates received from the server, so does not need a set of CA certificates.
4. *Can I use the properties `javax.net.ssl.keyStore`, `javax.net.ssl.keyStoreType`, and `javax.net.ssl.keyStorePassword` to specify a global keystore?*

   No. The Grinder does not use these properties, primarily because the JSSE does not provide a way to access its default SSLContext.

5 Picking a certificate from a key store [Advanced]

Here's an example script that provides its own `X509KeyManager` implementation which controls which client certificate to use. The example is hard coded to always use the certificate with the alias `myalias`.

```
from com.sun.net.ssl import KeyManagerFactory,X509KeyManager
from java.io import FileInputStream
from java.security import KeyStore
from jarray import array

class MyManager(X509KeyManager):
    def __init__(self, keyStoreFile, keyStorePassword):
        keyStore = KeyStore.getInstance("jks")
        keyStore.load(FileInputStream(keyStoreFile), keyStorePassword)

        keyManagerFactory = \
         KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm())
        keyManagerFactory.init(keyStore, keyStorePassword)

        # Assume we have one key manager.
        self._delegate = keyManagerFactory.keyManagers[0]

    def __getattr__(self, a):
        """Some Python magic to pass on all invocations of methods we
        don't define on to our delegate."""

        if self.__dict__.has_key(a): return self.__dict__[a]
        else: return getattr(self._delegate, a)

    def chooseClientAlias(self, keyTypes, issuers):
        return "myalias"

myManager = MyManager("keystore.jks", "password")
myManagerArray = array((myManager,), X509KeyManager)

class TestRunner:
    def __call__(self):
        grinder.SSLControl.setKeyManagers(myManagerArray)
        # ...
```

6 Debugging

When debugging SSL interactions, you may find it useful to set the following in `grinder.properties`.

```
grinder.jvm.arguments=-Djavax.net.debug=ssl
# or -Djavax.net.debug=all
```