# **Spark Security**

Spark currently supports authentication via a shared secret. Authentication can be configured to be on via the spark.authenticate configuration parameter. This parameter controls whether the Spark communication protocols do authentication using the shared secret. This authentication is a basic handshake to make sure both sides have the same shared secret and are allowed to communicate. If the shared secret is not identical they will not be allowed to communicate. The shared secret is created as follows:

* For Spark on [YARN](http://spark.apache.org/docs/latest/running-on-yarn.html) deployments, configuring spark.authenticate to true will automatically handle generating and distributing the shared secret. Each application will use a unique shared secret.
* For other types of Spark deployments, the Spark parameter spark.authenticate.secret should be configured on each of the nodes. This secret will be used by all the Master/Workers and applications.

## **Web UI**

The Spark UI can also be secured by using [javax servlet filters](http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html) via the spark.ui.filters setting. A user may want to secure the UI if it has data that other users should not be allowed to see. The javax servlet filter specified by the user can authenticate the user and then once the user is logged in, Spark can compare that user versus the view ACLs to make sure they are authorized to view the UI. The configs spark.acls.enableand spark.ui.view.acls control the behavior of the ACLs. Note that the user who started the application always has view access to the UI. On YARN, the Spark UI uses the standard YARN web application proxy mechanism and will authenticate via any installed Hadoop filters.

Spark also supports modify ACLs to control who has access to modify a running Spark application. This includes things like killing the application or a task. This is controlled by the configs spark.acls.enable and spark.modify.acls. Note that if you are authenticating the web UI, in order to use the kill button on the web UI it might be necessary to add the users in the modify acls to the view acls also. On YARN, the modify acls are passed in and control who has modify access via YARN interfaces.

Spark allows for a set of administrators to be specified in the acls who always have view and modify permissions to all the applications. is controlled by the config spark.admin.acls. This is useful on a shared cluster where you might have administrators or support staff who help users debug applications.

## **Event Logging**

If your applications are using event logging, the directory where the event logs go (spark.eventLog.dir) should be manually created and have the proper permissions set on it. If you want those log files secured, the permissions should be set to drwxrwxrwxt for that directory. The owner of the directory should be the super user who is running the history server and the group permissions should be restricted to super user group. This will allow all users to write to the directory but will prevent unprivileged users from removing or renaming a file unless they own the file or directory. The event log files will be created by Spark with permissions such that only the user and group have read and write access.

## **Encryption**

Spark supports SSL for Akka and HTTP (for broadcast and file server) protocols. SASL encryption is supported for the block transfer service. Encryption is not yet supported for the WebUI.

Encryption is not yet supported for data stored by Spark in temporary local storage, such as shuffle files, cached data, and other application files. If encrypting this data is desired, a workaround is to configure your cluster manager to store application data on encrypted disks.

### **SSL Configuration**

Configuration for SSL is organized hierarchically. The user can configure the default SSL settings which will be used for all the supported communication protocols unless they are overwritten by protocol-specific settings. This way the user can easily provide the common settings for all the protocols without disabling the ability to configure each one individually. The common SSL settings are at spark.ssl namespace in Spark configuration, while Akka SSL configuration is at spark.ssl.akka and HTTP for broadcast and file server SSL configuration is at spark.ssl.fs. The full breakdown can be found on the [configuration page](http://spark.apache.org/docs/latest/configuration.html).

SSL must be configured on each node and configured for each component involved in communication using the particular protocol.

### **YARN mode**

The key-store can be prepared on the client side and then distributed and used by the executors as the part of the application. It is possible because the user is able to deploy files before the application is started in YARN by using spark.yarn.dist.files or spark.yarn.dist.archivesconfiguration settings. The responsibility for encryption of transferring these files is on YARN side and has nothing to do with Spark.

For long-running apps like Spark Streaming apps to be able to write to HDFS, it is possible to pass a principal and keytab to spark-submit via the--principal and --keytab parameters respectively. The keytab passed in will be copied over to the machine running the Application Master via the Hadoop Distributed Cache (securely - if YARN is configured with SSL and HDFS encryption is enabled). The Kerberos login will be periodically renewed using this principal and keytab and the delegation tokens required for HDFS will be generated periodically so the application can continue writing to HDFS.

### **Standalone mode**

The user needs to provide key-stores and configuration options for master and workers. They have to be set by attaching appropriate Java system properties in SPARK\_MASTER\_OPTS and in SPARK\_WORKER\_OPTS environment variables, or just in SPARK\_DAEMON\_JAVA\_OPTS. In this mode, the user may allow the executors to use the SSL settings inherited from the worker which spawned that executor. It can be accomplished by settingspark.ssl.useNodeLocalConf to true. If that parameter is set, the settings provided by user on the client side, are not used by the executors.

### **Preparing the key-stores**

Key-stores can be generated by keytool program. The reference documentation for this tool is [here](https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html). The most basic steps to configure the key-stores and the trust-store for the standalone deployment mode is as follows: \* Generate a keys pair for each node \* Export the public key of the key pair to a file on each node \* Import all exported public keys into a single trust-store \* Distribute the trust-store over the nodes

### **Configuring SASL Encryption**

SASL encryption is currently supported for the block transfer service when authentication (spark.authenticate) is enabled. To enable SASL encryption for an application, set spark.authenticate.enableSaslEncryption to true in the application’s configuration.

When using an external shuffle service, it’s possible to disable unencrypted connections by setting spark.network.sasl.serverAlwaysEncrypt totrue in the shuffle service’s configuration. If that option is enabled, applications that are not set up to use SASL encryption will fail to connect to the shuffle service.

## **Configuring Ports for Network Security**

Spark makes heavy use of the network, and some environments have strict requirements for using tight firewall settings. Below are the primary ports that Spark uses for its communication and how to configure those ports.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **From** | **To** | **Default Port** | **Purpose** | **Configuration Setting** | **Notes** |
| Browser | Standalone Master | 8080 | Web UI | spark.master.ui.port / SPARK\_MASTER\_WEBUI\_PORT | Jetty-based. Standalone mode only. |
| Browser | Standalone Worker | 8081 | Web UI | spark.worker.ui.port / SPARK\_WORKER\_WEBUI\_PORT | Jetty-based. Standalone mode only. |
| Driver / Standalone Worker | Standalone Master | 7077 | Submit job to cluster / Join cluster | SPARK\_MASTER\_PORT | Akka-based. Set to "0" to choose a port randomly. Standalone mode only. |
| Standalone Master | Standalone Worker | (random) | Schedule executors | SPARK\_WORKER\_PORT | Akka-based. Set to "0" to choose a port randomly. Standalone mode only. |

### **Standalone mode only**

### **All cluster managers**

| **From** | **To** | **Default Port** | **Purpose** | **Configuration Setting** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| Browser | Application | 4040 | Web UI | spark.ui.port | Jetty-based |
| Browser | History Server | 18080 | Web UI | spark.history.ui.port | Jetty-based |
| Executor / Standalone Master | Driver | (random) | Connect to application / Notify executor state changes | spark.driver.port | Akka-based. Set to "0" to choose a port randomly. |
| Driver | Executor | (random) | Schedule tasks | spark.executor.port | Akka-based. Set to "0" to choose a port randomly. |
| Executor | Driver | (random) | File server for files and jars | spark.fileserver.port | Jetty-based |
| Executor | Driver | (random) | HTTP Broadcast | spark.broadcast.port | Jetty-based. Not used by TorrentBroadcast, which sends data through the block manager instead. |
| Executor | Driver | (random) | Class file server | spark.replClassServer.port | Jetty-based. Only used in Spark shells. |
| Executor / Driver | Executor / Driver | (random) | Block Manager port | spark.blockManager.port | Raw socket via ServerSocketChannel |

See the [configuration page](http://spark.apache.org/docs/latest/configuration.html) for more details on the security configuration parameters, and [org.apache.spark.SecurityManager](https://github.com/apache/spark/tree/master/core/src/main/scala/org/apache/spark/SecurityManager.scala) for implementation details about security.