FIRST Server Documentation

Release 0.1 BETA

Angel M. Villegas

Apr 23, 2018

Server Documentation

A public, freely available server is located at first-plugin.us. The below information goes into how to stand up your own FIRST server. Keep in mind the current authorization mechanism is OAuth2 from Google. This can be expanded to include other OAuth2 services, however, it is important to keep in mind that OAuth2 authorization requires a developer/app/project specific data to enable. Furthermore, Google's OAuth will only redirect to localhost or a domain name.

CHAPTER 1

Installing

Note: To quickly install FIRST, we use Docker. Install Docker before following the below instructions.

Installing your own FIRST server can be quick and easy with an Ubuntu machine and docker. The below instructions will use Docker to install FIRST, its dependencies, configure Apache, and create self signed certs. This is more of a production type build, if you wish to install FIRST in a developer environment then you might want to leverage Django's development server (scroll down for instructions). To install, enter the below commands into a shell.

Important: After cloning the Git repo

Save your google auth json information to install/google_secret.json. To generate a google_secret.json file you will need to go to https://console.developers.google.com, create a project, select the project, select Credentials in the left set of links under APIs & services. Once selected, select the Create credentials drop down menu and click OAuth client ID. Select Web application, and fill out the details. Set the Authorized redirect URIs to your server name with */oauth/google*

Examples

```
http://localhost:8888/oauth/google
http://first.talosintelligence.com/oauth/google
```

Once created you will have the option to down the JSON file containing the generated secret. Optionally, you can add install/ssl/apache.crt and apache.key file if you have an SSL certificate you would prefer to use.

```
$ apt-get install docker
$ git clone https://github.com/vrtadmin/FIRST-server.git
$ cd FIRST-server
$ docker-compose -p first up -d
```

When the FIRST server is installed, no engines are installed. FIRST comes with three Engines: `ExactMatch`, `MnemonicHashing`, and `BasicMasking`. Enable to engines you want active by using the `utilities/engine_shell.py` script.

Note: Before engines can be installed, the developer must be registered with the system. This can be accomplished through the web UI if OAuth has been setup or manually by the user_shell.py located in the utilities folder.

```
$ cd FIRST-server/server/utilities
$ python user_shell.py adduser <user_handle: johndoe#0001> <user email: john@doe.com>
```

Ensure the developer is registered before progressing.

Python script `engine_shell.py` can be provided with command line arguments or used as a shell. To quickly install the three available engines run the below commands:

```
$ cd FIRST-server/server/utilities
$ python engine_shell.py install first_core.engines.exact_match ExactMatchEngine
↪<developer_email>
$ python engine_shell.py install first_core.engines.mnemonic_hash MnemonicHashEngine
↪<developer_email>
$ python engine_shell.py install first_core.engines.basic_masking BasicMaskingEngine
↪<developer_email>
```

Once an engine is installed you can start using your FIRST installation to add and/or query for annotations. Without engines FIRST will still be able to store annotations, but will never return any results for query operations.

Attention: Manually installing FIRST

FIRST can be installed manually without Docker, however, this will require a little more work. Look at docker's `install/requirements.txt` and install all dependencies. Afterwards, install the engines you want active (see above for quick engine installation) and run:

```
$ cd FIRST-server/server
$ python manage.py runserver 0.0.0.0:1337
```

1.1 RESTful API

All RESTful APIs are available to users with active and valid API keys. To get register and get an API key see registering. All below URLs make the assumption the FIRST server is located at FIRST_HOST. If using the public FIRST server this will be http://first-plugin-us.

All RESTful APIs require a vaild API key in the URL. For example if you were trying to test your connection to FIRST, you would perform a GET request to FIRST_HOST/api/test_connection.

```
import requests

response = requests.get('FIRST_HOST/api/test_connection/00000000-0000-0000-0000-
↪000000000000'})
```

```
> curl FIRST_HOST/api/test_connection/00000000-0000-0000-0000-000000000000
```

An HTTP 401 is returned if a valid api_key is not provided as a GET variable.

1.1.1 Test Connection

Used to test and ensure the FIRST client can connect to, validate the API key, and received the expected response.

Client Request

METHOD	URL	Params
GET	/api/test_connection/<api_key>	**api_key**: user's API key

Server Response

```
{"status" : "connected"}
```

1.1.2 Plugin Version Check

Used to check if the client is using the latest version of FIRST.

Danger: TODO: Implement and document [currently just planning]

Client Request

METHOD	URL	Params
GET	/api/plugin/check	**api_key**: user's API key
		type: client type
		v: version information

Param **type**

idapython	Hex Ray's IDA Pro plugin
python	Python module
radare	Radare plugin
viper	Viper plugin

1.1.3 Get Architectures

An HTTP 401 is returned if a valid api_key is not provided as a GET variable.

Client Request

METHOD	URL	Params
GET	/api/sample/architectures/<api_key>	**api_key**: user's API key

Server Response:

```
# Successful
{"failed" : false, "architectures" : ['intel32', 'intel64', 'arm', 'arm64', 'mips',
↪'ppc', 'sparc', 'sysz', ...]}

# Failed - Error
{"failed" : true, "msg" : <String>}
```

1.1.4 Sample Checking

An HTTP 401 is returned if a valid api_key is not provided as a GET variable.

Client Request

METHOD	URL	Params
POST	/api/sample/checkin/<api_key>	**api_key**: user's API key

```
{
   #   Required
   'md5' : /^[a-fA-F\d]{32}$/,
   'crc32' : <32 bit int>,

   #   Optional
   'sha1': /^[a-fA-F\d]{40}$/,
   'sha256': /^[a-fA-F\d]{64}$/
}
```

Server Response:

```
# Successful
{"failed" : false, "checkin" : true}

# Successful -
{"failed" : false, "checkin" : false}

# Failed - Error
{"failed" : true, "msg" : <String>}
```

Failure Strings	Description
Sample info not provided	MD5/CRC32 not provided
MD5 is not valid	MD5 should be 32 hex characters
CRC32 value is not an integer	Integer value is required for the CRC32
Unable to connect to FIRST DB	Connection could not be established

1.1.5 Upload Metadata

Client Request

METHOD	URL	Params
POST	/api/metadata/add/<api_key>	**api_key**: user's API key

```
{
   'md5' : /^[a-fA-F\d]{32}$/,
   'crc32' : <32 bit int>,

   'functions' : Dictionary of json-ed Dictionaries (max_length = 20)
   {
      'client_id' :
      {
         'opcodes' : String (base64 encoded)
         'architecture' : String (max_length = 64)
```

```
        'name' : String (max_length = 128)
        'prototype' : String (max_length = 256)
        'comment' : String (max_length = 512)

        'apis' : List of Strings (max_string_length = 64)

        #   Optional
        'id' : String
    }
  }
}
```

Server Response

1.1.6 Get Metadata History

Client Request

METHOD	URL	Params
POST	/api/metadata/history/<api_key>	**api_key**: user's API key

```
{
  'metadata' : List of Metadata IDs (max_length = 20)
               [<metadata_id>, ... ]
}
```

Server Response

```
{
  'failed': False,
  'results' : Dictionary of dictionaries
  {
    'metadata_id' : Dictionary
     {
        'creator' : String (max_length = 37) (/^[\s\d_]{1,32}#\d{4}$/)
        'history : List of dictionaries
           [{
               'name' : String (max_length = 128)
               'prototype' : String (max_length = 256)
               'comment' : String (max_length = 512)
               'committed' : Datetime
           }, ...]
    }
  }
}
```

1.1.7 Apply Metadata

Client Request

METHOD	URL	Params
POST	/api/metadata/applied/<api_key>	**api_key**: user's API key

```
{
   'md5' : /^[a-fA-F\d]{32}$/
   'crc32' : <32 bit int>

   'id' : /^[\da-f]{24}$/
}
```

Server Response

1.1.8 Unapply Metadata

Client Request

METHOD	URL	Params
POST	/api/metadata/unapplied/<api_key>	**api_key**: user's API key

```
{
   'md5' : /^[a-fA-F\d]{32}$/
   'crc32' : <32 bit int>

   'id' : /^[\da-f]{24}$/
}
```

Server Response

1.1.9 Get Metadata

Client Request

METHOD	URL	Params
POST	/api/metadata/get/<api_key>	**api_key**: user's API key

```
{
  'metadata' : List of Metadata IDs (max_length = 20)
          [<metadata_id>, ... ]
}
```

Server Response

1.1.10 Delete Metadata

Client Request

METHOD	URL	Params
GET	/api/metadata/delete/<api_key>/<id>	**api_key**: user's API key
		id: metadata id

Server Response

1.1.11 Get Metadata Created

Client Request

METHOD	URL	Params
GET	/api/metadata/created/<api_key>	**api_key**: user's API key
GET	/api/metadata/created/<api_key>/<page>	**api_key**: user's API key **page**: page to grab

Server Response

```
{
   'failed': False,
   'page' : Integer (current page requested,
   'pages' : Integer (total number of pages)
   'results' : Dictionary of dictionaries
   {
     'metadata_id' : Dictionary
      {
         'name' : String (max_length = 128)
         'prototype' : String (max_length = 256)
         'comment' : String (max_length = 512)
         'rank' : Integer
         'id' : String (length = 24)
      }
   }
}
```

1.1.12 Scan for Similar Functions

Client Request

METHOD	URL	Params
POST	/api/metadata/scan/<api_key>	**api_key**: user's API key

```
{
   'functions' : Dictionary of json-ed Dictionaries (max_length = 20)
   {
     'client_id' :
      {
         'opcodes' : String (base64 encoded)
         'architecture' : String (max_length = 64)
         'apis' : List Strings
      }
   }
}
```

Server Response

1.2 Engines

1.2.1 Engine Shell

The Python script `engine_shell.py` provides you with some functionality to manage engines installed into FIRST. Below you will see the script's operations.

```
+=========================================================+
|                  FIRST Engine Shell Menu                |
+=========================================================+
| list      | List all engines currently installed        |
| info      | Get info on an engine                       |
| install   | Installs engine                             |
| delete    | Removes engine record but not other DB data |
| enable    | Enable engine (Engine will be enabled)      |
| populate  | Sending all functions to engine             |
| disable   | Disable engine (Engine will be disabled)    |
+---------------------------------------------------------+
```

1.2.2 Testing Engines

TODO

1.3 DB Index