

# Using hashing to confirm a file download

The site dftt.sourceforge.net is the Digital Forensics Tool Testing image repository. This site contains 14 forensic images that can be used to test forensic analysis tools. These images are also useful in practicing and developing skills in using forensic tools and techniques before working on actual crime scene collected evidence. This is just one of many similar repositories of forensic image testing files.

**Digital Forensics Tool Testing Images**

Testing in the public view is an important part of increasing confidence in software and hardware tools. Developing extensive and exhaustive tests for digital investigation tools is a lengthy and complex process, which the [Computer Forensic Tool Testing \(CFTT\)](#) group at NIST has taken on.

To fill the gap between extensive tests from NIST and no public tests, I have been developing small test cases. The following are file system and disk images for testing digital (computer) forensic analysis and acquisition tools. They were submitted to the [Computer Forensic Tool Testing \(CFTT\)](#) e-mail list on Yahoo! Groups. The results can be found in the CFTT list archives and in the [Test Report Tracker](#).

**Test Images:**

- 1. [Extended Partition Test](#) (July '03)
- 2. [FAT Keyword Search Test](#) (Aug '03)
- 3. [NTFS Keyword Search Test #1](#) (Oct '03)
- 4. [EXT3FS Keyword Search Test #1](#) (Nov '03)
- 5. [FAT Daylight Savings Test](#) (Jan '04)
- 6. [FAT Undelete Test #1](#) (Feb '04)
- 7. [NTFS Undelete \(and leap year\) Test #1](#) (Feb '04)
- 8. [JPEG Search Test #1](#) (Jun '04)
- 9. [FAT Volume Label Test #1](#) (Aug '04)
- 10. [NTFS Autodetect Test #1](#) (Jan '05)
- 11. [Basic Data Carving Test #1](#) (Mar '05) (by Nick Mikus)
- 12. [Basic Data Carving Test #2](#) (Mar '05) (by Nick Mikus)
- 13. [Windows Memory Analysis #1](#) (Jan '06) (by Jesse Kornblum)
- 14. [ISO9660 Interpretation Test #1](#) (Aug '10)

**Test Results:**

- [Submission and Posting Requirements](#)
- [Report Tracker](#)

Copyright © 2005-2010 by Brian Carrier  
Email: carrier <at> digital-evidence <dot> org

Last Updated: August 11, 2010

## EXT3FS Keyword Search #1 -Digital Forensics Tool Testing Image (#4)

**Introduction**

This test image is an EXT3FS file system with several ASCII strings. There are only 4 strings to search for, so this one is quite simple and short. It only tests the basic features of EXT3FS.

**Download**

This test image is a 'raw' partition image (i.e. 'dd') of an EXT3FS file system. The file system is 5MB and is compressed to 4MB. The MD5 of the image is 30e7f792cc853e34e17335b243605d3a. This image is released under the [GPL](#), so anyone can use it.

- [ZIP](#)

**Search Terms**

These should all be performed case sensitive and not as regular expressions. [Results Form](#)

Num	String	Sector - Offset	Fragment - Offset	File	Note
1	first	330 - 100	165 - 100	/, /., /..., /lost+found/..	File Name
	first	392 - 100	196 - 100	inode #8	Journal entry
	first	432 - 100	216 - 100	inode #8	Journal entry
	first	2416 - 181	1208 - 181	/file1	Allocated file
2	second	2419 - 509	1209 - 1021	/file2	Fragmented String
3	third	2420 - 80	1210 - 80	/file3 (deleted)	Unallocated file
4	slacker	2417 - 179	1208 - 691	/file1	Slack space of file1

**Author**

Brian Carrier (carrier at cerias.purdue.edu) created the test cases and the test image. This test was released on November 24, 2003.

**Disclaimers**

Neither Purdue University or CERIAS sponsor this work.

These tests are not a complete test suite. These were the first ones that I thought of and no formal theory was put into their design.

Passing these tests provides no guarantees about a tool. Always use additional test cases (and email them to me so we can all benefit!).

**ls /media/cdrom0** (To view the contents of the DVD Drive)

```
cp /media/cdrom0/* /root/Downloads/ (To copy the forensic test image files to the /root/Download directory)
```

```
cd /root/Downloads (to change into the directory.)
```

```
ls -l
```

```
to view the contents and verify that the 4-kwsrch-ext3.zip file is present and is the size of 3954200 bytes.
```

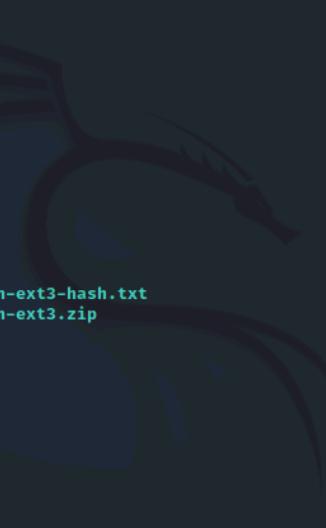
```
unzip 4-kwsrch-ext3.zip
```

```
to extract the contents of the zip archive into its default subdirectories.
```

```
ls -l 4-kwsrch-ext3
```

```
to view the contents of the new subdirectories.
```

There should be four files. One of them is ext3-img-kw-1.dd, which should be 5242880 bytes in size.



```
root@kali: ~/Downloads
File Actions Edit View Help
└─# ls media/cdrom0/
ls: cannot access 'media/cdrom0/': No such file or directory

└─(root@kali)─[~]
└─# ls /media/cdrom0/
4-kwsrch-ext3-hash.txt  4-kwsrch-ext3.zip

└─(root@kali)─[~]
└─# dvbvdvv~v
dvbvdvv~v: command not found

└─(root@kali)─[~]
└─# cp /media/cdrom0/* /root/Downloads/

└─(root@kali)─[~]
└─# cd /root/Downloads/
└─# ls -l
total 3868
-r-xr-xr-x 1 root root      34 Nov 22 06:25 4-kwsrch-ext3-hash.txt
-r-xr-xr-x 1 root root 3954200 Nov 22 06:25 4-kwsrch-ext3.zip

└─(root@kali)─[~/Downloads]
└─# unzip 4-kwsrch-ext3.zip
Archive: 4-kwsrch-ext3.zip
  inflating: 4-kwsrch-ext3/COPYING-GNU.txt
  inflating: 4-kwsrch-ext3/README.txt
  inflating: 4-kwsrch-ext3/ext3-img-kw-1.dd
  inflating: 4-kwsrch-ext3/index.html

└─(root@kali)─[~/Downloads]
└─# ls -l 4-kwsrch-ext3
total 5148
-rw-r--r-- 1 root root 18009 Nov 23 2003 COPYING-GNU.txt
-rw-r--r-- 1 root root 5242880 Nov 23 2003 ext3-img-kw-1.dd
-rw-r--r-- 1 root root    2810 Apr  4 2004 index.html
-rw-r--r-- 1 root root     766 Apr  4 2004 README.txt

└─(root@kali)─[~/Downloads]
└─#
```

Most binary numbers are converted to hex when presented on the screen.

This is convenient as it uses 1/4 the number of characters to represent the same value.

This is because four digits in binary can be used to represent the decimal values of 0 when all the bits are zero through 15 when all the bits are ones.

These are the same ranges of values for a single hex character (i.e., 0-9 & A-F (10-15)).

With that information, you can count up the number of hex characters in a hash, then multiply by four (4) to determine the hash length.

Then, by knowing the hash length, you can make a reasonable guess as to the hashing algorithm used. Here is a brief chart of common hashing algorithms:

Algorithm	Bit length	Hex length
MD5	128	32
SHA-1	160	40
SHA-224	224	56
SHA-256	256	64
SHA-384	384	96
SHA-512	512	128

What is the bit length of an MD5 hash?

160

128

64

32

256

In the Terminal window, enter **ls -l** to view the contents of the extracted files.

```
cat 4-kwsrch-ext3-hash.txt.
```

*This file contains the hash of the ext3-img-kw-1.dd drive image file.*

It was copied directly from dftt.sourceforge.net. This should be the same hash value you viewed on the website for the drive image file.

Enter the following:

```
md5sum 4-kwsrch-ext3/ext3-img-kw-1.dd >> 4-kwsrch-ext3-hash.txt
```

*This command hashes the file and adds that hash to expected hash file.*

```
cat 4-kwsrch-ext3-hash.txt
```

```
[root@kali] ~]# md5sum 4-kwsrch-ext3/ext3-img-kw-1.dd >> 4-kwsrch-ext3-hash.txt
[root@kali] ~]# cat 4-kwsrch-ext3-hash.txt
30e7f792cc853e34e17335b243605d3a
30e7f792cc853e34e17335b243605d3a 4-kwsrch-ext3/ext3-img-kw-1.dd
[root@kali] ~]#
```

The first hash was copied directly from dftt.sourceforge.net for the ext3-img-kw-1.dd file.

The second hash was added by hashing the file obtained from the DVD/ISO resource.

Compare the two hashes to verify that they match.

Since the two hashes match exactly, you have verified the file's integrity. The file offered by the website is the file present on the Kali system.

*The use of MD5 hash has been deprecated. This means it is no longer the preferred or recommended hashing algorithm to use to verify integrity. It is often better to use SHA-256, SHA-512, or even SHA-1. However, many sites and services still use MD5 in spite of this. So, you have to check the integrity of the files you download using whatever hashing algorithm the site uses. The process performed in this exercise would be the same. You need to switch to the hashing utility for the specific hashing algorithm needed, such as sha1sum, sha256sum, or sha512sum.*

Older hashing algorithms are not necessarily broken or compromised, but they are more prone to **collisions**. A collision is when two different datasets produce the same hash value (when hashed by the same hashing algorithm). This is a known aspect of hashing due to the fact that hash algorithms accept nearly infinite inputs to produce a hash of a fixed length. *Generally, algorithms that produce a shorter hash value output are more prone to collisions than those with longer hash value outputs.*

You have now used hash matching to confirm the integrity of a download file.

This same process can be used to verify the integrity of any file from any source as long as you know the correct/expected hash and the hashing algorithm used to calculate that hash.

## Use the MetaDefender to evaluate files

You have discovered a file that you are concerned about on a client workstation. The user claims not to know anything about the file. You think that it could be a hacker tool and possibly malicious, but you want to confirm that before initiating a system wipe and rebuild. You decide to perform an initial evaluation of the file via hash identification. In this exercise, you will perform suspicious file analysis through an online evaluation service.

Connect to the KALI virtual machine and sign in as root using Pa\$\$w0rd as the password.

Open a Terminal window and then maximize the Terminal window.

Enter `cd /usr/share/windows-resources/binaries` to change into the folder where the suspicious file is located.

Enter `ls -l` to view a long listing of the contents of the directory.

The file of concern is nc.exe.

Create a SHA1 hash value from the file by entering: `sha1sum nc.exe`.

**sha1sum [filename]**

Result should be: 57F0839433234285CC9DF96198A6CA58248A4707

The screenshot shows the MetaDefender Cloud web interface. At the top, there is a dark header with the text "Upload any file, URL, IP, or hash for scanning and online malware analysis. Detect ransomware with 20+ antivirus engines, inspect files with the Adaptive Sandbox, and neutralize threats with Deep CDR. [Learn more...](#)". Below the header are four buttons: "Scan a file", "Search a URL", "Lookup", and a button with a star icon. The main area has a dark background with the text "Trust no Hash" in white. A search bar contains the SHA1 hash "57F0839433234285CC9DF96198A6CA58248A4707". To the right of the search bar is a "Process" button. At the bottom of the main area, a small note reads: "By submitting a file to MetaDefender Cloud, you are granting permission to share the results of your submission with the cybersecurity community and agreeing to the [Terms of Service](#) and [Privacy Policy](#)". At the very bottom of the page, there is a footer note: "① Access our [docs](#) and [your API key](#) to automate scans and file analysis".

The screenshot shows a file analysis interface for the file NC.EXE. At the top, there are several status indicators: Multiscanning (19 Threats Detected), Adaptive Sandbox (No Results Available), Deep CDR (No Sanitization Available), Proactive DLP (No Results Available), and Vulnerabilities (No Vulnerabilities Found). Below these are sections for Community feedback (0 comments) and a file hash (534c12cd57b7e56b01a... 7abb5).

**Archive Extraction:** Shows the file hash 534c12cd57b7e56b01a... 7abb5 and a download link for NC.EXE.

**Multiscanning:** Shows 19 engines with threats detected. The engines and their verdicts are:

Engine Name	Verdict
AhnLab	Win-AppCare/NTSniff_v110
Antiy	Trojan[APT]/Win32.Equation
Avira	BDS/Backdoor.Gen
Bkav Pro	W32.Malware.CB97C1A4
CMC	HackTool.Win32_RemoteAdmin_MSR
Xcitium	ApplieUnsafe.Win32_RemoteAdmin
Aurora	Malware_-10
ESET	a variant of Win32/RemoteAdmin.NetCat.AB1
Huorong	HackTool/NetCat
K7	Trojan (005970a11)

**File Overview:** Provides detailed file metadata:

Category	E	Scanned	GMT
File Type	Executable File	Duration	a few seconds
File Extension	exe	MD5	E0F946C... C22A1
TrID	-	SHA-1	57F08394... A4707
LibMagic	-	SHA-256	BE4211FE... 9421B
Magika	-	Company Name	-
File Size	59.4 kB	File Description	-
Uploaded	-	File Version	-
SSDEEP	-	Internal Name	-
Architecture	-	Legal Copyright	-
Is DotNet	-	Original File Name	NC.EXE
Is Packed	-	Product Name	-
Is Digitally Signed	-	Product Version	-
Entropy	-		

The results page should indicate that this hash is related to the nc.exe file.

Also notice that some, but not all, scanning engines detected something concerning about the file. Based on this information, you could make a decision on whether this suspicious file is malicious, a **potentially unwanted program (PUP)**, or benign.

Just because an analysis engine lists a file as a threat does not actually mean it is malicious code. It is possible that the file is an administrative utility or even a hacker tool that could be used for malicious purposes, but is not inherently malicious. For example, a network sniffer is a useful tool, but not something you would want your typical user to have on their system. The same is true for keystroke loggers and password crackers. Not all analysis engines include PUPs in their detection databases.

Perform additional searches for other files. You can use file names or hashes from your own system, or you can return to the Kali VM and obtain file names and hashes from there.

For example, you could search for information on the klogger.exe file using the SHA1 hash of: 196BF6F43F85F97CC2851C840DA8E451256995CB.

**EXE** malware\Phase1\malware6.exe | Not Available (Country of Origin) Add.COO

Multiscanning Threats Detected | Adaptive Sandbox No Results Available | Deep CDR™ No Sanitization Available | Proactive DLP No Results Available | Vulnerabilities No Vulnerabilities Found

Community feedback 0 comments 0 likes 0 dislikes

### Archive Extraction

File Path	Verdict	Last engine update
19ba9f2a24472a0aef2a7... 71f71	malware\Phase1\ma... 2.exe	22
19ba9f2a24472a0aef2a7... 9.exe	malware\Phase1\m... 9.exe	17
19ba9f2a24472a0aef2a7... 7.exe	malware\Phase1\ma... 7.exe	28
19ba9f2a24472a0aef2a7... 3.exe	malware\Phase1\ma... 3.exe	21
19ba9f2a24472a0aef2a7... 1.exe	malware\Phase1\m... 1.exe	17
19ba9f2a24472a0aef2a7... 4.exe	malware\Phase1\ma... 4.exe	26
19ba9f2a24472a0aef2a7... 8.exe	malware\Phase1\m... 8.exe	20
19ba9f2a24472a0aef2a7... 1.exe	malware\Phase1\ma... 1.exe	5
19ba9f2a24472a0aef2a7... 6.exe	malware\Phase1\ma... 6.exe	23
19ba9f2a24472a0aef2a7... 5.exe	malware\Phase1\ma... 5.exe	26
19ba9f2a24472a0aef2a7... 0.exe	malware\Phase1\m... 0.exe	17

### Multiscanning 22 /30 ENGINES

Engine Name	Verdict	Last engine update
AegisLab	Trojan.Win32.Klogger.tr9g	07/18/2023 00:51 AM GMT
AhnLab	Trojan/Win32.Klogger	07/17/2023 19:00 PM GMT
Antiy	Trojan[Spy]Win32.Klogger	07/18/2023 05:17 AM GMT
Avira	TR/Klogger	07/17/2023 21:46 PM GMT
ClamAV	Win.Spyware.1752.2	07/17/2023 02:29 AM GMT
Emsisoft	Application.KeyLogger.QRZ (B)	07/18/2023 00:03 AM GMT
ESET	Win32/Spy.Klogger trojan	07/17/2023 21:51 PM GMT
IKARUS	Trojan-Spy.Win32.Klogger	07/17/2023 13:15 PM GMT
K7	Spyware (.005Se3db1)	07/17/2023 23:26 PM GMT
NANOAV	Trojan.Win32.Klog.dkjpot	07/17/2023 20:46 PM GMT

### File Overview

Category	E	Entropy	File Type	Scanned	GMT
File Extension	exe	-	File Type	Executable File	07/18/2023 01:24 AM
TrID	-	-	Duration	-	a few seconds
LibMagic	-	-	MD5	EBF2B608... 90A9	EBF2B608... 90A9
Magika	-	-	SHA-1	196BF64... 995CB	196BF64... 995CB
File Size	23.6 kB	-	SHA-256	CF2340B2... C54EB	CF2340B2... C54EB
Uploaded	07/18/2023 01:24 AM GMT	-	Company Name	-	-
SSDEEP	-	-	File Description	-	-
Architecture	-	-	File Version	-	-
Is DotNet	-	-	Internal Name	-	-
Is Packed	-	-	Legal Copyright	-	-
Is Digitally Signed	-	-	Original File Name	malw...el\malware6.exe	-
			Product Name	-	-
			Product Version	-	-

Sha1 sum performed on whoami.exe (hash was 8bcfd002d484fd03a0e95c66d80e0c64f65db73)

**EXE** Win RK 2000 full\NETMGMT\whoami.exe | Not Available (Country of Origin) Add.COO

Multiscanning No Threats Detected | Adaptive Sandbox No Results Available | Deep CDR™ No Sanitization Available | Proactive DLP No Results Available | Vulnerabilities Critical Vulnerabilities Found

Community feedback 0 comments 0 likes 0 dislikes

### Archive Extraction

File Path	Verdict	Last engine update
3335f851ea2f2b8d4666... 46ac0	Win RK 2000 full\NET... r.exe	1
19ba9f2a24472a0aef2a7... e.exe	Win RK 2000 full\DIAG... e.exe	✓
19ba9f2a24472a0aef2a7... g.dll	Win RK 2000 full\DIAG... g.dll	✓
19ba9f2a24472a0aef2a7... h.exe	Win RK 2000 full\DI... h.exe	✓
19ba9f2a24472a0aef2a7... i.exe	Win RK 2000 full\DI... i.exe	✓
19ba9f2a24472a0aef2a7... g.exe	Win RK 2000 full\FIL... g.exe	✓
19ba9f2a24472a0aef2a7... e.exe	Win RK 2000 full\FIL... e.exe	✓
19ba9f2a24472a0aef2a7... r.exe	Win RK 2000 full\NIS\h... r.exe	✓
19ba9f2a24472a0aef2a7... g.exe	Win RK 2000 full\DEP... g.exe	✓
19ba9f2a24472a0aef2a7... g.exe	Win RK 2000 full\CO... g.exe	✓
19ba9f2a24472a0aef2a7... t.exe	Win RK 2000 full\CO... t.exe	✓

### Multiscanning 0 /24 ENGINES

Engine Name	Verdict	Last engine update
AhnLab	No Threats Detected	04/13/2024 19:00 PM GMT
Avira	No Threats Detected	04/14/2024 01:49 AM GMT
Bitdefender	No Threats Detected	04/14/2024 00:10 AM GMT
Bkav Pro	No Threats Detected	04/13/2024 00:14 AM GMT
CMC	No Threats Detected	04/13/2024 12:44 PM GMT
ClamAV	No Threats Detected	04/13/2024 03:24 AM GMT
Emsisoft	No Threats Detected	04/14/2024 04:11 AM GMT
Filseclab	No Threats Detected	04/13/2024 18:32 PM GMT
IKARUS	No Threats Detected	04/14/2024 02:46 AM GMT
K7	No Threats Detected	04/13/2024 20:25 PM GMT

### File Overview

Category	E	Entropy	File Type	Scanned	GMT
File Extension	exe	-	File Type	Executable File	04/14/2024 06:20 AM
TrID	-	-	Duration	-	a few seconds
LibMagic	-	-	MD5	E4F1B4E5... D35F6	E4F1B4E5... D35F6
Magika	-	-	SHA-1	88CFD002... 5DB73	88CFD002... 5DB73
File Size	66.6 kB	-	SHA-256	BBCA0CC5... F32AF	BBCA0CC5... F32AF
Uploaded	04/14/2024 06:19 AM GMT	-	Company Name	-	-
SSDEEP	-	-	File Description	-	-
Architecture	-	-	File Version	-	-
Is DotNet	-	-	Internal Name	-	-
Is Packed	-	-	Legal Copyright	-	-
Is Digitally Signed	-	-	Original File Name	WI...MGMT\whoami.exe	-
			Product Name	-	-
			Product Version	-	-

## Explore salting

Salting is a means to improve the strength of hashing as a defense against brute force password cracking attacks. The salting process is performed automatically by some OSes, such as Linux. In this exercise, you will view the existence of salts for existing user accounts. Next, you will create password hashes both with and without salting.

Finally, you will crack the non-salted password quickly, and then you will see how salted password cracking takes significantly more time.

Connect to the KALI virtual machine and, if needed, sign in as root using Pa\$\$w0rd as the password.

A Terminal window should already be open.

Enter **cd ~**.

This command returns you to the root user's home directory (i.e., /root).

Enter **grep '\\$' /etc/shadow**.

This command displays the entry lines for only those accounts which have a stored password hash.

**Modern Linux systems salt password hashes by default.** The most common default hashing scheme of Linux is **yescrypt**. This is confirmed in the output by the "y" between the first two dollar signs. The "j9T" value between the second and third dollar signs are parameters used during the hashing process. The value between the third and fourth dollar signs is the salt used when producing the hash. Then, the value after the fourth dollar sign (until the colon) is the salted password hash.

- I. Since hashes produced by the yescrypt algorithm are extremely difficult to crack, generate your own password hashes to crack.

Enter:

```
openssl passwd -salt "" pass1 > hash.txt
```

This command will generate an MD5 hash of the password `pass1` *without using salting* and save it in the `hash.txt` file.

Enter **cat hash.txt** to view the contents of the file.

*Notice that there is no value between the second and third dollar signs. This indicates the password hash is not salted.*

**The openssl tool generates MD5 hashes by default.** This is encoded in the hash output by the 1 value between the first two dollar signs. There are other password hash options available by using parameters. For example **-5 uses SHA-256 and -6 uses SHA-512**. Each different hashing algorithm may alter the hash presentation. For example, MD5 hashes don't have parameters; thus, its output only contains the algorithm, salt, and hash.

```
(root㉿kali)-[~/usr/share/windows-resources/binaries]
└─# cd ~

[root㉿kali]-
└─# grep '\$' /etc/shadow
root:$y$j9T$WUrbNn0.InGryQxx05v.Y1$m4dbcSi.X6b3qGV6DTiW.qUj4uA710uyTM33i0pj2MD:19312:0:99999:7:::
kali:$y$j9T$c5JlQMsq8voGkG5hD1NSx/$1gIBFqENxHhjuXzqZifIiifg5z0mh4ClzUiSbwB70sC:19312:0:99999:7:::

[root㉿kali]-
└─# openssl passwd -salt "" pass1 > hash.txt
[root㉿kali]-
└─# ls
Desktop Documents Downloads hash.txt history LOD Music Pictures Public Templates Videos

[root㉿kali]-
└─# cat hash.txt
$1$6GOVYTqLJ9ax5wt0qsQ780

[root㉿kali]-
└─# john -incremental hash.txt
Created directory: /root/.john
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 128/128 SSE2 4x3])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
pass1      (?)
1g 0:00:00:09 DONE (2025-11-22 07:03) 0.1057g/s 30241p/s 30241c/s 30241C/s papal..pasie
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

[root㉿kali]-
└─#
```

Use **John the Ripper** to perform a brute force password crack against the unsalted password hash by entering:

## **john -incremental hash.txt**

John should crack the unsalted password hash in less than 10 seconds.

II. Generate a salted password hash by entering:

**openssl passwd -salt SALT pass1 > salted-hash.txt**

**A proper salt value is a random string.** Some hashing algorithms support salts of up to 512 bytes. Here, you are using a pre-selected salt of SALT to make it very obvious when viewing the hashing output.

Enter **cat salted-hash.txt** to view the contents of the file.

*Notice that the salt value of SALT is present between the 2nd & 3rd \$*

Use John the Ripper to perform a brute force password crack

## **john -incremental salted-hash.txt**

John should crack the salted password hash in less than 10 seconds. The full Linux hash statement is present in the salted-hash.txt file. Therefore, John is provided the salt value, which allows it to quickly crack the password.

```
(root㉿kali)-[~]
└─# openssl passwd -salt SALT  pass1 > salted-hash.txt

(root㉿kali)-[~]
└─# cat salted-hash.txt
ca: command not found

(root㉿kali)-[~]
└─# cat salted-hash.txt
$1$SALT$78YGmHvIahaaMyya90aAn.

(root㉿kali)-[~]
└─# john -incremental salted-hash.txt
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 128/128 SSE2 4x3])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
pass1      (?)
1g 0:00:00:09 DONE (2025-11-22 07:05) 0.1018g/s 29132p/s 29132c/s 29132C/s papal..pasie
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Remove the salt value from the file by entering:

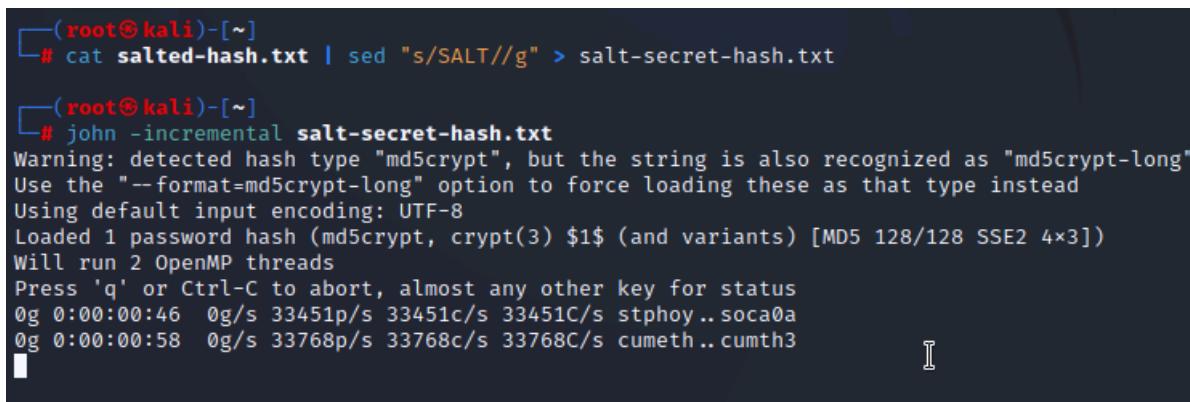
```
cat salted-hash.txt | sed "s/SALT//g" > salt-secret-hash.txt
```

Use John the Ripper to perform a brute force password crack against the salted password hash when the salt value is not known by entering:

```
john -incremental salt-secret-hash.txt
```

*John will take up to 814,506,250 seconds (~25.8 years) to crack the salted password when it does **not** have knowledge of the salt value.*

This is because the hash is effectively produced from 9 characters instead of just the 5 of the original password.



```
(root㉿kali)-[~]
# cat salted-hash.txt | sed "s/SALT//g" > salt-secret-hash.txt

(root㉿kali)-[~]
# john -incremental salt-secret-hash.txt
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 128/128 SSE2 4x3])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:46 0g/s 33451p/s 33451c/s 33451C/s stphoy..soca0a
0g 0:00:00:58 0g/s 33768p/s 33768c/s 33768C/s cumeth..cumth3
```

The default character set used by John the Ripper is the standard US-ASCII table of 95 characters (i.e., uppercase, lowercase, numbers, and symbols present on a standard US keyboard).

Thus, with four salt characters (which could be any of the 95 character options) and the original five-character password (i.e., pass1), which took 10 seconds (or less) to crack, the salted hash of that password (with a 4 character salt) would take  $10 \text{ seconds} \times 95 \times 95 \times 95 = 814,506,250$  seconds.

Wait about 10 seconds, then press the spacebar on your keyboard to get a status report from John. Wait 10 more seconds, then press the spacebar again. You can repeat this several times.

Notice the far-right status element. This is the range of passwords being attempted at the instant you pressed the spacebar to present the status message.

The status output includes:

- Successful guesses (## g)
- Time elapsed, successful guesses/second (## g/s),
- Candidate passwords tested/second (## p/s),
- "Crypts" (password hash or cipher computations)/second (## c/s)
- Combinations of candidate password & target hash/second (## C/s)
- Current candidate passwords.

Type CTRL+C to terminate John.