Challenge Type Plugins

Challenge Types

In CTFd, there is a concept of a type of challenge. Most CTFs only ever provide challenges as a snippet of text alongside some files. CTFd expands upon this and allows developers to create new challenge types which diversify what users will see.

Ultimately, users will still read some text, and submit some value but CTFd allows you to style and customize this so users can submit data in new ways.

For example, instead of an input to submit a single flag value, you might require teams to submit multiple flags or you might create some kind of customized UI where teams need to arrange blocks or text in some order.

The approach used by CTFd here is to give each "type" of challenge an ID and a name.

> **TIP**
>
> You can see how CTFd implements its default standard challenge here. You can also see how CTFd implements dynamic scoring using this feature.

Each challenge is implemented as a child class of the `BaseChallenge` and implements static methods named `create`, `read`, `update`, `delete`, `attempt`, `solve`, and `fail`.

When a user attempts to solve a challenge, CTFd will look up the challenge type and then call the `solve` method as shown in the following snippet of code:

```
chal_class = get_chal_class(chal.type)
status, message = chal_class.attempt(chal, request)

if status:  # The challenge plugin says the input is right
```

```
    if ctftime() or is_admin():
        chal_class.solve(team=team, chal=chal, request=request)
    return jsonify({'status': 1, 'message': message})

else:  # The challenge plugin says the input is wrong
    if ctftime() or is_admin():
        chal_class.fail(team=team, chal=chal, request=request)
```

This structure allows each Challenge Type to dictate how they are attempted, solved, and marked incorrect.

The Challenge Type also dictates the database table that it uses to store data. By default this uses the `type` column as a `polymorphic_identity` to implement table inheritance. Effectively each child table will use the Challenges table as a parent. The child table can add whatever columns it wishes but still leverage the existing columns from the parent.

We can see in the following code that the polymorphic_identity is specified to be `dynamic` as well as the `type` parameter. We can also see the call to `create_all()` which will create the table in our database.

```
class DynamicChallenge(Challenges):
    __mapper_args__ = {'polymorphic_identity': 'dynamic'}
    id = db.Column(None, db.ForeignKey('challenges.id'), primary_key=True)
    initial = db.Column(db.Integer)
    minimum = db.Column(db.Integer)
    decay = db.Column(db.Integer)

    def __init__(self, name, description, value, category, type='dynamic', minimum=1, d
        self.name = name
        self.description = description
        self.value = value
        self.initial = value
        self.category = category
        self.type = type
        self.minimum = minimum
        self.decay = decay


def load(app):
    app.db.create_all()
    CHALLENGE_CLASSES['dynamic'] = DynamicValueChallenge
    register_plugin_assets_directory(app, base_path='/plugins/DynamicValueChallenge/ass
```

This code creates the necessary tables for the Challenge Type plugin which should be used in addition to the staticmethods used to define the challenge's behavior.

Every challenge type must be added to the global dictionary that specifies all challenge types:

```
CHALLENGE_CLASSES = {
    "standard": CTFdStandardChallenge
}


def get_chal_class(class_id):
    cls = CHALLENGE_CLASSES.get(class_id)
    if cls is None:
        raise KeyError
    return cls
```

The Standard Challenge type provided within CTFd can be used as a base from which to build additional Challenge Type plugins.

Once new challenges are registered, CTFd will provide a dropdown allowing you to choose from all the challenge types you can create.

Each Challenge Type contains templates and scripts dictionaries which contain the routes for HTML and JS files needed for the operation of the modals used to create and update the challenges.

These routes are not automatically defined by CTFd.

Each challenge type plugin specifies the location of their own templates and scripts. An example is the built in standard challenge type plugin. It specifies the URLs that the assets are located at for the user's browser to load:

```
templates = {  # Templates used for each aspect of challenge editing & viewing
    'create': '/plugins/challenges/assets/create.html',
    'update': '/plugins/challenges/assets/update.html',
    'view': '/plugins/challenges/assets/view.html',
}
scripts = {  # Scripts that are loaded when a template is loaded
    'create': '/plugins/challenges/assets/create.js',
```

```
    'update': '/plugins/challenges/assets/update.js',
    'view': '/plugins/challenges/assets/view.js',
}
```

These files are registered with Flask with the following code:

```
from CTFd.plugins import register_plugin_assets_directory

def load(app):
    register_plugin_assets_directory(app, base_path='/plugins/challenges/assets/')
```

The aforementioned code handles the Python logic around new challenges but in order to fully integrate with CTFd you will need to create new Nunjucks templates to give admins/teams the ability to modify/update/solve your challenge. The templates used by the Standard Challenge Type should serve as examples.

Previous
« Developing CTFd Plugins

Next
Flag Type Plugins »