Cyber Security Scoring Engine Framework Documentation

Release

Brahm Lower

Jan 23, 2018

Overview

Danger: This project is no longer under development. There won't be any further updates to the project, however the repo will remain available for future reference. Since the project is being formally discontinued, I am relicensing it as MIT. If you have any questions, please feel free to email me.

CHAPTER 1

Introduction

Danger: This project is no longer under development. There won't be any further updates to the project, however the repo will remain available for future reference. Since the project is being formally discontinued, I am relicensing it as MIT. If you have any questions, please feel free to email me.

The Cyber Security Scoring Engine Framework (CSSEF) is an easy to use framework for hosting security competitions. The primary purpose of the framework has been to make scoring security competitions as simple as possible, so that more time and energy may be spent setting up the competition environment itself. Many of the features and requirements were determined by the competitions hosted inhouse by the University of Alaska's Cyber Security Club, which attended the National Cyber Collegdet Defense Competition several years in a row. While initial development focused on providing utilities for CCDC-like competitions, the project expanded to facilitate other types of competitions as well, such as capture the flag events.

1.1 Features

Features include but are certainly not limited to the following. Please see the respective section of documentation for more inforation on any particular feature.

- Multiple organization
- Password and token based authentication
- Easy to use command line interface
- Modern(ish) web client
- Plugin interface for multiple types of competitions
- A prebuilt plugin for CCDC-like competitions (see the plugins section)

1.2 Framework Components

1.2.1 Server

This is where the bulk of the framework lives. The server provides facilities to host various types of security related competitions. Those facilities are consumed by plugins that use then to build some form of competition. This means you can host a capture the flag competition and a CCDC-like competition on the same service. The server communicates with the other clients via HTTP RPC calls, and uses sqlalchemy for databasing. For additional information, see the server documentation.

1.2.2 Client

The client package provides endpoints for client applications (in the event you want to write your own), as well as a command line tool. If you plan to install the web client, this package will be a required depenancy. See the client documentation for more information, or the command line documentation for cli reference.

1.2.3 Web Client

The web client is meant to be an easy and painless tool for interacting with the server, as well as the various features provided by competition plugins- especially when competitors may be submitting data during competition environments that don't actually have the cssef client available. The web client is currently using django, which itself simply consumes endpoints within the cssef client. For more information about the web client, see the cssef webui documentation.

CHAPTER 2

Getting Started

This short guide will walk through the process of setting up a basic installation of the CSSEF server. This includes installing and configuring the dependancies, and verifying that the client is able to communicate properly with the server.

2.1 Server Installation

The server requires systemd, python and pip.

Install the prerequisets

```
user@debian:~$ sudo apt-get install -y git python-pip python-dev systemd libsystemd-
↪dev
```

Install the CSSEF server

```
user@debian:~$ git clone https://github.com/bplower/cssef.git
user@debian:~$ cd cssef/CssefServer
user@debian:~/cssef/CssefServer$ sudo make install
```

Verify the installation was successful

```
user@debian:~/cssef/CssefServer$ sudo systemctl is-enabled cssef-server.service
enabled
user@debian:~/cssef/CssefServer$ sudo systemctl start cssef-server.service
user@debian:~/cssef/CssefServer$ sudo systemctl status cssef-server.service | grep␣
↪Active:
Active: active (running) since Thu 2016-09-01 22:00:49 AKDT; 6s ago
```

2.2 Client Installation

The client requires python and pip.

If you don't have the repo cloned yet, clone it.

```
user@debian:~$ git clone https://github.com/bplower/cssef.git
```

Move to the client package directory and install the python package and execuables via make. The required pip dependancies are `prettytable`, `jsonrpcclient`, and `PyYAML`.

```
user@debian:~$ cd cssef/CssefClient
user@debian:~/cssef/CssefClient$ sudo make install
```

2.3 Admin Configuration

Initially, there are no users or organization, so we will have to make them by using an administrator token. Set the `admin-token` in the server configuration field to a secure passphrase to use for the initial configuration. Note that this will be provided in plain text several times, but will be revoked after the configuration is complete.

Here we're creating the token and assigning it to an environment variable. We're changing the configuration file via sed, however you may use the text editor of your choice. Afterward, restart the server to apply the configuration changes.

```
user@debian:~$ admintoken=`openssl rand -hex 16`
user@debian:~$ sudo sed -i "s|admin-token:|admin-token: $admintoken|" /etc/cssef/
↪cssef-server.yml
user@debian:~$ sudo systemctl restart cssef-server.service
```

We can now provide the admintoken while executing commands. This will allow us to bypass authentication checks and limits. First create an administrator organization for the admin account to exist within.

```
user@debian:~$ cssef-cli --admin-token $admintoken organization add --
↪name=Administrators --maxMembers=10
+------------+-------------+----------------+------+----+-----------------+-----------
↪------------+----------------+--------------------+-------------+-----------+
| maxMembers | description |      name      | url  | id | maxCompetitions |␣
↪canDeleteCompetitions | canDeleteUsers | canAddCompetitions | canAddUsers |␣
↪deletable |
+------------+-------------+----------------+------+----+-----------------+-----------
↪------------+----------------+--------------------+-------------+-----------+
|     10     |     None    | Administrators | None | 1  |       None      |          ␣
↪True         |      True      |        True        |     True    |    True   |
+------------+-------------+----------------+------+----+-----------------+-----------
↪------------+----------------+--------------------+-------------+-----------+
```

And now, create an administrator user to use.

```
user@debian:~$ cssef-cli --admin-token $admintoken user add --organization=1 --
↪name=Admin --username=admin --password=admin
+----------+-------------+--------------+-------------------------------------------
↪-----------------+----+-------+
| username | description | organization |                                password
↪                  | id |  name |
+----------+-------------+--------------+-------------------------------------------
↪-----------------+----+-------+
|  admin   |     None    |      1       | $2b$10
↪$cHzkaFpT3va5LoTyjV4gHuxd3MZQpvm5OUQCGcSiwbxYmsI74j9a6 | 1  | Admin |
+----------+-------------+--------------+-------------------------------------------
↪-----------------+----+-------+
```

Disable admintoken access by removing the admin token from the server configuration file.

```
user@debian:~$ sudo sed -i "s|admin-token:*|admin-token:|" /etc/cssef/cssef-server.yml
user@debian:~$ sudo systemctl restart cssef-server.service
```

This can be verified by attempting to list availble users using the admin token we used. At this point in time, the server doesn't explicitly deny the use of the admin-token, so it will continue to attempt to authenticate the user as normal. Since we haven't provided a valid username or password, access is not granted.

```
user@debian:~$ cssef-cli --admin-token $admintoken user get
An error was encountered:
No username provided.
```

Lastly, we can verify that authorization is working by logging in. Here we are not specifying the password, so we are prompted for it. Since the authentication is successful, we received an authentication token, which will be automatically provided by the client in future requests.

```
user@debian:~$ cssef-cli login --username admin
Password:
Authentication was successful.
```

Now that we've been authenticated, we can list the available users.

```
user@debian:~$ cssef-cli user get
+----------+-------------+--------------+-------------------------------------------
↪-----------------+----+-------+
| username | description | organization |                                password
↪                 | id |  name |
+----------+-------------+--------------+-------------------------------------------
↪-----------------+----+-------+
|  admin   |     None    |       1      | $2b$10
↪$cHzkaFpT3va5LoTyjV4gHuxd3MZQpvm5OUQCGcSiwbxYmsI74j9a6 | 1  | Admin |
+----------+-------------+--------------+-------------------------------------------
↪-----------------+----+-------+
```

CHAPTER 3

CSSEF Server

3.1 Server Installation

The server requires systemd, python and pip.

Install the prerequisets

```
user@debian:~$ sudo apt-get install -y git python-pip python-dev systemd libsystemd-
↪dev
```

Install the CSSEF server

```
user@debian:~$ git clone https://github.com/bplower/cssef.git
user@debian:~$ cd cssef/CssefServer
user@debian:~/cssef/CssefServer$ sudo make install
```

Verify the installation was successful

```
user@debian:~/cssef/CssefServer$ sudo systemctl is-enabled cssef-server.service
enabled
user@debian:~/cssef/CssefServer$ sudo systemctl start cssef-server.service
user@debian:~/cssef/CssefServer$ sudo systemctl status cssef-server.service | grep␣
↪Active:
Active: active (running) since Thu 2016-09-01 22:00:49 AKDT; 6s ago
```

3.2 Server Configuration

Configurations can be loaded from several different sources, where values loaded later will overwrite previously set values. The order configuration values are loaded is as follows:

1. Default (hard coded)
2. Global config file

3. Command line configs

Please consider the following example:

> *The default client cache time for available endpoints is 24 hours, but is overwritten to a value of 12 hours in the global config file. However, you are troublshooting something related to plugins, so you've started the daemon with the cache time value set to 0 meaning the cache will be refreshed on each client request.*

The value of the client cache time was overwritten twice in this example: once by the global configuration, and once by the value provided on the command line. Any configuration option may be set through any of the configuration sources (excluding the default configs for obvious reasons).

3.2.1 Available Options

admin-token This should only be used for initial setup, but may be used in the event you are locked out of administrator accounts. The client may provide the token to authorize requests by completely bypassing username and password checks. If you are not actively using this, the value should be left blank, meaning admin-token auth is disabled.

Default:

Example config file

```
# Setting a weak admin token for initial setup
admin-token: abc123def456
```

Example command line

```
user@debian:~$ cssefd start --admin-token abc123def456
```

database-path While using sqlite as the backend database, this option will be for the absolute path to store the database file at.

Default: `/var/opt/cssef/db.sqlite3`

Example config file

```
# Hold the database in memory for performance while testing
database-path: ''
```

Example command line

```
user@debian:~$ cssefd start --database-path ''
```

database-table-prefix This value will be the prefix for every table in the database. Depending on your database backend, this may not be as important. The default will result in tables that look similar to "cssef_users".

> **Attention:** This feature is broken as of commit 993d87e. The prefix is hardcoded to "cssef" for the time being.

Default: `cssef`

Example config file

```
# Table prefix for production cssef installation
database-table-prefix: cssef-prod
```

Example command line

```
user@debian:~$ cssefd start --database-table-prefix cssef-prod
```

logging I've completely skipped the logging values because they're all basically useless right now...

installed-plugins This is a list of plugins that conform to the CSSEF plugin model that should be imported. Those plugins must already be installed, and the entries in this list must be the names of the modules.

Default:

Example config file

```
# Include the default CCDC like competition and CTF competition
installed-plugins:
- cssef-cdc
- cssef-ctf
```

Example command line

```
user@debian:~$ cssefd start --installed-plugins cssef-ccdc,cssef-ctf
```

CHAPTER 4

CSSEF Client

4.1 Client Installation

The client requires python and pip.

If you don't have the repo cloned yet, clone it.

```
user@debian:~$ git clone https://github.com/bplower/cssef.git
```

Move to the client package directory and install the python package and execuables via make. The required pip dependancies are `prettytable`, `jsonrpcclient`, and `PyYAML`.

```
user@debian:~$ cd cssef/CssefClient
user@debian:~/cssef/CssefClient$ sudo make install
```

4.2 Client Configuration

4.2.1 General

verbose This is by default false, but when set to true, will allow additional output to be printed detailing events and actions that are happening.

Default: `False`

Example config file

```
# I want to know EXACTLY what the client is doing all the time!
verbose: True
```

Example command line

```
user@debian:~$ cssef-cli --verbose organization get
```

task-timeout The time in seconds to wait for a task to be completed. This is in case the server is not running, or has crashed while handling your request.

Default: 5

Example config file

```
# My server is super fast so I should never have to wait.
task-timeout: 1
```

Example command line

```
user@debian:~$ cssef-cli --task-timeout 30 organization get
```

4.2.2 Server Connection

rpc-hostname This is the hostname or IP address for the CSSEF server.

Default: localhost

Example config file

```
# The CSSEF server for the practice competition
rpc-hostname: 10.0.0.50
```

Example command line

```
user@debian:~$ cssef-cli --rpc-hostname cssef.example.com login
```

rpc-port This is the port the CSSEF server is using on the remote host.

Default: 5000

Example config file

```
# Running the service on a non-standard port
rpc-port: 9001
```

Example command line

```
user@debian:~$ cssef-cli --rpc-port 1234 login
```

4.2.3 Authentication

organization This is the organization you belong to. At this stage of development, the value is the ID of the organization, but this will eventually be updated to be the organizations name.

Defaut:

Example config file

```
# Setting the organization so that we don't have to provide it each
# time we authenticate
organization: 1
```

Example command line

```
user@debian:~$ cssef-cli --organization 1 organization get
```

username This is the username for your account.

Default:

Example config file:

```
# I'm getting sick of reintroducing myself all the time
username: admin
```

Example command line

```
user@debian:~$ cssef-cli --username admin organization get
```

password The password for your account. If you do not provide your password in a situation where it is required (assuming you provide the rest of your credentials), you will be prompted for your password. This is exemplified in the command line examples section.

> **Warning:** It is an extremely bad idea to leave your password in plain text in a file. Please don't set this in a configuration file.

Default:

Example config file:

```
# I make very bad decisions in life. This is one of them.
password: mypassword
```

Example command line

```
user@debian:~$ cssef-cli --password mypassword organization get
...
user@debian:~$ cssef-cli organization get
Password:
```

4.2.4 Token

token-auth-enabled This simply enables or disables the token authentication system. Setting this to 'False' makes the login command useless since the login command is only used to retrieve an authentication token.

Default: True

Example config file

```
# I was once bullied by tokens in school, so I don't want them on my
# client at all. This will disable token authentication.
token-auth-enabled: False
```

Example command line

```
user@debian:~$ cssef-cli --token-auth-enabled false organization get
```

token-file This is the file to store your current token in. This is a configuration you will most often set within your local configuration file, since this tells the client where to find your token file.

Default: ~/.cssef/token

Example config file

```
# I don't like file names less than two words in length, so I'm
# renaming the token file
token-file: ~/.cssef/auth-token-file
```

Example command line

```
user@debian:~$ cssef-cli --token-file ~/.cssef/tmp-token login
```

token-renewal-enabled Most tokens have expirations. When you log in, your token will expire after some period of time, after which you will have to login again. Token renewal will request a new token each time you execute a command. If the token expiration time is 'T', this means you won't have to log in again unless it has been T time since you last executed a cssef-cli request.

4.2.5 Endpoint Caching

endpoint-cache-enabled The client gets a list of available commands the server provides. This allows the server to add and remove plugins (thus changing the available commands) without requiring the client to install or uninstall additional components. Endpoint caching lets the client retain that list of endpoints so that it doesn't have to ask the server for it each time.

Default: True

Example config file

```
# I'm a bleeding edge kind of guy- I have to make sure I have the
# updated list as soon as it's availble, therefore I've disabled
# endpoint caching.
endpoint-cache-enabled: False
```

Example command line

```
user@debian:~$ cssef-cli --endpoint-cache-enabled False organization get
```

force-endpoint-cache In some cases, you may want to force the the client to use the cached endpoint data. If you already had cached data and decided that you never wanted to check available endpoints again, you could set this a configuration file- but that is not recommended.

Default: False

Example config file

```
# I will only ever be using the core endpoints, which I already have cached, so I␣
↪don't want to check updated endpoint EVER.
force-endpoint-cache: True
```

Example command line

```
user@debian:~$ cssef-cli --force-endpoint-cache True organization get
```

force-endpoint-server In some cases, you may want to force the client to check the server for available endpoints. It is rather senseless to set this in a configuration file, since that would effectively act the same as setting `enpoint-cache-enabled: False`.

Default: False

Example config file

```
# I'm not a rationable human, so I want endpoint caching enabled, but I never␣
↪want to use my cached copy of the data.
force-enpoint-server: True
```

Example command line

```
user@debian:~$ cssef-cli --force-endpoint-server True organization get
```

endpoint-cache-file This is the path to the file to cache the available endpoint data.

Default: ~/.cssef/endpoint-cache

Example config file

```
# I have a super secret hiding place for special data like this
endpoint-cache-file: /dev/null
```

Example command line

```
user@debian:~$ cssef-cli --endpoint-cache-file ~/.caches/cssef_endpoint-cache␣
↪organization get
```

endpoint-cache-time This is the maximum amount of time that may pass before the client will check for available endpoints. This is based on the last time the file specified by `endpoint-cache-file` was modified. You can see when a file was last modified by using stat. There isn't much point to specifying this via command line, unless to induce the same functionality as `force-endpoint-server`.

If an integer with no metric is provided, it will be assumed to be seconds. For simplicity, you may provide metrics for seconds, minutes, hours, and days using one of the following:

- The first letter of the metric (example: 'd' for days)
- The singlular of the metric (example: 'hour')
- The plurl of the metric (example: 'minutes')

Default: 12h

Example config file

```
# My server is pretty fluid, and gets new/different plugins quite often, and I␣
↪want to be sure I get those updates in a reasonable amount of time.
endpoint-cache-time: 5minutes
```

Example command line

```
user@debian:~$ cssef-cli --endpoint-cache-time 5s organization get
```

CHAPTER 5

CSSEF Web Client

Download the AdminLTE resources:

```
wget https://almsaeedstudio.com/download/AdminLTE-master
```

Unzip that file and move the bootstrap, dist, and plugins folders to `cssef/WebInterface/WebInterface/static/`.

CHAPTER 6

Overview

The CSSEF repository is broken into several distinct components, each of which is represented by directories at the root of the repository. The following is a brief overview of each component. Continue on for more information about each section.

CSSEF Server (CssefServer/) This is where the server side service and library lives. Like the client, all necessary resources for the server pip package are contained here.

CSSEF Client (CssefClient/) This is where the client application and library lives. This includes all resources needed for the pip package.

CSSEF Web Client (WebInterface/) This is a web client that depends on the client library to function. It serves the same purpose as the CLI client, but through a web interface. At this time, there are no package resources available.

Documentation (docs/) This directory contains all documentation resouces for the project.

Competition Plugins (plugins/) This directory is for the development of competition plugins to use with the CSSEF. At this time, it only houses code for the cssefcdc, which is largely composed of old code from the previous itterations of the scoring engine framework.

CHAPTER 7

CSSEF Client

This is some content

CHAPTER 8

CSSEF Server

Lets start with a quick overview of what everything in this directory is for, since most of it hasn't been formally organized.

cssefserver/ This directory contains the source for the pip package 'cssef-server'.

tests/ Directory for tests for the 'cssef-server' package.

readme.md A readme file for github.

cssef-server An executable file to host a CSSEF server. This is what is started by systemd.

cssef-server.service A systemd service file, used to define the 'cssef-server' service.

cssef-server.yml A service configuration file, which is read in when the cssef server is started.

makefile Used to make running several tasks related to installation and testing easier.

setup.py A setup script that defines the 'cssef-server' python package for pip.

8.1 Makefile

At this time, the makefile must be run from the current directory (. . . /CssefServer) since resources are pathed referentially.

install The actions of the install process are divided into four steps:

1. Install the 'cssef-server' package via pip.
2. Create necessary directories.
3. Copy service and config files.
4. Enable (but not start) the service within systemd.

uninstall Uninstall does most of the install process, but will leave the configuration files and database files, in case those are still needed afterward.

1. Stop and disabled the the service within systemd.

2. Remove the .service file and service executable
3. Remove the python package via pip

reinstall This simply calls the *uninstall* and *install* portions of the make file. It is important to note here that the contents of */etc/cssef/cssef-server.yml* will be overwritten.

test This will run pylint and nosetests on the library and the cssef-server executable.

CHAPTER 9

CSSEF Web Client

This is some content

CHAPTER 10

Documentation

Sphinx is used for documentation, and is hosted through Read The Docs. New commits to the repository will update the live documentation.

The autodoc extension for Sphinx is used to create reference documentation from docstrings throughout the code. This helps to enforce proper code documentation practices.

10.1 Building the docs

Building the documentation relies on sphinx and the sphinx rtd theme, but I also highly recommend installing sphinx-autobuild.

```
user@debian:~$ pip install sphinx sphinx-rtd-theme sphinx-autobuild
```

The documentation can be built manually or automatically. I'm lazy so I suggest using the automatic method, but I'll cover the manual process anyway.

10.2 Building Manually

The makefile in the root of the `docs` folder is a lightly modified version of the standard makefile that comes with sphinx, so if you're familiar with sphinx, the process is the same.

```
user@debian:~/cssef/docs$ make html
```

And that's it! The resulting html files will be located within docs/build/html. From time to time, you may want to run a `make clean` before rebuilding the documentation, just to make sure everything is fresh and up to date.

10.3 Building Automatically

Documentation can be automatically built and locally served using sphinx-autobuild. I've set up an option in the makefile to watch the whole project (excluding the documentation build directory, and the autodocs directory), that way we don't have to remember how to propery run sphinx-autobuild. All you have to do is run the following command:

```
user@debian:~/cssef/docs$ make livehtml
```

CHAPTER 11

Competition Plugins

This is some content