

16th April 2019

1. Given the shadow file that you pulled from the DJI Phantom 3 Controller, please uncover the salt and demonstrate the process that modern Unix/Linux systems use to store and retrieve passwords. Use the actual DJI Phantom 3 password in your documentation

The Shadow file is a file that only available to root users as it stores users password in an encoded form. It is stored in /etc/shadow. When a user logs in the system should hash the password logged in with using the information in the shadow file and compare it the stored value in the shadow file to check if it matches.

The file record user's password in the following format:

```
vivek:$1$fnfffc$PgTeyHdicpGOfffXX4ow#5:13064:0:99999:7::
```

1 2 3 4 5 6

1. Is the user name
Between 1 and 2 is \$the_id_of_the_hash_encryption:
 \$1\$ is MD5(in our case above)
 \$2a\$ is Blowfish
 \$2y\$ is Blowfish
 \$5\$ is SHA-256
 \$6\$ is SHA-512
2. Is the password for the user in 1. The format of this field is as follows:
 \$salt\$hashed_password
3. Is the number of days since Jan 1, 1970 that the password was changed
4. Is the minimum number of days required between password changes
5. Is the maximum number of days that the password is valid
6. Is the number of days before the password expires and the user is warned that the password must be changed

Usually, the salt value is appended to the plaintext password and then the result is hashed:

hash(password || salt). Note that both password and salt are strings.

In our case the shadow file is as follows. We will focus on the first row in the shadow file that we retrieved on the ftp server running on the white controller:

```
root:$1$Sh6N5Lf0$S3YjgK5WoI2Vty2XJjETm1:16534:0:99999:7:::
```

```
daemon:*:0:0:99999:7:::
```

```
ftp:*:0:0:99999:7:::
```

```
network:*:0:0:99999:7:::
```

```
nobody:*:0:0:99999:7:::
```

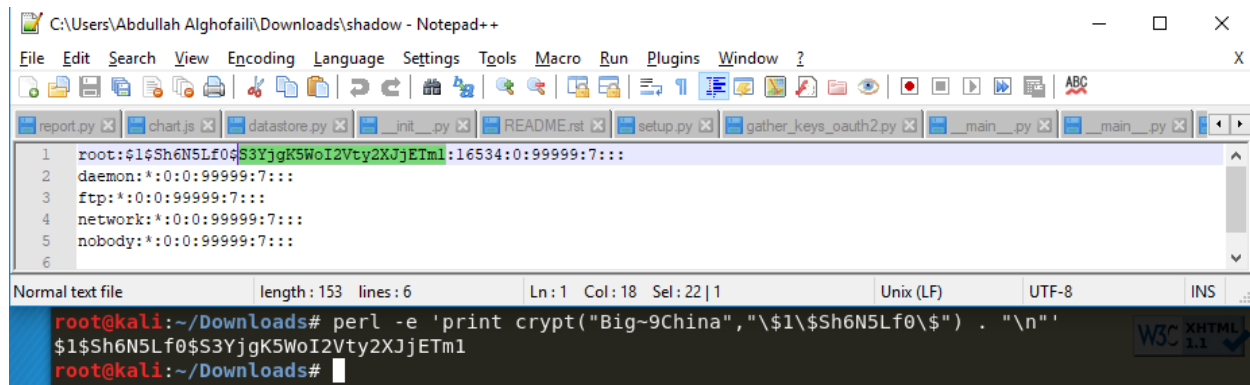
This entry (root:\$1\$Sh6N5Lf0\$S3YjgK5WoI2Vty2XJjETm1:16534:0:99999:7:::) present the user “root” account, and using the past demonstration the password of the account is going to be as follows:

1. The user name is “root”
2. The password information is:
 - a. The id of the encryption is “1” = “MD5”
 - b. The salt is “Sh6N5Lf0”
 - c. The hashed password, h(password || salt) is “S3YjgK5WoI2Vty2XJjETm1”

Note: Both password and salt are strings. Salt is a random string and it is appended to the password, which is then used to calculate hash value. We are hashing a longer, appended string.

3. “16534” is the number of days since Jan 1, 1970 that the password was changed
4. “0” is the minimum number of days required between password changes
5. “99999” is the maximum number of days that the password is valid
6. “7” Is the number of days before the password expires and the user is warned that the password must be changed

We verified the password of the “root” account using a perl script that will use the salt and the MD5 hash alongside the “Big~9China” password to produce the hashed password. In Figure 1, the resulted hash is identical to the hashed password from the shadow file, meaning the password in a plain form is indeed “Big~9China”.



The screenshot shows a Notepad++ window titled 'C:\Users\Abdullah Alghofaili\Downloads\shadow - Notepad++'. The window contains a shadow file with the following entries:

```
1 root:$1$Sh6N5Lf0$S3YjgK5WoI2Vty2XJjETm1:16534:0:99999:7:::
2 daemon:*:0:0:99999:7:::
3 ftp:*:0:0:99999:7:::
4 network:*:0:0:99999:7:::
5 nobody:*:0:0:99999:7:::
6
```

Below the Notepad++ window, a terminal window shows the execution of a perl script to verify the password for the root user:

```
root@kali:~/Downloads# perl -e 'print crypt("Big~9China","\$1\$Sh6N5Lf0\$") . "\n"'
$1$Sh6N5Lf0$S3YjgK5WoI2Vty2XJjETm1
root@kali:~/Downloads#
```

Figure 1

Hashcat part to recover the password from the shadow file:

Step 1:

Inside Kali by default, there is a folder called rockyou.txt.gz under the /usr/share/wordlists folder:

```
root@kali:/usr/share/wordlists# pwd
/usr/share/wordlists
root@kali:/usr/share/wordlists# ls
dirb dirbuster dnsmap.txt fasttrack.txt fern-wifi metasploit nmap.lst rockyou.txt.gz sqlmap.txt wfuzz
```

Step 2:

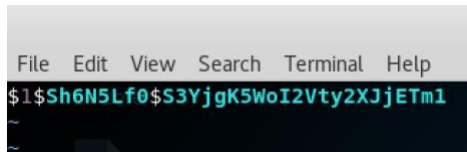
Now, I will unzip rockyou.txt to extract rockyou.txt. This is place where I will crack the password:

```
root@kali:/usr/share/wordlists# gzip -d rockyou.txt.gz
root@kali:/usr/share/wordlists# ls
dirb dirbuster dnsmap.txt fasttrack.txt fern-wifi metasploit nmap.lst rockyou.txt sqlmap.txt wfuzz
```

Step 3: We had to make sure that the real password is inside rockyou.txt, which is the password input space. If the real password string is not in this rockyou.txt, the successful association between the real

password and the hash will NOT happen. In other words, we had to ensure that Big~9China is in rockyou.txt.

Step 4: Now, we will create an input file called hash_output.lst which contains the type of hash, salt, and hash value(of both the salt and password). Note that hashcat needs this file to be in this format:

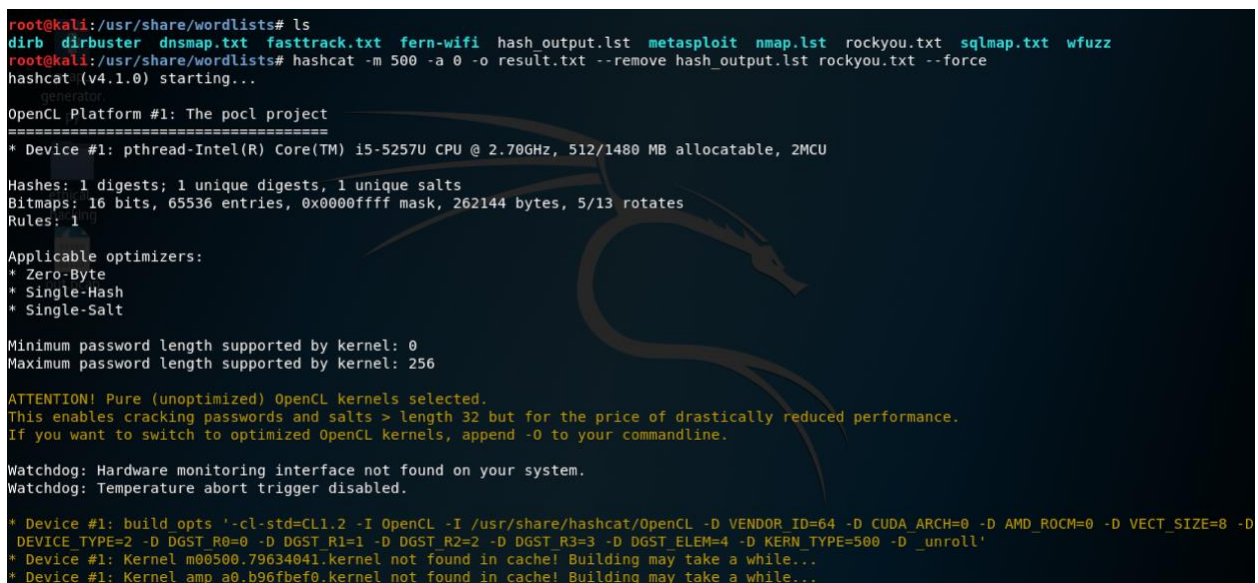


```
File Edit View Search Terminal Help
$1$Sh6N5Lf0$S3YjgK5WoI2Vty2XJjETm1
```

Note: This file will be depleted and emptied after a successful cracking(after step 6)

Step 5:

Now, we are ready to find the specific password in our list of password(aka input space) for this specific hash from step 4. -m is the mode, -o is the output file called result.txt. Now, we use hashcat to get the password. Because \$1 indicates MD5, so I used mode 500.



```
root@kali: /usr/share/wordlists# ls
dirb dirbuster dnsmap.txt fasttrack.txt fern-wifi hash_output.lst metasploit nmap.lst rockyou.txt sqlmap.txt wfuzz
root@kali: /usr/share/wordlists# hashcat -m 500 -a 0 -o result.txt --remove hash_output.lst rockyou.txt --force
hashcat (v4.1.0) starting...
generator
OpenCL Platform #1: The pocl project
=====
* Device #1: pthread-Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz, 512/1480 MB allocatable, 2MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Single-Hash
* Single-Salt

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

ATTENTION! Pure (unoptimized) OpenCL kernels selected.
This enables cracking passwords and salts > length 32 but for the price of drastically reduced performance.
If you want to switch to optimized OpenCL kernels, append -O to your commandline.

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

* Device #1: build opts '-cl-std=CL1.2 -I /usr/share/hashcat/OpenCL -D VENDOR_ID=64 -D CUDA_ARCH=0 -D AMD_ROCM=0 -D VECT_SIZE=8 -D
DEVICE_TYPE=2 -D DGST_R0=0 -D DGST_R1=1 -D DGST_R2=2 -D DGST_R3=3 -D DGST_ELEM=4 -D KERN_TYPE=500 -D _unroll'
* Device #1: Kernel m00500.79634041.kernel not found in cache! Building may take a while...
* Device #1: Kernel amp_a0.b96fbef0.kernel not found in cache! Building may take a while...
```

```

Dictionary cache built:
* Filename.: rockyou.txt
* Passwords.: 14344393
* Bytes.: 139921518
* Keyspace.: 14344386
* Runtime.: 3 secs

out.pcap

Session.....: hashcat
Status.....: Cracked
Hash.Type.....: md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5)
Hash.Target.....: $1$Sh6N5Lf0$S3YjgK5WoI2Vty2XJjETm1
Time.Started.....: Thu Apr 18 12:41:35 2019 (0 secs)
Time.Estimated...: Thu Apr 18 12:41:35 2019 (0 secs)
Guess.Base.....: File (rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....: 5022 H/s (5.57ms) @ Accel:256 Loops:62 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 512/14344386 (0.00%)
Rejected.....: 0/512 (0.00%)
Restore.Point....: 0/14344386 (0.00%)
Candidates.#1....: 123456 -> brandy
HWMon.Dev.#1.....: N/A

Started: Thu Apr 18 12:41:24 2019
Stopped: Thu Apr 18 12:41:36 2019

```

Now, the cracking has been successful.

Step 6:

```

Stopped: Thu Apr 18 12:41:36 2019
root@kali: /usr/share/wordlists# cat result.txt
$1$Sh6N5Lf0$S3YjgK5WoI2Vty2XJjETm1:Big~9China

```

As shown in result.txt, the password for **\$1\$Sh6N5Lf0\$S3YjgK5WoI2Vty2XJjETm1** is Big~9China.

References

<https://www.cyberciti.biz/faq/understanding-etcshadow-file/>

[https://en.wikipedia.org/wiki/Salt_\(cryptography\)](https://en.wikipedia.org/wiki/Salt_(cryptography))

<https://stackoverflow.com/questions/9594125/salt-and-hash-a-password-in-python>