ScoringEngine Documentation

Release latest

pwnbus

May 02, 2023

Contents

CHAPTER 1

Overview

1.1 Why?

The goal of the ScoringEngine is to keep track of service up time in a blue teams/red team competition.

1.2 How does it work?

The general idea of the ScoringEngine is broken up into 3 separate processes, Engine, Worker, and Web.

1.2.1 Engine

The engine is responsible for tasking *Checks* that are used to verify network services each round, and determining/saving their results to the database. This process runs for the entire competition, and will sleep for a certain amount of time before starting on to the next round.

1.2.2 Worker

The worker connects to Redis and waits for *Checks* to get tasked in order to run them against . Once it receives a *Check*, it executes the command and sends the output back to the Engine.

1.2.3 Web

The web application provides a graphical view of the Competition. This includes things like a bar graph of all team's scores as well as a table of the current round's results. This can also be used to configure the properties of each service per team.

1.2.4 External Resources

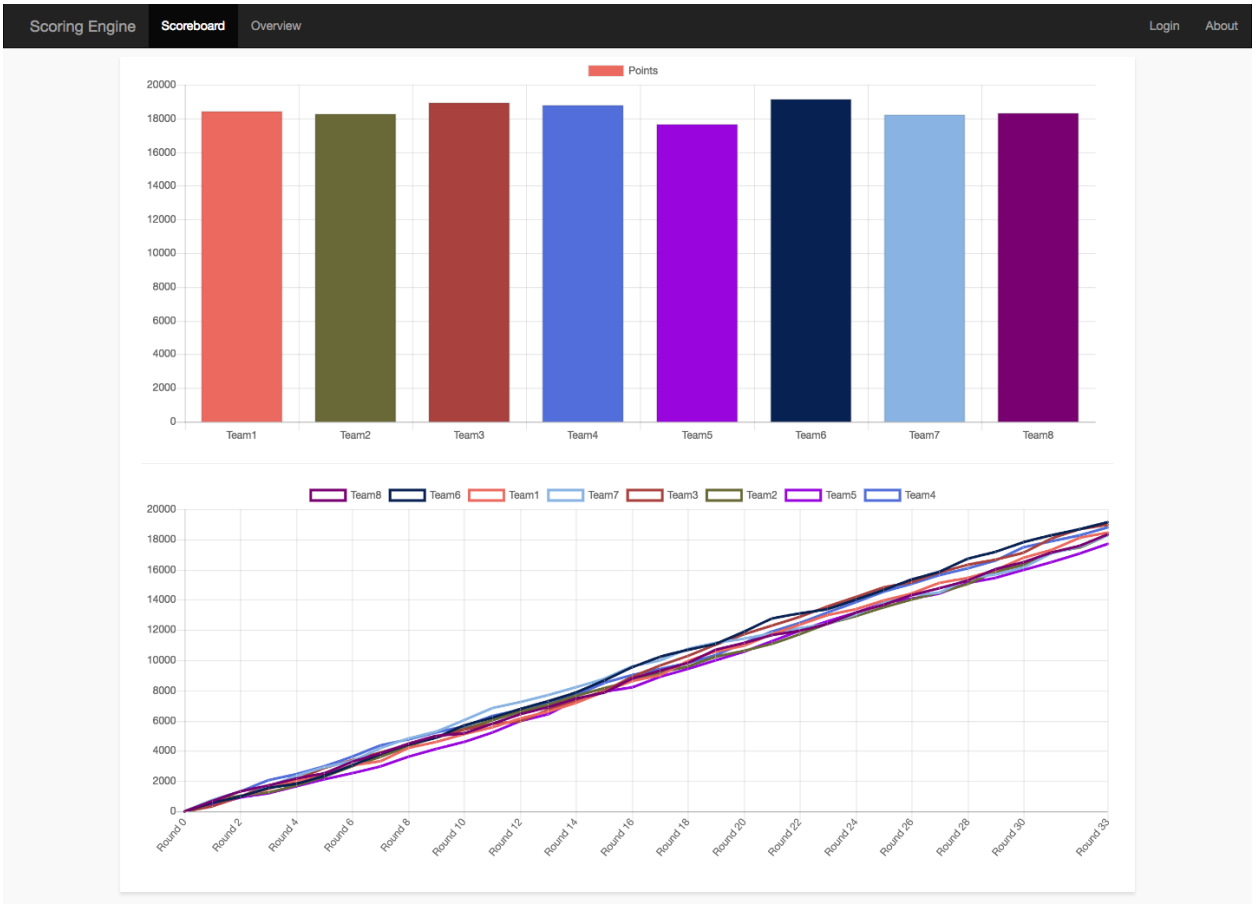
We currently use MySQL as the database, and Redis as the data store for tasks while they are getting scheduled.

1.2.5 Putting it all together

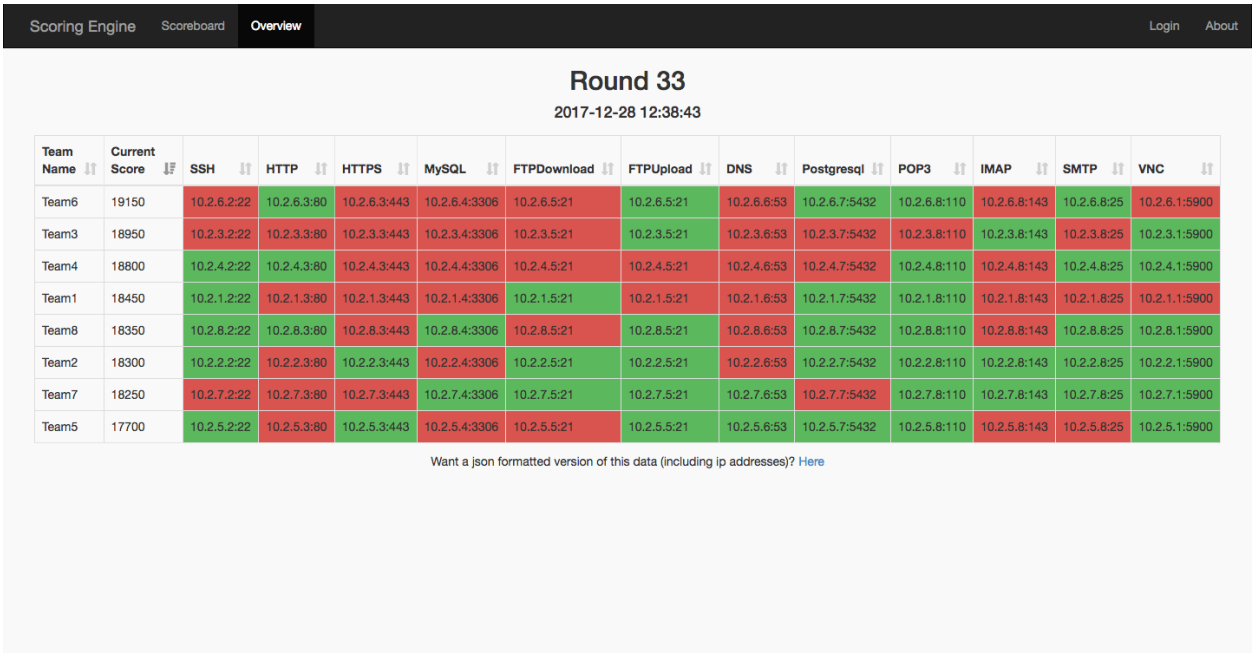
- The *Engine* starts
- The first *Round* starts
- The *Engine* tasks *Checks* out to the *Workers*
- The *Workers* execute the *Checks* and return the output to the *Engine*
- The *Engine* waits for all *Checks* to finish
- The *Engine* determines the results of each *Check*, and saves the results to the DB
- The *Engine* ends the *Round*
- The *Engine* sleeps for some time
- The second *Round* starts
- ...

1.3 Screenshots

1.3.1 Scoreboard

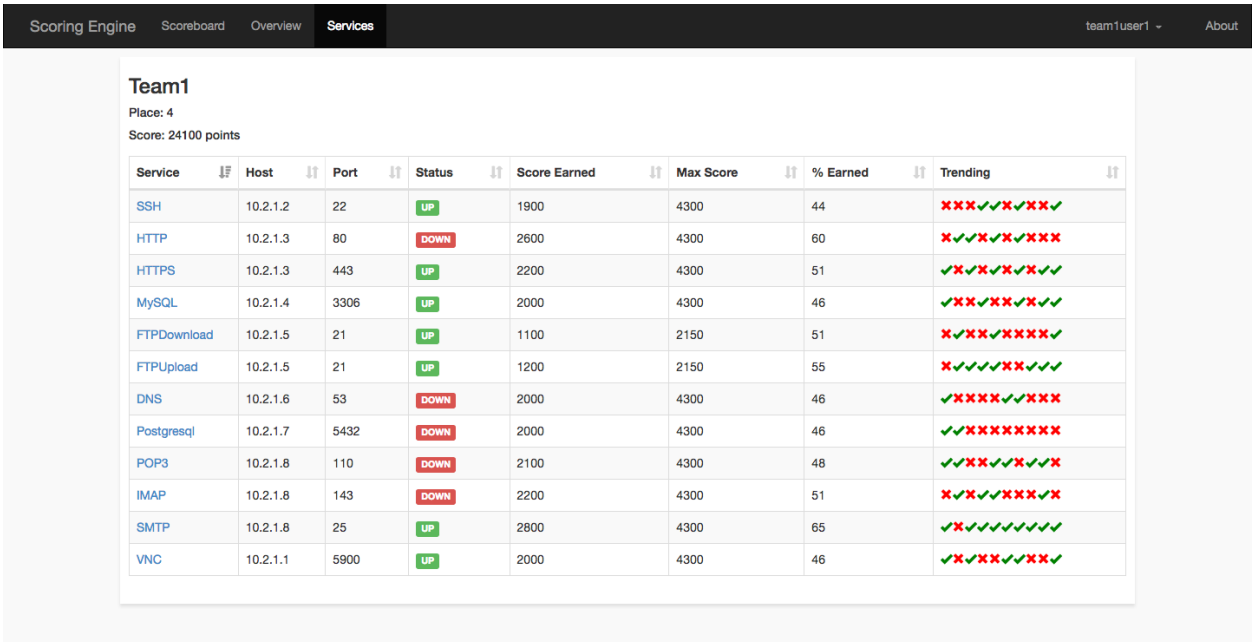


1.3.2 Overview



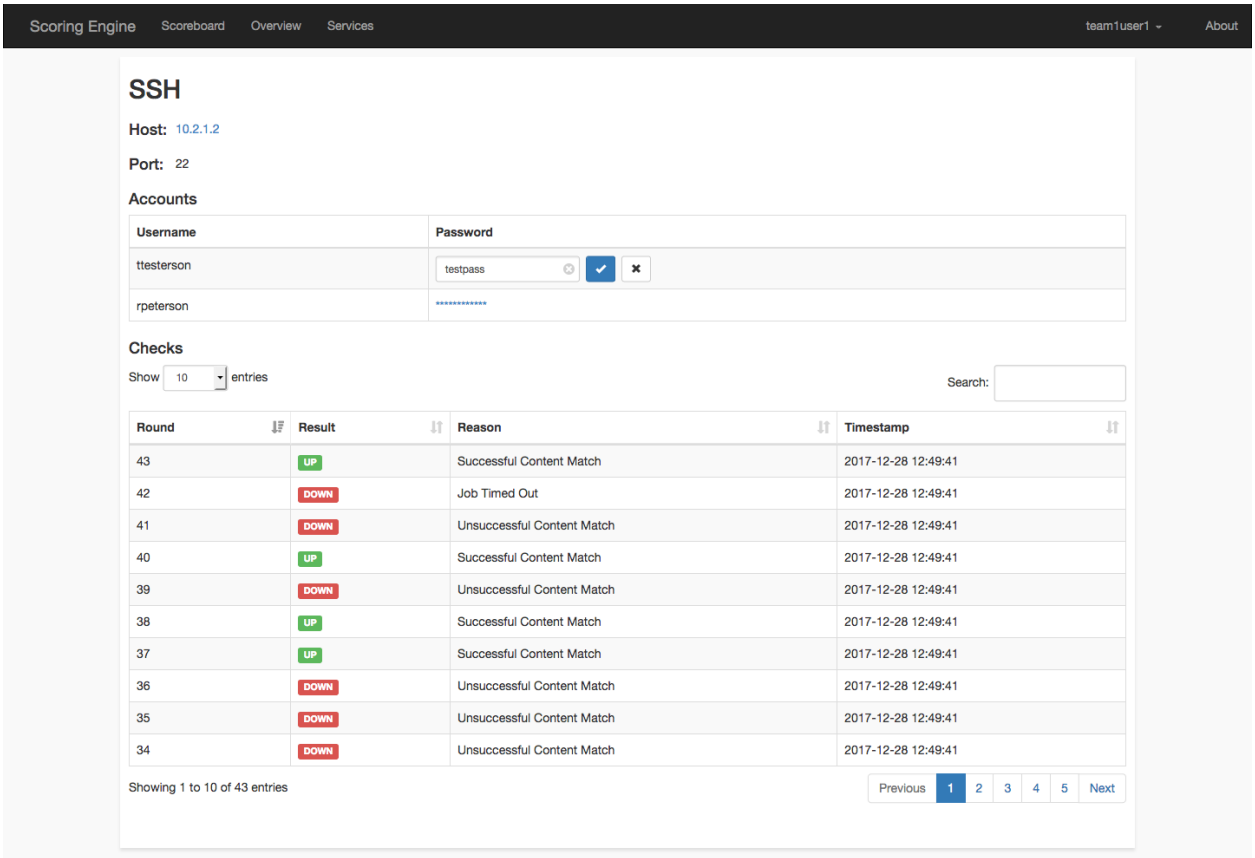
Team Name	Current Score	SSH	HTTP	HTTPS	MySQL	FTPDownload	FTPUpload	DNS	Postgresql	POP3	IMAP	SMTP	VNC
Team6	19150	10.2.6.2:22	10.2.6.3:80	10.2.6.3:443	10.2.6.4:3306	10.2.6.5:21	10.2.6.5:21	10.2.6.6:53	10.2.6.7:5432	10.2.6.8:110	10.2.6.8:143	10.2.6.8:25	10.2.6.1:5900
Team3	18950	10.2.3.2:22	10.2.3.3:80	10.2.3.3:443	10.2.3.4:3306	10.2.3.5:21	10.2.3.5:21	10.2.3.6:53	10.2.3.7:5432	10.2.3.8:110	10.2.3.8:143	10.2.3.8:25	10.2.3.1:5900
Team4	18800	10.2.4.2:22	10.2.4.3:80	10.2.4.3:443	10.2.4.4:3306	10.2.4.5:21	10.2.4.5:21	10.2.4.6:53	10.2.4.7:5432	10.2.4.8:110	10.2.4.8:143	10.2.4.8:25	10.2.4.1:5900
Team1	18450	10.2.1.2:22	10.2.1.3:80	10.2.1.3:443	10.2.1.4:3306	10.2.1.5:21	10.2.1.5:21	10.2.1.6:53	10.2.1.7:5432	10.2.1.8:110	10.2.1.8:143	10.2.1.8:25	10.2.1.1:5900
Team8	18350	10.2.8.2:22	10.2.8.3:80	10.2.8.3:443	10.2.8.4:3306	10.2.8.5:21	10.2.8.5:21	10.2.8.6:53	10.2.8.7:5432	10.2.8.8:110	10.2.8.8:143	10.2.8.8:25	10.2.8.1:5900
Team2	18300	10.2.2.2:22	10.2.2.3:80	10.2.2.3:443	10.2.2.4:3306	10.2.2.5:21	10.2.2.5:21	10.2.2.6:53	10.2.2.7:5432	10.2.2.8:110	10.2.2.8:143	10.2.2.8:25	10.2.2.1:5900
Team7	18250	10.2.7.2:22	10.2.7.3:80	10.2.7.3:443	10.2.7.4:3306	10.2.7.5:21	10.2.7.5:21	10.2.7.6:53	10.2.7.7:5432	10.2.7.8:110	10.2.7.8:143	10.2.7.8:25	10.2.7.1:5900
Team5	17700	10.2.5.2:22	10.2.5.3:80	10.2.5.3:443	10.2.5.4:3306	10.2.5.5:21	10.2.5.5:21	10.2.5.6:53	10.2.5.7:5432	10.2.5.8:110	10.2.5.8:143	10.2.5.8:25	10.2.5.1:5900

1.3.3 Team Services

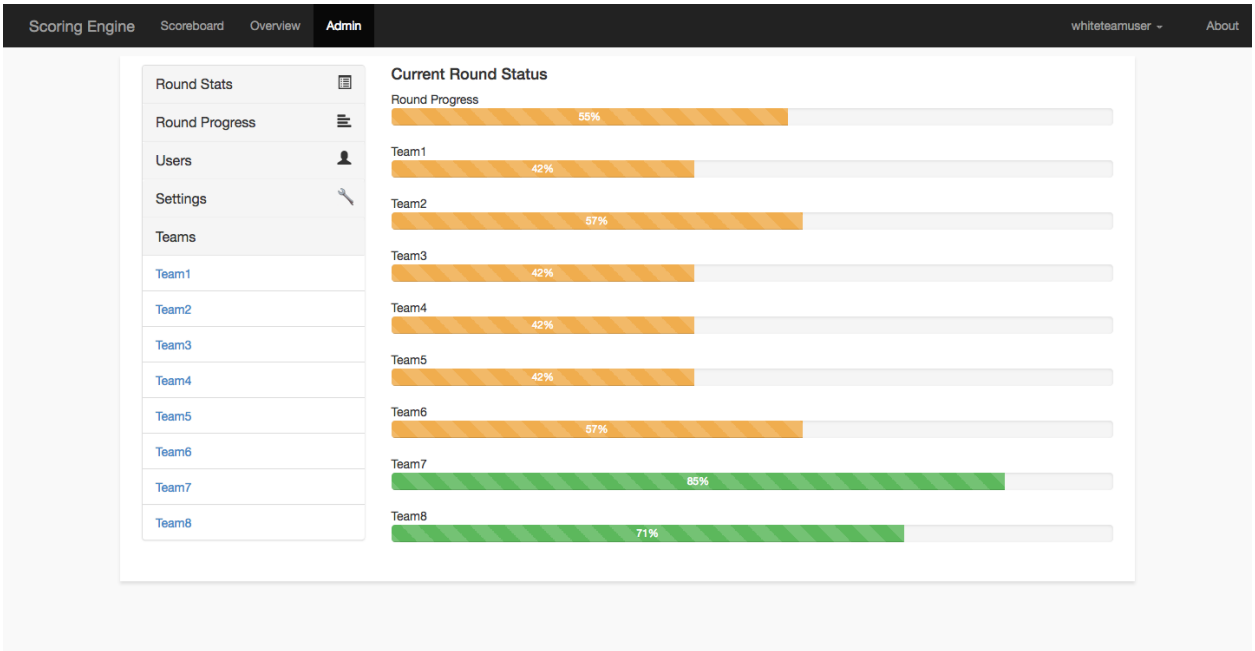


Service	Host	Port	Status	Score Earned	Max Score	% Earned	Trending
SSH	10.2.1.2	22	UP	1900	4300	44	✗✗✗✓✓✗✓✗✗✓
HTTP	10.2.1.3	80	DOWN	2600	4300	60	✗✓✓✗✓✗✓✗✗✗
HTTPS	10.2.1.3	443	UP	2200	4300	51	✓✗✓✗✓✗✓✗✓✓
MySQL	10.2.1.4	3306	UP	2000	4300	46	✓✗✗✓✗✗✓✗✓✓
FTPDownload	10.2.1.5	21	UP	1100	2150	51	✗✓✗✗✓✗✗✗✗✓
FTPUpload	10.2.1.5	21	UP	1200	2150	55	✗✓✓✓✓✗✗✓✓✓
DNS	10.2.1.6	53	DOWN	2000	4300	46	✓✗✗✗✗✓✓✗✗✗
Postgresql	10.2.1.7	5432	DOWN	2000	4300	46	✓✓✗✗✗✗✗✗✗✗
POP3	10.2.1.8	110	DOWN	2100	4300	48	✓✓✗✗✓✓✗✓✓✗
IMAP	10.2.1.8	143	DOWN	2200	4300	51	✗✓✗✓✓✗✗✗✓✗
SMTP	10.2.1.8	25	UP	2800	4300	65	✓✗✓✓✓✓✓✓✓✓
VNC	10.2.1.1	5900	UP	2000	4300	46	✓✗✓✗✗✓✓✗✗✓

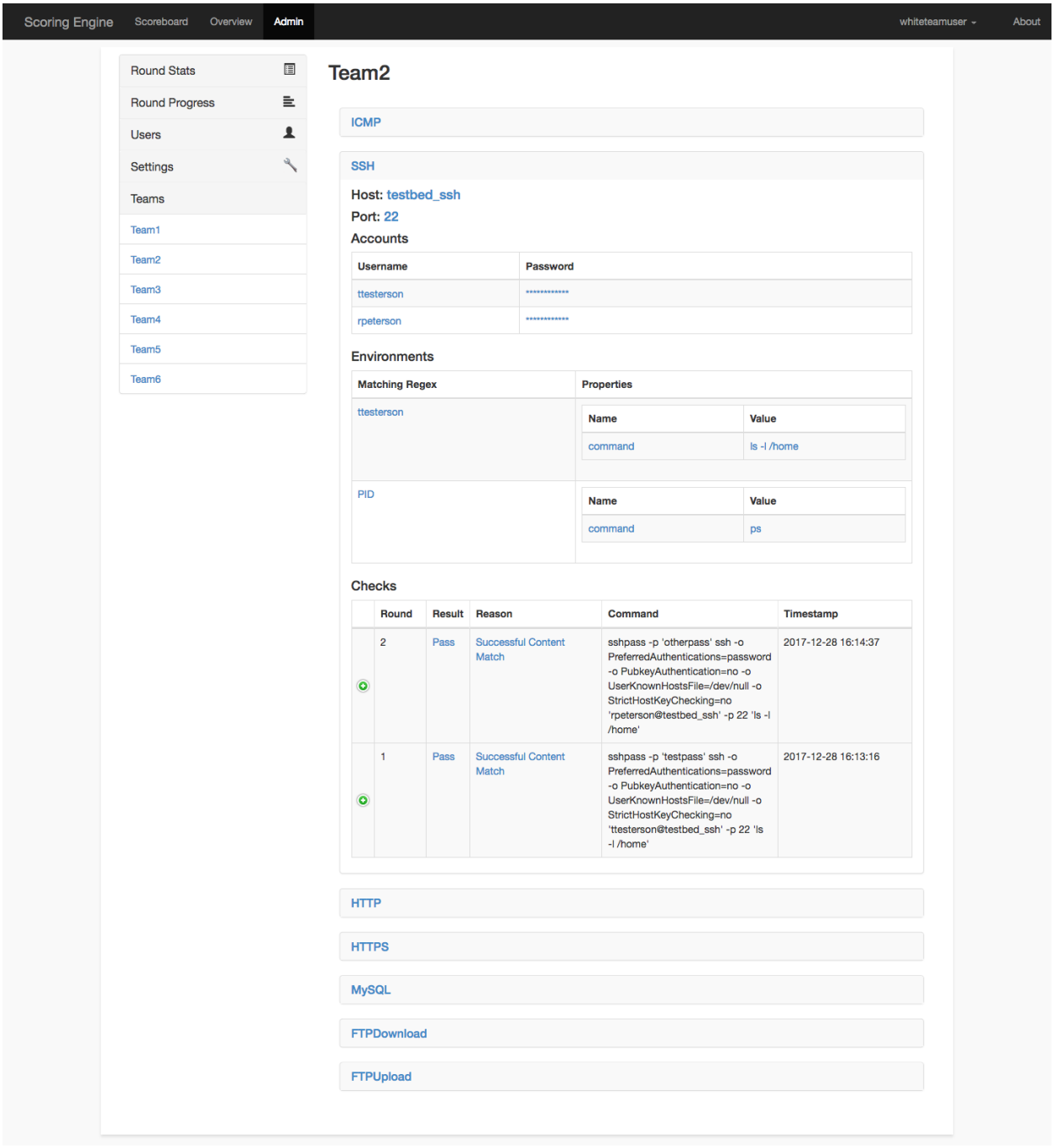
1.3.4 Specific Service



1.3.5 Round Status



1.3.6 Admin Team View



CHAPTER 2

Installation

2.1 Docker

Note: It takes a minute or 2 for all of the containers to start up and get going!

2.1.1 TestBed Environment

```
make rebuild-testbed-new
```

This command will build, stop any pre-existing scoring engine containers, and start a new environment. As part of the environment, multiple containers will be used as part of the testbed environment.

2.1.2 Environment Variables

We use certain environment variables to control the functionality of certain docker containers.

SCORINGENGINE_OVERWRITE_DB If set to true, the database will be deleted and then recreated during startup.

SCORINGENGINE_EXAMPLE If set to true, the database is populated with sample db, and the engine container will be paused. This is useful for doing development on the web app.

You can set each environment variable before each command executed, for example:

```
SCORINGENGINE_EXAMPLE=true make rebuild-new
```

2.1.3 Production Environment

Modify the bin/competition.yaml file to configure the engine according to your competition environment. Then, run the following make command to build, and run the scoring engine.

> **Warning:** This will delete the previous database, exclude the 'new' part from the command to not rebuild the db.

```
make rebuild-new
```

Then, to 'pause' the scoring engine (Ex: At the end of the day):

```
docker-compose -f docker-compose.yml stop engine
```

To 'unpause' the engine:

```
docker-compose -f docker-compose.yml start engine
```

2.2 Manual

2.2.1 Base Setup

> **Note:** Currently, the only OS we have documentation on is Ubuntu 16.04.

Install dependencies via apt-get

```
apt-get update
apt-get install -y python3.5 wget git python3.5-dev build-essential libmysqlclient-dev
```

Create engine user

```
useradd -m engine
```

Download and Install pip

```
wget -O /root/get-pip.py https://bootstrap.pypa.io/get-pip.py
python3.5 /root/get-pip.py
rm /root/get-pip.py
```

Setup virtualenvironment

```
pip install virtualenv
su engine
cd ~/
mkdir /home/engine/scoring_engine
virtualenv -p /usr/bin/python3.5 /home/engine/scoring_engine/env
```

Setup src directory

```
git clone https://github.com/scoringengine/scoringengine /home/engine/scoring_engine/
↪src
```

Install scoring_engine src python dependencies

```
source /home/engine/scoring_engine/env/bin/activate
pip install -e /home/engine/scoring_engine/src/
```

Copy/Modify configuration

```
cp /home/engine/scoring_engine/src/engine.conf.inc /home/engine/scoring_engine/src/
↪engine.conf
vi /home/engine/scoring_engine/src/engine.conf
```

Create log file locations (run as root)

```
mkdir /var/log/scoring_engine
chown -R syslog:adm /var/log/scoring_engine
```

Copy rsyslog configuration

```
cp /home/engine/scoring_engine/src/configs/rsyslog.conf /etc/rsyslog.d/10-scoring_
↪engine.conf
```

Restart rsyslog

```
systemctl restart rsyslog
```

2.2.2 Web

Install MySQL Server

```
apt-get install -y mariadb-server
sed -i -e 's/127.0.0.1/0.0.0.0/g' /etc/mysql/mysql.conf.d/mysqld.cnf
systemctl restart mysql
```

Setup MySQL

```
mysql -u root -p<insert password set during installation>
CREATE DATABASE scoring_engine;
CREATE USER 'engineuser'@'%' IDENTIFIED BY 'enginepass';
GRANT ALL on scoring_engine.* to 'engineuser'@'%' IDENTIFIED by 'enginepass';
```

Install Nginx

```
apt-get install -y nginx
```

Setup SSL in Nginx

```
mkdir /etc/nginx/ssl
cd /etc/nginx/ssl
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout server.key -out server.crt
```

Copy nginx config

```
cp /home/engine/scoring_engine/src/configs/nginx.conf /etc/nginx/sites-available/
↪scoring_engine.conf
ln -s /etc/nginx/sites-available/scoring_engine.conf /etc/nginx/sites-enabled/
rm /etc/nginx/sites-enabled/default
systemctl restart nginx
```

Setup web service

```
cp /home/engine/scoring_engine/src/configs/web.service /etc/systemd/system/scoring_
↪engine-web.service
```

Modify configuration

```
vi /home/engine/scoring_engine/src/engine.conf
```

Install uwsgi

```
pip install uwsgi
```

Start web

```
systemctl enable scoring_engine-web
systemctl start scoring_engine-web
```

Monitoring

```
journalctl -f _SYSTEMD_UNIT=scoring_engine-web.service
tail -f /var/log/scoring_engine/web.log
tail -f /var/log/scoring_engine/web-nginx.access.log
tail -f /var/log/scoring_engine/web-nginx.error.log
```

2.2.3 Engine

Install Redis

```
apt-get install -y redis-server
```

Setup Redis to listen on external interface

```
sed -i -e 's/bind 127.0.0.1/bind 0.0.0.0/g' /etc/redis/redis.conf
systemctl restart redis
```

Setup Engine service (run as root)

```
cp /home/engine/scoring_engine/src/configs/engine.service /etc/systemd/system/scoring_
↪engine-engine.service
```

Modify configuration

```
su engine
vi /home/engine/scoring_engine/src/engine.conf
```

Setup scoring engine teams and services

```
su engine
vi /home/engine/scoring_engine/src/bin/competition.yaml
source /home/engine/scoring_engine/env/bin/activate
/home/engine/scoring_engine/src/bin/setup
```

Start engine service (must run as root)

```
systemctl start scoring_engine-engine
```

Monitor engine

```
journalctl -f _SYSTEMD_UNIT=scoring_engine-engine.service
tail -f /var/log/scoring_engine/engine.log
```

2.2.4 Worker

Modify hostname

```
hostname <INSERT CUSTOM HOSTNAME HERE>
```

Setup worker service (run as root)

```
cp /home/engine/scoring_engine/src/configs/worker.service /etc/systemd/system/scoring_
↪engine-worker.service
```

Modify configuration

Change REDIS host/port/password fields to main engine host::

```
vi /home/engine/scoring_engine/src/engine.conf
```

Modify worker to customize number of processes. Append '–concurrency <num of processes>' to the celery command line. If not specified, it defaults to # of CPUs.

```
vi /home/engine/scoring_engine/src/bin/worker
```

Start worker service (must run as root)

```
systemctl enable scoring_engine-worker
systemctl start scoring_engine-worker
```

Monitor worker

```
journalctl -f _SYSTEMD_UNIT=scoring_engine-worker.service
tail -f /var/log/scoring_engine/worker.log
```

Install dependencies for DNS check

```
apt-get install -y dnsutils
```

Install dependencies for HTTP/HTTPS check

```
apt-get install -y curl
```

Install dependencies for most of the checks

```
apt-get install -y medusa
```

Install dependencies for SSH check

```
source /home/engine/scoring_engine/env/bin/activate && pip install -I "cryptography>
↪=2.4,<2.5" && pip install "paramiko>=2.4,<2.5"
```

Install dependencies for LDAP check

```
apt-get install -y ldap-utils
```

Install dependencies for Postgresql check

```
apt-get install -y postgresql-client
```

Install dependencies for Elasticsearch check

```
source /home/engine/scoring_engine/env/bin/activate && pip install -I "requests>=2.21,
↪<2.22"
```

Install dependencies for SMB check

```
source /home/engine/scoring_engine/env/bin/activate && pip install -I "pysmb>=1.1,<1.2
↪"
```

Install dependencies for RDP check

```
apt-get install -y freerdp-x11
```

Install dependencies for MSSQL check

```
apt-get install -y apt-transport-https
curl -s https://packages.microsoft.com/keys/microsoft.asc | apt-key add -
curl -s https://packages.microsoft.com/config/ubuntu/16.04/prod.list | tee /etc/apt/
↪sources.list.d/msprod.list
apt-get update
ACCEPT_EULA=Y apt-get install -y locales mssql-tools unixodbc-dev
echo "en_US.UTF-8 UTF-8" >> /etc/locale.gen
locale-gen
```

Install dependencies for SMTP/SMTPS check

```
cp /home/engine/scoring_engine/src/scoring_engine/checks/bin/smtp_check /usr/bin/smtp_
↪check
cp /home/engine/scoring_engine/src/scoring_engine/checks/bin/smtps_check /usr/bin/
↪smtps_check
```

(continues on next page)

(continued from previous page)

```
chmod a+x /usr/bin/smtp_check
chmod a+x /usr/bin/smtps_check
```

Install dependencies for NFS check

```
apt-get install -y libnfs-dev
source /home/engine/scoring_engine/env/bin/activate && pip install -I "libnfs==1.0.
↪post4"
```

Install dependencies for OpenVPN check

```
apt-get install -y openvpn iproute2 sudo
cp /home/engine/scoring_engine/src/docker/worker/sudoers /etc/sudoers
```

Install dependencies for Telnet check

```
source /home/engine/scoring_engine/env/bin/activate && pip install -I "telnetlib3==1.
↪0.1"
```

CHAPTER 3

Configuration

3.1 Location to config file

3.1.1 Docker

Note: This file needs to be edited before running the make commands.

```
<path to source root>/docker/engine.conf.inc
```

3.1.2 Manual

Note: Need to restart each scoring engine service once the config is modified.

```
/home/engine/scoring_engine/src/engine.conf
```

3.2 Configuration Keys

Note: Each of these config keys can be expressed via environment variables (and take precendence over the values defined in the file). IE: To define target_round_time, I'd set SCORINGENGINE_TARGET_ROUND_TIME=3.

Key Name	Description
checks_location	Local path to directory of checks
target_round_time	Length of time (seconds) the engine should target per round
worker_refresh_time	Amount of time (seconds) the engine will sleep for in-between polls of worker status
worker_num_concurrent_tasks	The number of concurrent tasks the worker will run. Set to -1 to default to number of processors.
worker_queue	The queue name for a worker to pull tasks from. This can be used to control which workers get which service checks. Default is 'main'
timezone	Local timezone of the competition
debug	Determines wether or not the engine should be run in debug mode (useful for development). The worker will also display output from all checks.
db_uri	Database connection URI
cache_type	The type of storage for the cache. Set to null to disable caching
redis_host	The hostname/ip of the redis server
redis_port	The port of the redis server
redis_password	The password used to connect to redis (if no password, leave empty)

CHAPTER 4

Implemented Checks

4.1 DNS

Queries a DNS server for a specific record

Custom Properties:

qtype	type of record (A, AAAA, CNAME, etc)
domain	domain/host to query for

4.2 Elasticsearch

Uses python requests to insert message and then query for same message

Custom Properties:

index	index to use to insert the message
doc_type	type of the document

4.3 FTP

Uses python ftplib to login to an FTP server, upload a file, login again to FTP and download file

Uses Accounts

Custom Properties:

remotefilepath	absolute path of file on remote server to upload/download
filecontents	contents of the file that we upload/download

4.4 HTTP(S)

Sends a GET request to an HTTP(S) server

Custom Properties:

useragent	specific useragent to use in the request
vhost	vhost used in the request
uri	uri of the request

4.5 ICMP

Sends an ICMP Echo Request to server

Custom Properties: *none*

4.6 IMAP(S)

Uses medusa to login to an imap server

Uses Accounts

Custom Properties:

domain	domain of the username

4.7 LDAP

Uses ldapsearch to login to ldap server. Once authenticated, it performs a lookup of all users in the same domain

Uses Accounts

Custom Properties:

domain	domain of the username
base_dn	base dn value of the domain (Ex: dc=example,dc=com)

4.8 MSSQL

Logs into a MSSQL server, uses a database, and executes a specific SQL command

Uses Accounts

Custom Properties:

database	database to use before running command
command	SQL command that will execute

4.9 MySQL

Logs into a MySQL server, uses a database, and executes a specific SQL command

Uses Accounts

Custom Properties:

database	database to use before running command
command	SQL command that will execute

4.10 NFS

Uses python libnfs to login to an NFS server, write a file, login again to NFS and read a file

Custom Properties:

remotefilepath	absolute path of file on remote server to upload/download
filecontents	contents of the file that we upload/download

4.11 POP3(S)

Uses medusa to login to an pop3 server

Uses Accounts

Custom Properties:

domain	domain of the username

4.12 PostgreSQL

Logs into a postgresql server, selects a database, and executes a SQL command

Uses Accounts

Custom Properties:

database	database to use before running command
command	SQL command that will execute

4.13 RDP

Logs into a system using RDP with an account/password

Uses Accounts

Custom Properties: *none*

4.14 SMB

Logs into a system using SMB with an account/password, and hashes the contents of a specific file on a specific share

Uses Accounts

Custom Properties:

share	name of the share to connect to
file	local path of the file to access
hash	SHA256 hash of the contents of the file

4.15 SMTP(S)

Logs into an SMTP server and sends an email

Uses Accounts

Custom Properties:

touser	address that the email will be sent to
subject	subject of the email
body	body of the email

4.16 SSH

Logs into a system using SSH with an account/password, and executes command(s)

Note: Each command will be executed independently of each other in a separate ssh connection.

Uses Accounts

Custom Properties:

commands	';' delimited list of commands to run (Ex: id;ps)

4.17 VNC

Connects and if specified, will login to a VNC server

Uses Accounts (optional)

Custom Properties: *none*

4.18 WinRM

Logs into a system using WinRM with an account/password, and executes command(s)

Uses Accounts

Custom Properties:

commands	';' delimited list of commands to run (Ex: ipconfig /all;whoami)

CHAPTER 5

Development

Note: Currently we support 2 ways of working on the Scoring Engine. You can either use the existing Docker environment, or you can run each service locally using python 3. If you choose to do your development locally, we recommend using virtual environments.

5.1 Initial Setup

These steps are for if you want to do your development locally and run each service locally as well.

5.1.1 Create Config File

```
cp engine.conf.inc engine.conf
sed -i '' 's/debug = False/debug = True/g' engine.conf
```

Hint: If debug is set to True, the web ui will automatically reload on changes to local file modifications, which can help speed up development. This config setting will also tell the worker to output all check output to stdout.

5.1.2 Install Required Dependencies

```
pip install -e .
```

5.1.3 Populate Sample DB

```
python bin/setup --example --overwrite-db
```

5.2 Run Services

5.2.1 Web UI

```
python bin/web
```

Then, access localhost:5000

Table 1: Credentials

Username	Password
whiteteamuser	testpass
redteamuser	testpass
team1user1	testpass
team2user1	testpass
team2user2	testpass

Note: The engine and worker do NOT need to be running in order to run the web UI.

5.2.2 Engine

Both the engine and worker services require a redis server to be running. Redis can be easily setup by using the existing docker environment.

```
python bin/engine
```

5.2.3 Worker

```
python bin/worker
```

5.3 Run Tests

We use the pytest testing framework.

Note: The tests use a separate db (sqlite in memory), so don't worry about corrupting a production db when running the tests.

First, we need to install the dependencies required for testing.

```
pip install -r tests/requirements.txt
```

Next, we run our tests

```
pytest tests
```

Hint: Instead of specifying the tests directory, you can specify specific file(s) to run: *pytest tests/scoring_engine/test_config.py*

5.4 Modifying Documentation

We use sphinx to build the documentation.

First, we need to install the dependencies required for documentation.

```
pip install -r docs/requirements.txt
```

Next, we build our documentation in html format.

```
cd docs
make html
open build/html/index.html
```

CHAPTER 6

Create New Service Check

Each service check (DNS, SSH, ICMP etc) are essentially simple commands that the worker will execute and gather the output of. This output is then handled by the engine to determine if a service check is successful or not for that round.

For the sake of explaination, we'll be walking through our documentation by taking a look at the SSH check.

6.1 Create Check Source File

Each check is stored in the scoring_engine/checks directory.

Let's take a look at what the SSH check file looks like (scoring_engine/checks/ssh_check.py):

```
1  class SSHCheck(BasicCheck):
2      required_properties = ['commands']
3      CMD = CHECKS_BIN_PATH + '/ssh_check {0} {1} {2} {3} {4}'
4
5      def command_format(self, properties):
6          account = self.get_random_account()
7          return (
8              self.host,
9              self.port,
10             account.username,
11             account.password,
12             properties['commands']
13         )
```

Note: The main point of each check source code, is to generate a command string. The format of this string is defined in the CMD variable. The plugin executes the command_format function, which outputs a list of the parameters to fill in the formatted CMD variable.

- **Line 1** - This is the Class name of the check, and will need to be something you reference in bin/competition.yaml
- **Line 2** - We specificy what properties this check requires. This can be any value, as long as it's defined in bin/competition.yaml.
- **Line 3** - This is the format of the command. The SSH Check requires an additional file to be created in addition to this file, which will be stored in CHECKS_BIN_PATH (this is scoring_engine/checks/bin). We're also specifying placeholders as parameters, as we will generate dynamically. If the binary that the command will be running is already on disk, (like ftp or nmap), then we don't need to use the CHECKS_BIN_PATH value, we can reference the absolute path specifically.
- **Line 5** - This is where we specify the custom parameters that will be passed to the CMD variable. We return a list of parameters that gets filled into the CMD.
- **Line 6** - This function provides the ability to randomly select an account to use for credentials. This allows the engine to randomize which credentials are used each round.

Now that we've created the source code file, let's look at what custom shell script we're referring to in the check source code.

```
#!/usr/bin/env python

# A scoring engine check that logs into SSH and runs a command
# The check will login each time and run ONE command
# The idea of running separate sessions is to verify
# the state of the machine was changed via SSH
# IE: Login, create a file, logout, login, verify file is still there, logout
#
# To install: pip install -I "cryptography>=2.4,<2.5" && pip install "paramiko>=2.4,
↪<2.5"

import sys
import paramiko


if len(sys.argv) != 6:
    print("Usage: " + sys.argv[0] + " host port username password commands")
    print("commands parameter supports multiple commands, use ';' as the delimeter")
    sys.exit(1)

host = sys.argv[1]
port = sys.argv[2]
username = sys.argv[3]
password = sys.argv[4]
commands = sys.argv[5].split(';')

# RUN SOME CODE
last_command_output = "OUTPUT FROM LAST COMMAND"

print("SUCCESS")
print(last_command_output)
```

For the sake of copy/paste, I've removed what code is actually run for SSH, but that can be seen here.

As we can see, this is just a simple script (and can in fact be any language as long as it's present on the worker), that takes in a few parameters, and prints something to the screen. The engine takes the output from each command, and determines if a check is successful by matching that against the matching_content value defined in bin/competition.yaml. Any output from this command will also get presented in the Web UI, so it can be used for troubleshooting purposes for white/blue teams.

In this example, our matching_content value will be “SUCCESS”.

6.2 Create Service Definition

Now that we’ve created our check source code, we now need to add it to the competition so that it will run!

```
1  - name: SSH
2    check_name: SSHCheck
3    host: testbed_ssh
4    port: 22
5    points: 150
6    accounts:
7    - username: ttesterson
8      password: testpass
9    - username: rpeterson
10     password: otherpass
11   environments:
12   - matching_content: "^SUCCESS"
13     properties:
14     - name: commands
15       value: id;ls -l /home
16   - matching_content: PID
17     properties:
18     - name: commands
19       value: ps
```

- **Line 1** - The name of the service. This value must be unique per team and needs to be defined for each team.
- **Line 2** - This is the classname of the check source code. This is how we tell the engine which check plugin we should execute.
- **Line 3** - The host/ip of the service to check.
- **Line 4** - The port of the service to check.
- **Line 5** - The amount of points given per successful check per round.
- **Line 6-10** - A list of credentials for this service. Each round, the engine will randomly select a set of credentials to use.
- **Line 11-19** - A list of environments for this service. Each round, the engine will randomly select an environment to use. This allows for the flexibility of running one SSH command this round, but another command another round, and so on.
- **Line 12** - We match this value against the output from the check command, and compare it to identify if the check is Successful or not. We define it per environment, as this might change depending on the properties for each round.
- **Line 13-15** - The properties defined in the check source code. Notice how we said the ‘commands’ property was required in the check source? This is where we define all of those properties. The value is whatever value this property should be.

6.3 Contribute Check to Repository

Depending on the check and what it does, we might be interested in including your check into our github repository!

6.3.1 Create Unit Test File

Each check source code has a corresponding unit test, which simply generates a test CMD, and compares that against the expected command string.

An example unit test for SSH looks like this (tests/scoring_engine/checks/test_ssh.py):

```
1  from scoring_engine.engine.basic_check import CHECKS_BIN_PATH
2
3  from tests.scoring_engine.checks.check_test import CheckTest
4
5
6  class TestSSHCheck(CheckTest):
7      check_name = 'SSHCheck'
8      properties = {
9          'commands': 'ls -l;id'
10     }
11     accounts = {
12         'pwnbus': 'pwnbuspass'
13     }
14     cmd = CHECKS_BIN_PATH + "/ssh_check '127.0.0.1' 1234 'pwnbus' 'pwnbuspass' 'ls -l;
   ↪id'"
```

- **Line 1** - Since we're adding additional files, we want to use the dynamically created CHECKS_BIN_PATH variable.
- **Line 3** - Import the CheckTest parent class which all check tests inherit from.
- **Line 6** - Create the unit test class. The classname must start with 'Test'.
- **Line 7** - This points to the classname of the check source code.
- **Line 8-10** - Define an example set of properties the test will use.
- **Line 11-13** - Define an example set of credentials the test will use.
- **Line 14** - Define an expected command string to verify the check source code works as expected.

6.3.2 Verify Unit Test

```
py.test tests/scoring_engine/checks/test_ssh.py
```

If all is well, then commit these files and Create a PR