**DePaul University**

**CORE Admin**

**Final Report**

**10/02/2018**

**Grant No. H98230-17-1-0318**

**Initial Setup** [**2**](#_gjdgxs)

**CORE Administration LAMP stack** [**2**](#_1fob9te)

**CORE Scoring Engine** [**2**](#_1fob9te)

[**Core Administration 2**](#_gjdgxs)

[**CORE gaming boxes**](#_1fob9te) 3

[**CORE Management Console**](#_3znysh7) 4

[Login Page](#_2et92p0) 4

[Lab/Competition Addition Page](#_tyjcwt) 5

[Lab/Competition Configuration Page](#_3dy6vkm) 6

[CORE Administrator Database](#_1t3h5sf) 9

[**Scoring Engine**](#_4d34og8) 10

[**Competition Database**](#_2s8eyo1) 12

[**RESTful API**](#_17dp8vu) [**1**](#_3rdcrjn)3

Troubleshooting 14

# Initial Setup

## CORE Administration LAMP Stack:

The CORE Administration LAMP stack hosts the CORE Management console. For this console to be functional there are a few manual modifications a user needs to complete:

1. Ensure that the Kubernetes master and worker node hostnames within resources/config.php are updated
2. Install Docker
   1. Administrators have the option of pulling the CORE LAMP stack container or building it with the provided files and Dockerfile
      1. The image may be pulled with the following command: docker pull morph3us/core\_lamp\_stack
3. Install a LAMP stack
   1. Ensure that the MySQL database password in resources/config.php is updated if the database password is not “password”
   2. Add the provided database (db.sql) to the MySQL database
4. Ensure that the apache user is able to query the Docker daemon
   1. This may be accomplished by setting a blank password for the apache user and adding the apache user to the sudoers file
5. Move all CORE Management Console files to the users desired location within the web-root directory (e.g. /var/www/ or /var/www/html)

Once the files are added to the appropriate directory browse to setup.php within your browser to configure an administrator account. Once the account is created you will be prompted to setup Google Authenticator with your account[[1]](#footnote-0). The Google Authenticator application may be downloaded on Windows, Linux, macOS, iOS, and Android. Once the account is setup the install of the CORE management console is complete.

## CORE Scoring Engine:

Each Kubernetes worker node must be running a version of Python 3. The scoring engine files must be deployed on each worker node and ran separately. The scoring engines do not communicate with each other across worker nodes. The scoring engine is comprised of three Python files: scoring.py, socket\_handler.py, and worker\_threads.py. An administrator may start a scoring engine by running the following command: *python3 socker\_handler.py*.

# Core Administration

The CORE platform consists of: (1) CORE gaming boxes; (2) CORE Management Console; (3) Scoring Engine; (4) Competition Database; and (5) RESTful API.

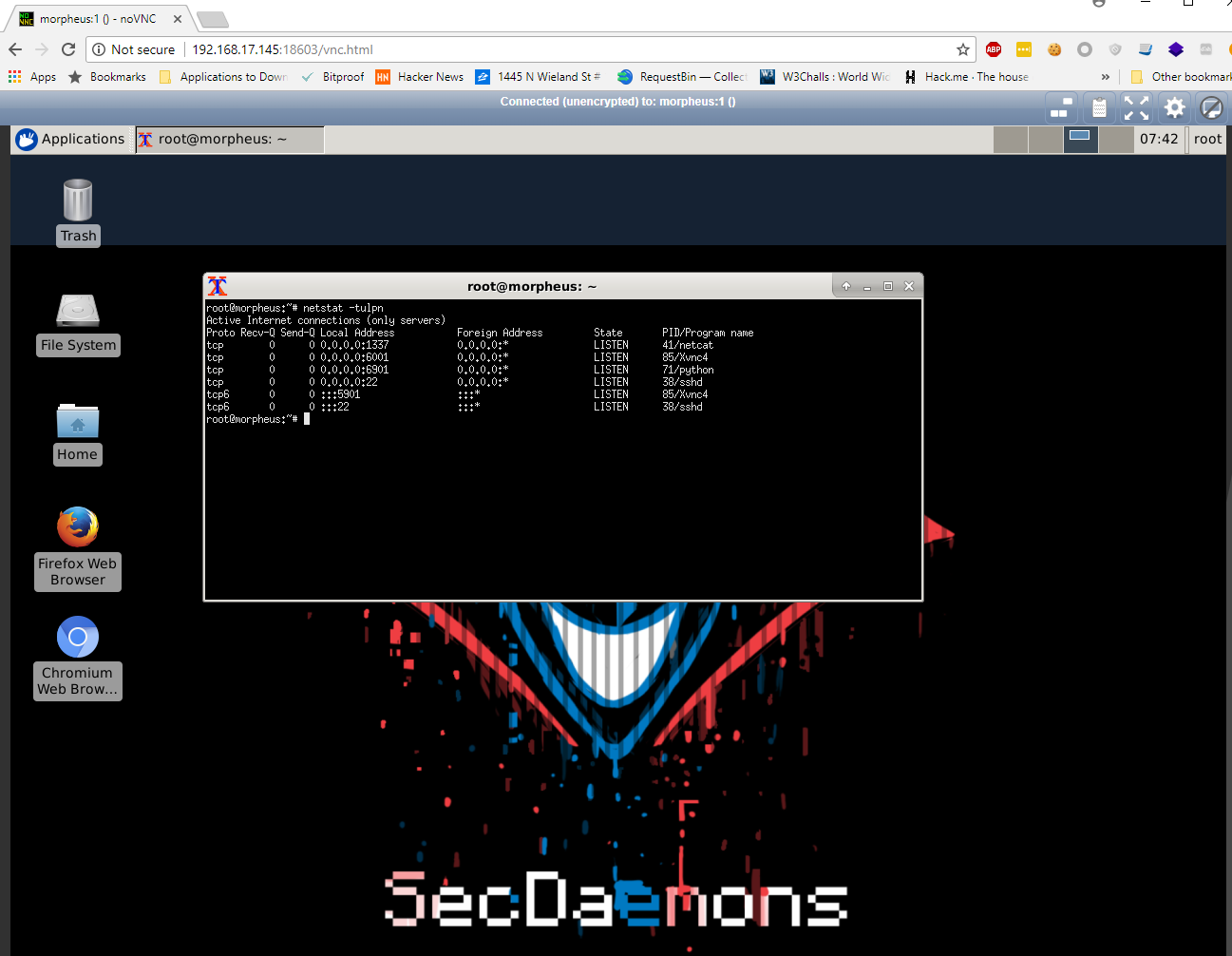
## CORE gaming boxes

The CORE gaming boxes are Docker containers which are assigned to competitors. The current implementation supports a 1-1 relationship between Docker container and competitor. A future implementation will support a many-1 relationship between Docker container and competitor. These Docker containers are based off images which have previously been built by an administrator in the CORE Management Console (discussed further below). Before a competition, lab, or final exam, each user will be provided with credentials to access their host via VNC or SSH. Figure 1 shows the VNC login page for VNC access to a CORE gaming box.



*Figure 1: CORE Gaming Box Remote Access via VNC.*

Once a user is authenticated to their CORE gaming box, he/she is presented with a VNC desktop environment which includes a terminal. An example of the supplied CORE gaming box is shown in Figure 2.



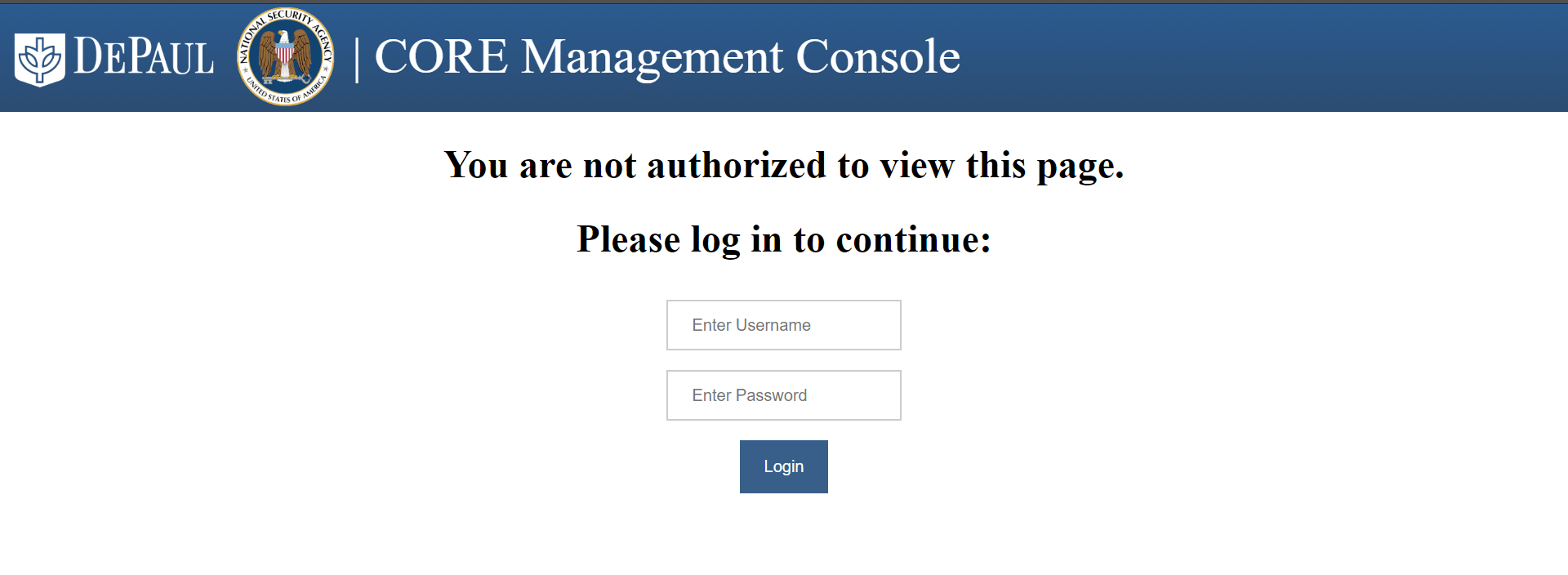
*Figure 2: CORE Gaming Box.*

## CORE Management Console

The CORE Administration engine simplifies deployment and management of competitions, labs, or final exams for administrators. Administrators are able to start/stop containers (gaming boxes), upload competition configurations, add administrators, register users for a competition, and manage a running competition, lab, or final exam in real-time.

### Login Page

Access to CORE Administration is limited to pre-defined administrators who authenticate with 2nd Factor Authentication (2FA) through Google Authenticator. Figure 3 shows the basic CORE Administration login page.

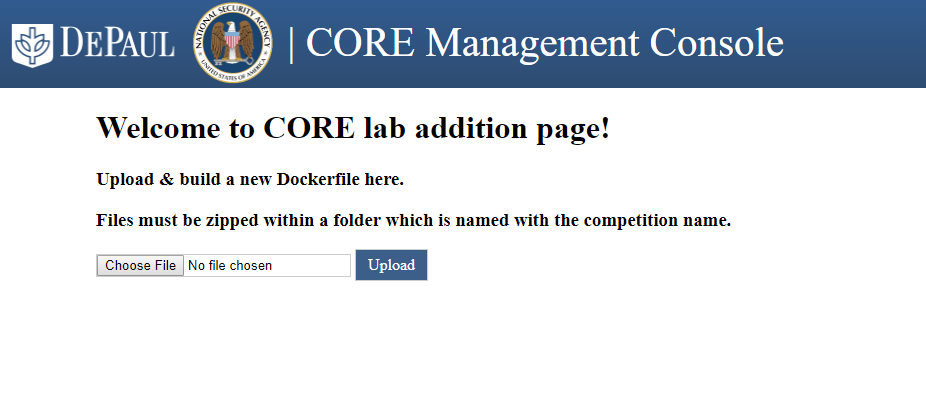


*Figure 3: CORE Administration Login Page.*

### Lab/Competition Addition Page

The CORE platform comes with a pre-loaded lab (King of the Hill) and a sample competition configuration. This preloaded lab is located on the Docker Hub and may be pulled onto a host by issuing the following command: docker pull morph3us/cyberrange. This image must manually be added to the local Docker image repository as it does not follow the workflow for Lab/Competition addition.

Administrators and educators who wish to upload their own labs, competitions, or final exams, must zip all files needed to accompany their Dockerfile (CORE Gaming Box) and upload them to the CORE platform as shown in Figure 4. To ensure that a Docker image is functional it is recommended that an administrator build their Docker image locally and thoroughly test that it works as intended before uploading to the CORE Management Console. Once the zip uploads the image is automatically created and is available for use.



*Figure 4: CORE Lab/Competition Addition Page.*

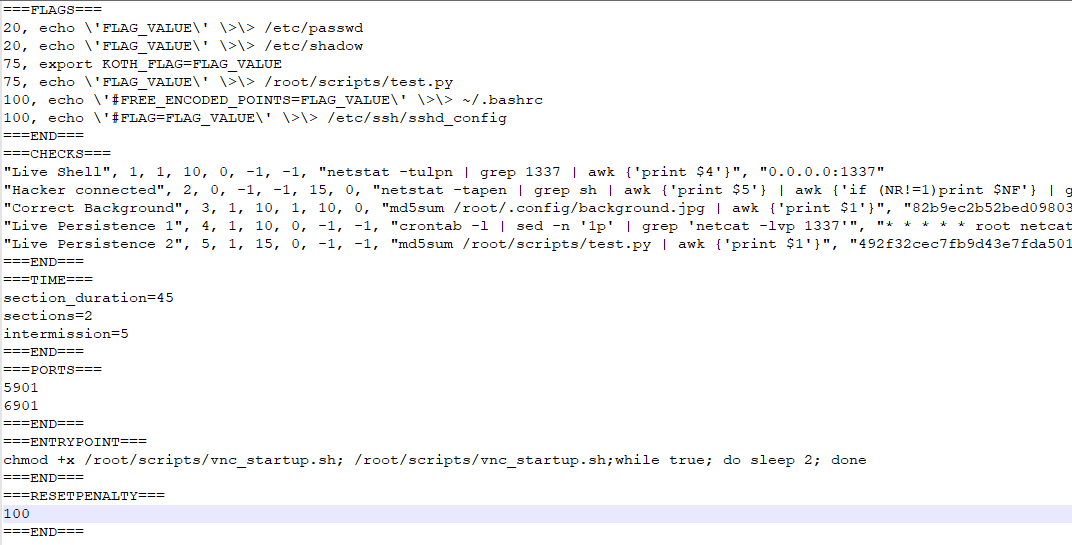
### 

### Lab/Competition Configuration Page

Once a competition image is uploaded, administrators and educators need to upload a configuration file which defines the flags, scoring checks, timing, ports to expose, the entrypoint for the Dockerfile, and the reset penalty on the CORE platform as shown in Figure 5. An example lab/competition configuration file is shown in Figure 6. Once the competition is added, administrators and educators are redirected to the CORE home page.



*Figure 5: CORE Lab/Competition Configuration Page.*



*Figure 6: CORE Lab/Competition Configuration File.*

The flag portion of the CORE configuration file requires a user to enter the amount of points awarded for submitting a flag, and the Operating System (OS) command to run on the CORE gaming box to deploy the flag to its desired location. Administrators should test these commands locally to ensure that they produce the desired outcome.

The scoring checks are a custom protocol which was written to support various scoring scenarios in real-time. These checks are ran every second and are displayed to competitors via the CORE Live Scoring front-end. For full comprehension of the protocol the following checks will be dissected:

***"Live Shell", 1, 1, 10, 0, -1, -1, "netstat -tulpn | grep 1337 | awk {'print $4'}", "0.0.0.0:1337",***

***"Hacker connected", 2, 0, -1, -1, 15, 0, "netstat -tapen | grep sh | awk {'print $5'} | awk {'if (NR!=1)print $NF'} | grep -v '0.0.0.0' | grep -v ':::\*'", ""***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name of Check** | **Check Number** | **Default Value (0=red,1=green)** | **Player points** | **Value to award points (-1=never, 0=False, 1=True)** | **Attacker points** | **Value to award points** | **Check OS command** | **Expected return value** |
| Live Shell | 1 | 1 | 10 | 0 | -1 | -1 | netstat... | 0.0.0.0:1337 |
| Hacker connected | 2 | 0 | -1 | -1 | 15 | 0 | netstat... | Empty String |

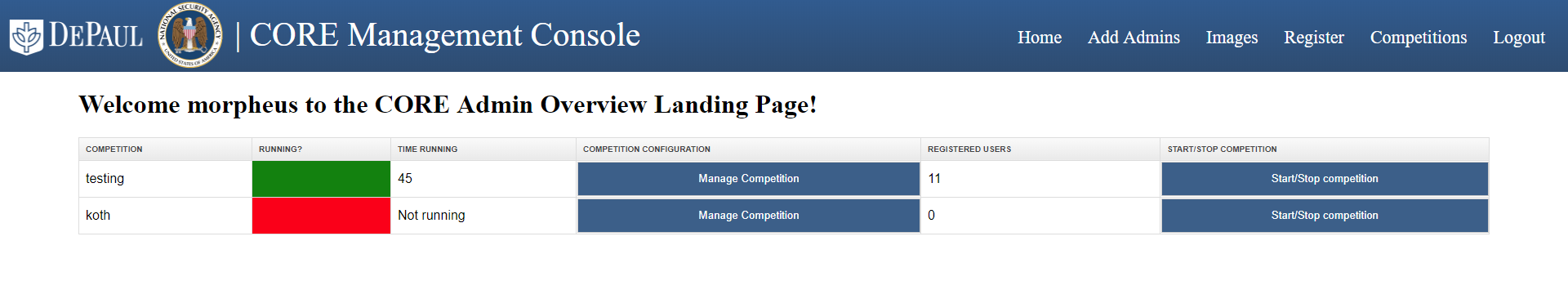
The value to awards points column specifies when a player or attacker should be awarded the points for the check. For example, the *Hacker Connected* check will award 15 points to the attacker when the return value does not equal the expected return value (i.e. the return value from the OS command is **not** an empty string). The *Hacker Connected* OS command will return the attacker’s IP address that is connected to the players gaming box. It is important to note that the player is never awarded points in this check scenario. In the *Live Shell* check scenario the attacker is never awarded points. The *Live Shell* check determines whether the player has identified the malicious open port on their machine and taken appropriate actions to close it.

The next few lines of the configuration file allow the administrator to specify the timing of the competition. The sections and intermission variables allow an administrator to define how many times should the competition timer start again and for how long should the timer stall between sections. Although these variables are stored in the CORE Management Console database, the Scoring Engine currently does not support sections and intermission.

The open ports section of the configuration file allows administrators to specify which ports need to be exposed by Kubernetes for competitors. The entrypoint section allows administrators to specify a bash one-liner which will help the container stay alive for the duration of a competition. If an infinite loop is not specified in the entrypoint the Docker container may die when deployed. It is up to administrators to determine whether they need to specify an entrypoint or not. The reset penalty allows administrators to specify how many points to deduct from a players score when their gaming box is reset to the default config. A players container may be reset if a competitors VNC process is killed or an attacker decides to rm -rf / a competitors gaming box.

***CORE Home Page***

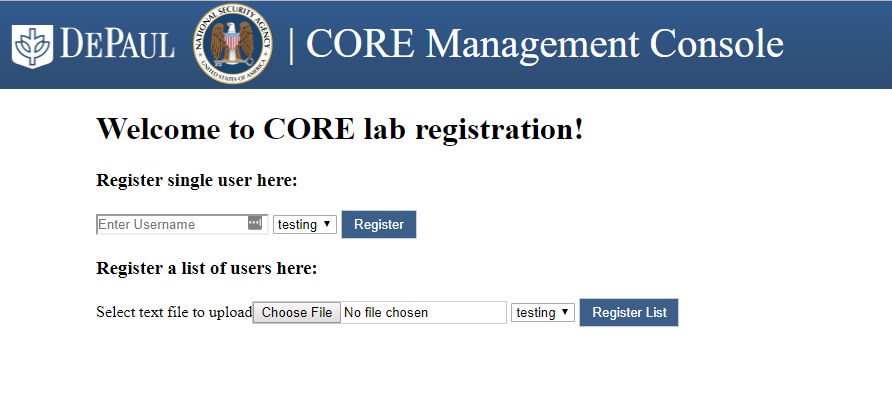
From the homepage, administrators and educators are able to see all running competitions, labs, or final exams. Running exercises will show up as green while not running or newly created competitions will show up in red as shown in Figure 7.



*Figure 7: CORE Active/Inactive Exercises*

***Lab Registration Page***

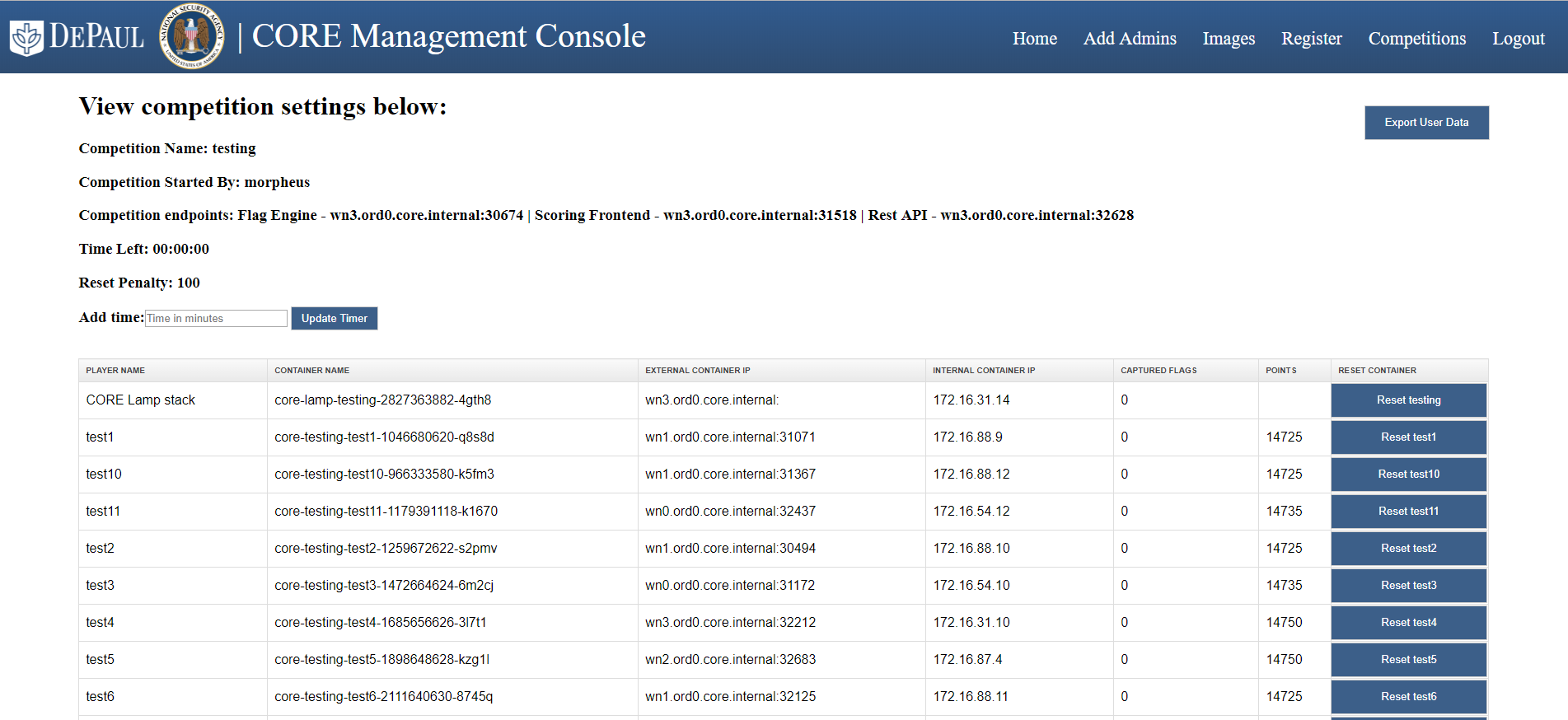
Administrators and educators can register either a single user or a list of user as shown in Figure 8. The list of users is separated by a newline.



*Figure 8: CORE User Registration*

***Competition Management Page***

For active exercises, administrators and educators can manage configurations and view endpoint mapping details for each competitor/student and CORE gaming box. The competition management page allows administrators to view competition endpoints (flag engine, competition scoring front-end, and REST API), reset containers, export user data, and add time during exercises, as shown in Figure 9. The export user data button will download a CSV file containing all user data seen in the table.

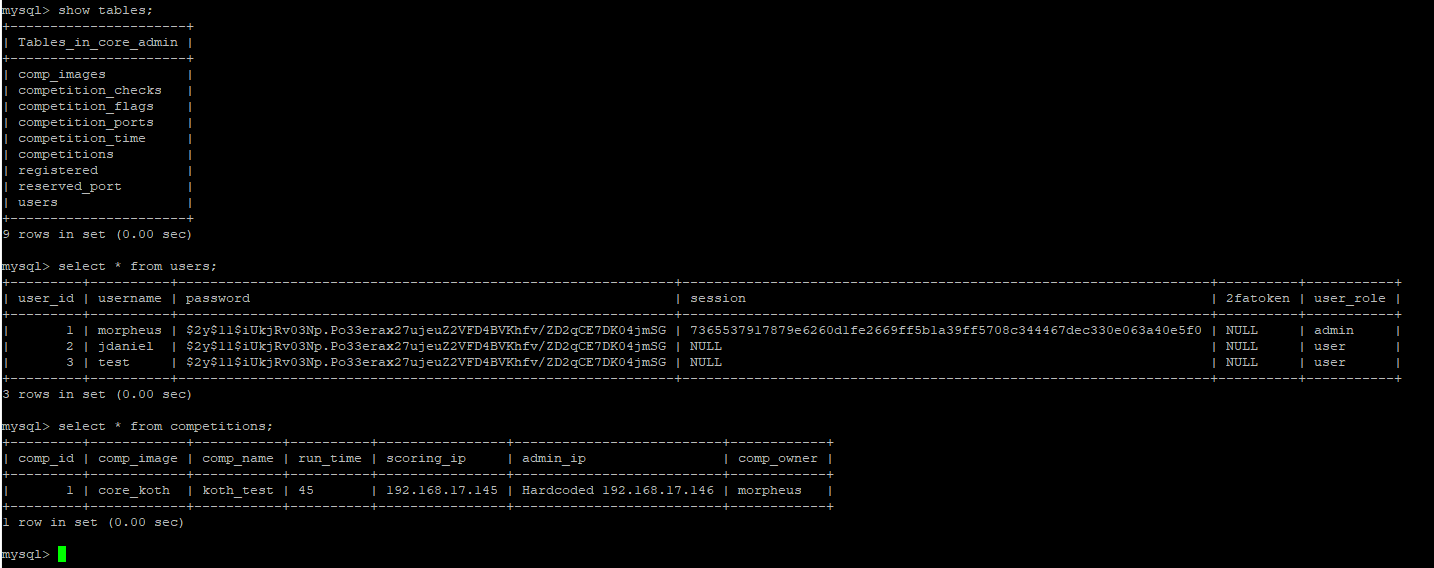


*Figure 9: CORE Competition Management Page.*

If an administrator would like to run commands within any of these containers it is recommended that they do so through the Kubernetes Cockpit, Kubernetes Dashboard, or directly on their Kubernetes worker nodes. The container name displayed in the competition management page helps with mapping which container an administrator should be runnings commands on.

### CORE Administrator Database

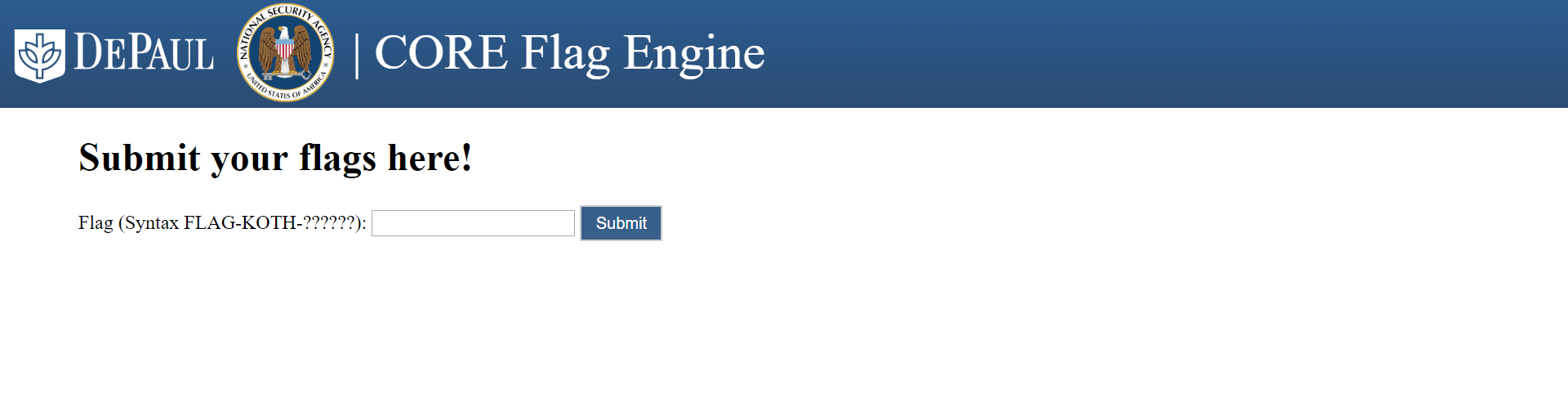
The CORE administration engine has its own database for managing exercises and users shown in Figure 10, if administrators prefer to make modifications outside of the CORE Management Console.



*Figure 10: CORE Admin Database.*

## Scoring Engine

The scoring of exercises in CORE encompasses: (1) capturing other competitor’s flags (attacking); (2) finding own flags and hardening own machines (defending); and (3) completing forensic tasks. A flag can be a simple value stored within a file or a process on the CORE gaming box. Each flag should be weighted depending on the level of difficulty it takes to find, and its point value is determined with the exercise configuration. Flags can be tied to a specific competition objective, lab learning objective, or a final exam. CORE users can submit their flags to the flag engine in exchange for points, as shown in Figure 11.



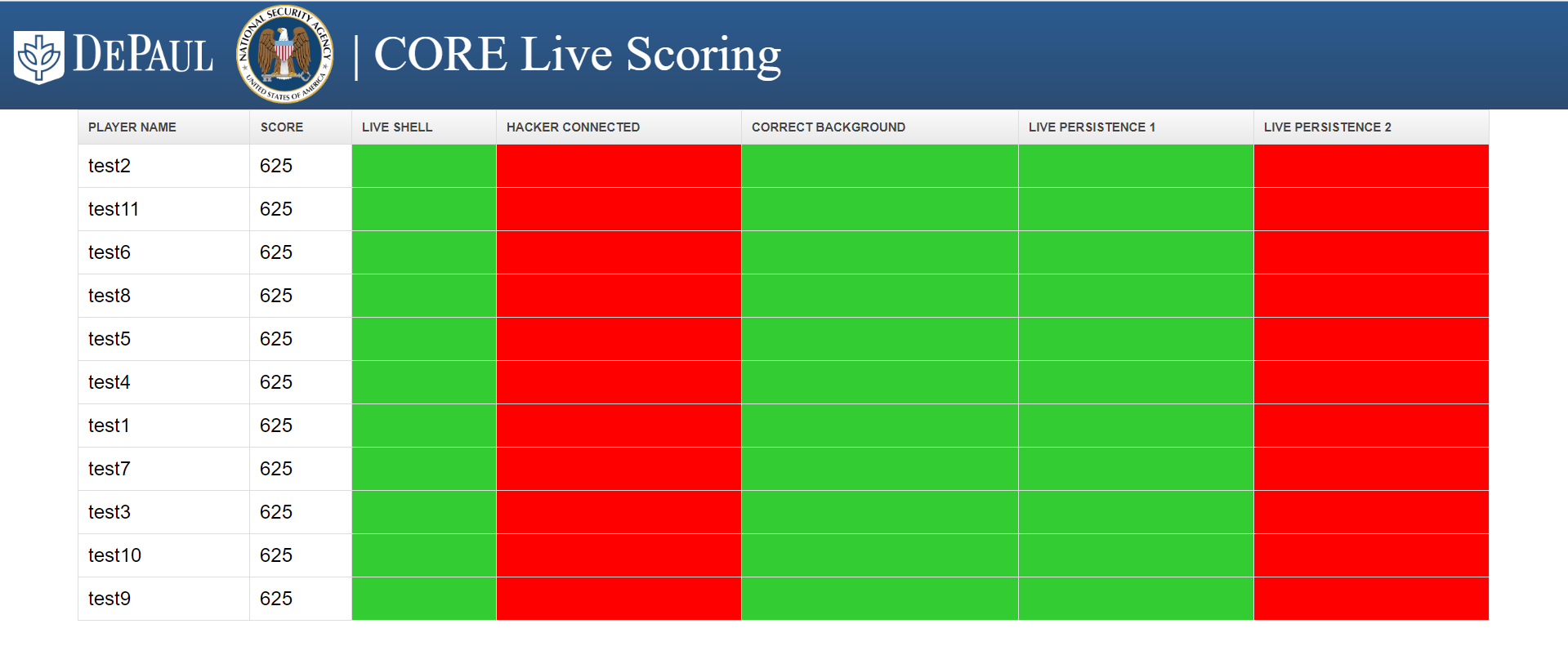
*Figure 11: CORE Flag Engine.*

The flag values within the competition configuration file will cause the CORE Management Console to generate a random flag value which will be added to a text file on each CORE gaming box within /root/scripts/flags.txt. It is up to the administrator to determine how to parse each flag from flags.txt and deploy it within each CORE gaming box. A sample script named deploy\_flags.py is provided within the KoTH sample lab.

In addition to flags, each competitor is scored based on a set of checks. These checks are conducted every second in order to maintain an accurate status of each competitors standing. Checks determine whether a competitor has successfully compromised another gaming box, is properly defending their gaming box, or has detected any persistence that has been deployed on their gaming box. Below are example checks provided in the example completion showcased above:

* **Live Shell -** Detects whether a *netcat* process is running on port 1337. This check is meant to help competitors identify peers that have not killed the *netcat* process.
* **Hacker connected -** Detects whether a competitor has been compromised. A competitor is considered compromised when a connection is established with the *netcat* process.
* **Correct Background -** An attackers goal for this competition is to change other competitors’ desktop background. They may do so once they are connected to the *netcat* process.
* **Live Persistence 1 -** Detects whether a competitor has found the *cronjob* which is starting the *netcat* process every minute
* **Live Persistence 2 -** Detects whether a competitor as found the malicious script that starts the *netcat* process. A forensics flag is hidden within this script and users must submit it to get the maximum points.

A full cycle of these checks may be seen in Figure 12. The checks are  configured to give points to an attacker if they successfully change a competitors background or connect to a process. Additionally, a competitor is rewarded points if a Live Shell, Live Persistence 1, or Live Persistence 2 check fails. If one of these checks fail it means that a competitor has successfully detected and removed malicious content from their gaming box. A player is also rewarded points for maintaining the correct background on their gaming box.



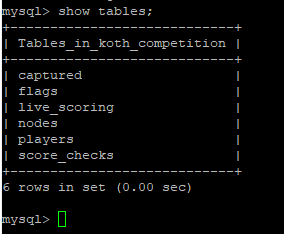
*Figure 12: CORE Scoring Engine - Front End.*

The CORE Administration Engine will submit competition data to the scoring engine over a TCP socket. The data contained within the competition is the REST API IP address and port, duration of the competition, players competing in competition, and the checks to be scored during the competition.

Once this data is received the TCP connection is closed and the scoring engine awaits data to start scoring a new competition. The scoring engine is multi-threaded to be able to handle simultaneous competition requests. Once the scoring engine closes the TCP connection with the CORE Administration Engine it will set the competition duration through a GET request to the REST API. When this is completed, the scoring engine will create a pool of processes, 1-1 mapping for process-container, and continue to conduct checks on the containers until the competition timer reaches 00:00:00. Once each check is complete it updates a users score by querying the REST API within the CORE LAMP stack container with new scores. A scoring cycle is complete when every container has undergone all checks specified for a competition. As mentioned earlier, each Kubernetes worker node has a scoring engine engine running on it. There are many factors which may affect the speed of scoring on each worker node. To ensure that scoring is done fairly each scoring engine will notify the CORE LAMP stack container’s REST API that it has completed a cycle. Afterwards, it will wait until all worker nodes have completed their check cycles and notified the REST API. Each scoring engine does so by querying an endpoint on the REST API which will notify the engine when it may begin scoring again. This enables administrators and educators to host multiple exercises concurrently and have a fair real-time scoring overview for each competition.

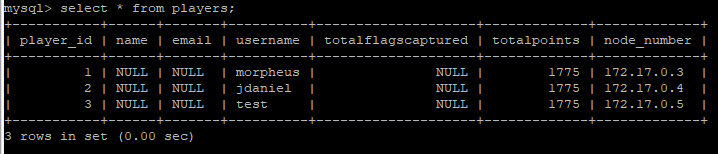
## Competition Database

For each exercise, CORE provisions a competition database within the competition LAMP stack. The tables in the this database are shown in Figure 13(a) and contain the player identification (as uploaded by the administrators and educators and further shown in Figure 13(b)), the total flags, the number of captured flags for each player, the live scoring, and the scoring checks.



*Figure 13(a): CORE Competition Database - All Tables.*

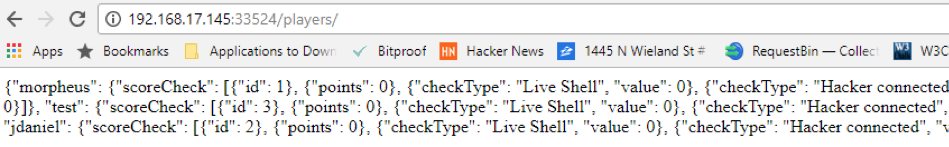
The captured table keeps track of which competitor has submitted a flag. This is to ensure that a competitor does not submit multiple flags at once. The flags table keeps track of all competition flags deployed for a competition, for which competitor they were generated, and the point values associated with the flag. This table is referenced by the Flag Engine when verifying whether a submitted flag is legitimate. The live\_scoring table keeps track of the current check status for each competitor. The nodes table keeps track of the internal IP address of each competitor. The score\_checks table keeps track of the id and name of each scoring engine check, and it is shown on Figure 13 above.



*Figure 13(b): CORE Competition Database - Players Table.*

## RESTful **API**

The RESTful API allows the scoring engine to update players scores. The REST API is located within each LAMP stack for every competition. An example scoring update is shown in Figure 14.

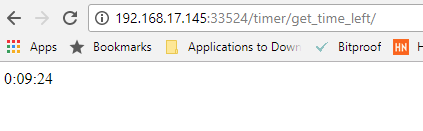


*Figure 14: CORE RESTful API - Player Data in JSON format.*

The REST API handles all REST API calls through get requests. The REST API connects to the competition management database to make any modifications to the tables within the database. Outlined below are the REST API calls which may be called.

* Update a competitors check status: [http://base\_url:rest\_api\_port/players/playername/check\_num/status](http://base_url/players/playername/check_num/status)
  + base\_url is the IP address of the node where the competition management container is located
  + rest\_api\_port is the port number of the REST API for the competition
  + playername is the competitor’s username
  + check\_num is the check number which is referenced within the competition database
  + status is a binary 0 or 1. 0 means that the resulting check will turn red, 1 means that the resulting check will turn green.
* Update a competitors points: [http://base\_url:rest\_api\_port/players/playername/points/](about:blank)
  + points is the amount of points which should be added to a competitors score
* Start a competition by setting the duration: [http://base\_url:rest\_api\_port/timer/startTime/time\_in\_minutes](about:blank)
  + time\_in\_minutes is the amount of time in minutes to start the competition with
* Add time to a competition: [http://base\_url:rest\_api\_port/timer/addTime/time\_in\_minutes/](about:blank)
* Check how much time is left for a competition:  [http://base\_url:rest\_api\_port/timer/get\_time\_left/](about:blank)
* Scoring engine cycle completion notification:  [http://base\_url:rest\_api\_port/cycle/validate/node\_number/](about:blank)
* Scoring engine check in endpoint which allows node to continue scoring once all nodes have checked in:  [http://base\_url:rest\_api\_port/cycle/checkin/node\_number/](about:blank)

In CORE, the traditional attack-defend rounds are replaced with two 45-minute rounds and one 15-minute intermission. The intermission is introduced to recap with the competitors. Because exercises are time-bounded, CORE allows for educators, students, or competitors to periodically check the by maintaining a timer, shown in Figure 15. CORE administrators and educators can setup different times for their exercises as part of the configuration they need to upload in the CORE administration engine.



*Figure 15: CORE Timer*

## Troubleshooting

Core Admin uses PHP 5.5. If problems arise try changing your PHP version to 5.5.

The upload directory doesn’t exist initially, and must be created. The permissions of the upload folder should also be changed from root to apache.

Files paths must be relative rather than static.

If using MySQL version 8.0, run query: “CREATE USER root@localhost identified with mysql\_native\_password by 'password';”

1. <https://support.google.com/accounts/answer/1066447?co=GENIE.Platform%3DAndroid&hl=en> [↑](#footnote-ref-0)