



# On the Security of User Passwords on Linux Operating Systems

Chair for System Security  
Horst Goertz Institute for IT-Security  
Ruhr-University Bochum

René Korthaus  
email: [sec@cordney.com](mailto:sec@cordney.com)  
web: <http://cordney.com>

April 25, 2008

## Abstract

In this paper we determine the security of user passwords on Linux based operating systems. We have a look at the two basic security mechanisms passwords are created and stored using a reference Linux distribution, locate common attack vectors and propose available countermeasures.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>User Passwords in Linux</b>	<b>4</b>
2.1	Basic password storage . . . . .	4
2.2	Advanced password storage using the Shadow Suite . . . . .	5
<b>3</b>	<b>Attack vectors</b>	<b>6</b>
3.1	Theoretical attacks . . . . .	6
3.1.1	Basic password storage . . . . .	6
3.1.2	Advanced Password Storage . . . . .	6
3.2	Practical attacks . . . . .	6
3.2.1	Using Crack . . . . .	6
3.2.2	Using John the Ripper . . . . .	7
3.2.3	Exhaustive Key Search . . . . .	8
<b>4</b>	<b>Countermeasures</b>	<b>9</b>
<b>5</b>	<b>Conclusion</b>	<b>10</b>

# 1 Introduction

Linux was originally written by Linus Torvalds in 1991. Until now many different versions, called distributions, emerged from the initial Linux version written by Torvalds. Despite the many different distributions available today, all use the same foundation. To create a standard base for all different flavors of Linux, the Linux Foundation established the "Linux Standard Base"<sup>1</sup> - guidelines for compatibility and interoperability.

These also feature a core specification, the "Linux Standard Base Core Specification", which also mentions basic commands and utilities to be implemented by Linux operating systems which conform to the "Linux Standard Base", including the *passwd* command we'll later use for creating user passwords.

On modern operating systems multiple-user support is an essential requirement. In the common scenario a username and a password is used to identify a user. If the user and password match, access to the system, file or a special area is granted. Choosing a *secure* password is important to prevent guessing or dictionary attacks. However, in this report we will not focus on choosing secure passwords.

Nowadays, user passwords are becoming even more essential as disk encryption is deployed more and more in businesses and homes, where the disk encryption key is derived from the user password in most cases.

Thus, although a password is the simplest method to protect data from unauthorized access, it is the most deployed security feature in the world in use in the mobile phone's sim-card, the banking- and credit card, pc's in homes and offices, online services and almost all other cases where men deal with information technology. Therefore it is crucial to securely store them in the it-system where it is compared to the user's input.

---

<sup>1</sup><http://www.linux-foundation.org/en/LSB>

## 2 User Passwords in Linux

*Our reference Linux distribution is a plain Debian Sarge (3.1) installation.*

Basically, there are two methods how passwords are stored in Linux:

### 2.1 Basic password storage

The basic password storage only uses the `/etc/passwd` file. User information is stored in the following format:

```
username:password:UID:GID:full_name:home_directory:shell
```

**username**

the user/login name

**password**

the encoded user/login password

**UID**

the numerical user ID

**GID**

the numerical group ID

**full name**

the user's full name

**home directory**

the user's home directory

**shell**

the user's default login shell

The password is taken from user input when using the `passwd` command. A cryptographic salt is padded with the password and encoded as:

```
Enc_P = Crypt::DES(Salt|P*)
```

For encoding, the `crypt` function is used. This is the standard Linux password encryption function. It is based on the well known Data Encryption Standard. `Crypt` obtains the 56-bit DES encryption key by taking the lower 7 bits from the first eight characters of the user's password to `P*` and encrypts a NULL string<sup>2</sup>. The result is the encrypted password, a 13 ASCII character wide string with the first two characters being the salt itself.

The salt is a randomly chosen 2-byte value. Its purpose is to complicate dictionary and brute-force attacks using rainbow tables. The result is the encoded password which contains the salt and the password hash.

The basic password storage is a security risk, as the `/etc/passwd` file and thus the encoded passwords are readable by every user. So anyone can mount an offline brute-force attack on the user passwords of the affected system. Therefore, in 1987, Julie Haugh wrote the "Shadow Password Suite" and it became standard on Linux operating systems from 1992 on.

---

<sup>2</sup>A NULL string is a string consisting of all zeros

## 2.2 Advanced password storage using the Shadow Suite

The advanced password storage, which is the default on most Linux distributions, uses the "Linux Shadow Password Suite". It contains replacement programs for *su*, *login*, *passwd*, *newgrp*, *chfn*, *chsh*, and *id*. The Shadow Suite uses both */etc/passwd* and */etc/shadow* files to store user information. The format of the */etc/shadow* file is the following:

```
username:passwd:last:may:must:warn:expire:disable:reserved
```

The only important bits for us here are username and password. The username is the same as in the */etc/passwd* file. The */etc/shadow* file is only readable by root and thus is secure against evedropping. When using the Shadow Suite, the password field of the entry in the */etc/passwd* file is replaced with a "x". This means that the encoded password is stored inside the */etc/shadow* file.

The password is also taken from user input when using the *passwd* command. A 12-byte cryptographic salt is padded with the password and encrypted as:

```
Enc_P = Crypt::MD5(Salt|Password)
```

As an example, adding the new user "foo" with the simple password "password" would result in the following entries in the */etc/passwd* and the */etc/shadow* file:

```
foo:x:1001:1001:,,,:/home/foo:/bin/bash
foo:$1$kHkGsBvU$1wIxzZBgIO1E5Hzb3Wn3q0:13947:0:99999:7:::
```

The "\$1" indicates that the MD5 hash algorithm was used. The eight characters following the second "\$", in this case "kHkGsBvU" are the salt. The rest is the actual hash of the password. Adding another user "bar" with the same password results in:

```
bar:x:1002:1002:,,,:/home/bar:/bin/bash
bar:$1$myqznT81$qLQvYCOafNJ1FqUGhHwwY/:13947:0:99999:7:::
```

As we can see, the encrypted passwords clearly differs although the original passwords were the same. This is another advantage of the cryptographic salt used by the *crypt* function. Flipping just one bit of the input to a cryptographic primitive with pseudo-random functionality will result in many bits flipped in the output. This is called the "Avalanche-Effect". A side-effect is that two users, choosing the same password, cannot be aware of it, as the random salt results in two different password hashes.

## 3 Attack vectors

### 3.1 Theoretical attacks

#### 3.1.1 Basic password storage

As the *crypt* function only takes the first eight characters from the user's input, the complexity is reduced significantly. If you choose a longer password, the higher bits will be trashed. This is due to the fixed key length of DES, accepting only 56-bit long keys. Thus the complexity of a brute-force attack is equal to the complexity of a DES brute-force attack, giving  $2^{78} = 2^{56}$  possible keys. A brute-force attack is feasible using parallel computing or using specialized machines such as the COPACOBANA<sup>3</sup>.

Furthermore, the DES nowadays is well evaluated and several attacks have been developed, mainly analytical attacks like *differential cryptanalysis* (DC) and *linear cryptanalysis* (LC) which reduce the complexity to  $2^{55}$  respectively  $2^{43}$ .

#### 3.1.2 Advanced Password Storage

Since advanced password storage uses MD5 by default, complexity equals to the strength of this hash function. The MD5 hash function gets an input of arbitrary length and produces a fixed length 128-bit output, called the hash. As the space of arbitrary length is significantly larger than the space of the output, different input values can result in the same hash. This is called a collision. Research in finding collisions is still being made and the first collision was found in 2004 by chinese scientists [5].

As it comes to using rainbow tables, the cryptographic salt comes in place again. As the salt pseudo-randomizes the hashed password, precomputed rainbow tables are of no use at all. Instead, the rainbow tables have to be computed live for each user, which we'll do in the following chapter.

### 3.2 Practical attacks

When it comes to practical attacks, basically two password cracking tools come in place:

#### 3.2.1 Using Crack

*crack* claims not to be designed for breaking user passwords but for finding weak passwords. It also supports distributed password cracking over networks. Getting and installing *Crack* is straightforward. Once this is done, we need to merge the */etc/passwd* and */etc/shadow* file into one file.

```
cp /etc/shadow /root/pwd-merged
shadmrg.sv > /root/pwd-merged
```

Now we can start cracking.

```
Crack /root/pwd-merged
```

---

<sup>3</sup><http://www.copacobana.org/>

The cracker runs as a daemon so we have to check for cracked passwords from time to time.

```
./Reporter -quiet
```

tells us about the current status. This can take a while, though.

All password crackers mainly work the same way. They first try simple passwords, such as "password" or the user name. Then they run a dictionary with common passwords against the password-file. Third they try foreign language or special dictionaries. If they were not successful they try all combinations of letters up to a limit of characters (the password length). Lastly, they try a combination of all lowercase and uppercase letters, numbers and punctuation marks up to a limit.

For an eight character password using *letters only*, the key-space is  $26^8$ , which is roughly between  $2^{36}$  and  $2^{37}$ . It dramatically increases when you put numbers and punctuation marks into place.

### 3.2.2 Using John the Ripper

"John the Ripper" is the classic password cracker. It supports cracking MD5, DES and Blowfish encrypted passwords. Like with *Crack*, we first need to merge the *passwd* and *shadow* file. John the Ripper comes with the *unshadow* command, which does this for us.

```
unshadow /etc/passwd /etc/shadow > /root/pwd-unshadowed
```

A simple

```
john /root/pwd-unshadowed
```

starts John. This will take it's time. John the Ripper knows 3 modes: In the first mode he tries to crack the password using the login/GECOS information (that could be full name, birth, adress, etc.). In the second mode he simply runs a wordlist against the password file. In the last, most powerful mode, he tries all different combinations of characters to resolve the password. All modes are configurable.

As with *Crack*, John the Ripper runs as a daemon in the background. To show his progress we can run:

```
john -show /root/pwd-unshadowed
```

If John is successful, the output will look like

```
foo:password:1001:1001:,,,:/home/foo:/bin/bash
bar:password:1002:1002:,,,:/home/bar:/bin/bash
```

where "password" was our user password.

### 3.2.3 Exhaustive Key Search

Whenever no vulnerability is found in a cryptosystem, an exhaustive key search or brute-force attack is the last remaining opportunity. Recent further research and development revealed some interesting possibilities for faster exhaustive key search, either using special purpose machines like COPACOBANA or using the massive computing power of the GPU in traditional PC's. Using these advanced machines and mechanisms brings a major performance boost to the traditional brute-force attack.

The table below gives some data on the performance of these new techniques.

Algorithm	Pentium4@3GHz	Copacobana	CUDA[10]	CUDA-SLi(4x) <sup>a</sup>
DES	545 years	6.4 days	no data	no data
MD5	$1.8 \cdot 10^{24}$ years	no data	$2.1 \cdot 10^{22}$ y	$5.4 \cdot 10^{21}$ years

Table 1: Average time for an exhaustive key search ( $2^{55} / 2^{127}$  keys)

<sup>a</sup>estimated theoretical performance of [10] using 4 GPUs in parallel via SLi

While we see a significant performance increase in DES exhaustive key search, MD5 key search is still above  $10^{21}$  space. This is due to the very large key space of  $2^{128}$  of MD5. The major difference in MD5 performance becomes much clearer when comparing the keys per second performance on MD5.

Algorithm	Pentium4@3GHz	Copacobana	CUDA[10]	CUDA-SLi(4x)
MD5	3 million	no data	250 million	1 billion

Table 2: Keys per second performance of MD5 exhaustive key search

You see a significant performance increase around 5-6 times faster than a consumer CPU. Using the power of GPUs will become much more important to cryptography in the future as the costs are relatively low in comparison to special purpose machines.

## 4 Countermeasures

A good password is essential to security. There are numerous guidelines for choosing secure passwords on the internet, so we will not go into detail here. Basically, it's about a minimum length of 8-10 characters consisting of small and capital letters, numbers and special characters. Furthermore, there are some *automated password generators* integrated into operating systems, such as the FIPS-181[11].

In the Linux case, if you are not already using it, you should switch to the advanced password storage using the "Password Shadow Suite" and use MD5 or Blowfish for encryption of the user password. As MD5 has been tampered already, in a long term SHA-256 should be used.

Intelligent brute-force attacks will always be possible, as people will always use easy-to-remember but cryptographically weak passwords. Cracking tools such as *Crack* and *John the Ripper* should be seen as hardening tools and used by administrators to detect weak passwords periodically and should inform the affected users about their insecure passwords.

## 5 Conclusion

The user password security system is based on open standards, such as DES and MD5, that have been well researched. User passwords on Linux operating systems are encrypted or hashed, together with a random salt, and stored in a global password file. The complexity of possible attacks derives from the complexity to break either the cipher or the hash algorithm. Users should keep themselves up-to-date about the overall security of cryptosystems used by their Linux distribution to protect passwords.

But as said before, the weakest link is the password itself. A cryptosystem is only as strong as it's weakest link, and attackers know that. So a good password, neither a password from the dictionary nor a password based on personal information of it's owner, is the best protection against unauthorized access and data theft.

Over the last years, Linux has gained more and more market share and is the overall leader in server statistics in conjunction with the Apache webserver [12]. Thus, security of Linux operating systems have become more and more interesting, as gaining control over a webserver could generate a and destroy massive revenue when using it for attacks like phishing and data fraud. Wordlist attacks on Linux servers on the internet are not unusual nowadays and the simplest method to break into a system. The weakest link of a system truly is the password and will stay it in a mid term. In a long term, we may see biometric systems substitute for passwords.

As a final conclusion, the security of user passwords on Linux operating systems rises and falls with the strength of these passwords.

## References

- [1] *passwd* manpages
- [2] *crypt* manpages
- [3] Linux Shadow Password HOWTO  
<http://tldp.org/HOWTO/Shadow-Password-HOWTO.html>
- [4] Linux Shadow Password HOWTO  
<http://tldp.org/HOWTO/Shadow-Password-HOWTO.html>
- [5] Cracking Unix Passwords  
<http://www.governmentsecurity.org/articles/CrackingUnixpasswordfilesforbeginners.php>
- [6] Installing and configuring Crack  
<http://articles.techrepublic.com.com/5100-6347-1053600.html>
- [7] Introduction to Cryptography and Data Security, Prof. Christof Paar, Chair for Communication Security, Ruhr-University Bochum, Germany  
<http://www.crypto.rub.de>
- [8] MD5 collision analysis  
<http://eprint.iacr.org/2004/199.pdf>
- [9] John the Ripper FAQ  
<http://www.cyberciti.biz/faq/unix-linux-password-cracking-john-the-ripper/>
- [10] Elcomsoft Distributed Password Recovery using Nvidia's CUDA framework  
<http://www.elcomsoft.com/edpr.html>
- [11] NIST: FIPS-181  
<http://www.itl.nist.gov/fipspubs/fip181.htm>
- [12] Netcraft web server survey, Feb 2008  
[http://news.netcraft.com/archives/2008/02/06/february\\_2008\\_web\\_server\\_survey.html](http://news.netcraft.com/archives/2008/02/06/february_2008_web_server_survey.html)