

Šifrování oddílů v GNU/Linuxu pomocí LUKS

Vložil/a [shadow](#) [1], 31. prosinec, 2007 - 19:32

- [Encryption](#) [2]
- [GNU/Linux a BSD](#) [3]
- [Security](#) [4]

Ještě před nedávnem existovalo vedle sebe více implementací šifrování diskových oddílů v GNU/Linuxu (loop-aes, dm-crypt), přičemž jednotlivé cesty bojovaly různě s problémy jako bezpečnost při šifrování tak ohromných objemů dat či vzájemná kompatibilita. Oba tyto problémy by měl vyřešit LUKS (Linux Unified Key Setup) o kterém je tento článek.

Oba problémy (stejně jako některé další) by měl vyřešit [LUKS](#) [5] (Linux Unified Key Setup), který představuje standard pro šifrování oddílů v GNU/Linuxu a navíc implementuje [špičkové metody](#) [6] pro bezpečné šifrování velkých objemů dat. Dostí však teorie (zejména vzhledem k tomu, že nejsem expert na šifrovací algoritmy) a pustíme se do návodu.

Budeme potřebovat balíček [cryptsetup-luks](#) [7], který v některých distribucích (Debian, Arch Linux) již dávno nahradil původní balíček cryptsetup. Ještě před tím, než předvedu, jak vytvořit zašifrovaný oddíl, pár slov k tomu, jak LUKS pracuje.

LUKS oddíl začíná hlavičkou, která obsahuje základní informace včetně použité šifry. Následují sloty pro celkem osm uživatelských klíčů a nakonec zašifrovaná data. Data jsou šifrována hlavním klíčem (master key), jehož kontrolní součet se nachází v hlavičce oddílu. Každý z použitých slotů obsahuje příslušným uživatelským heslem zašifrovaný master key. K dešifrování oddílu tedy stačí kterýkoliv klíč. Podrobnější informace o implementaci naleznete v příslušném [sheetu](#) [8] nebo v mé [seminární práci](#) [9] na téma diskové šifrování. Teď už nás čeká práce s LUKSem.

Oficiální dokumentace doporučuje provést před vytvořením šifrovaného oddílu jednak kontrolu špatných sektorů a přepsání celého oddílu náhodnými daty. K tomu je doporučován příkaz:

```
badblocks -c 10240 -s -w -t random -v /dev/hda4
```

Kontrolu špatných sektorů lze samozřejmě vynechat a oddíl zaplnit náhodnými daty pomocí dd. Zaplnění náhodnými daty generovanými pomocí /dev/urandom důrazně doporučuji, protože se jedná o jeden z nejkvalitnějších generátorů pseudonáhodných čísel. Příkaz pro zaplnění blokového zařízení hda4 náhodnými daty by vypadal takto :

```
dd if=/dev/urandom of=/dev/hda4
```

Očekávejte, že proces potrvá velmi dlouho, tzn. i několik hodin či v extrémních případech déle než den. Výsledkem však bude podstatně vyšší bezpečnost celého řešení. Já mohu snad jen doporučit použití nějakého pipemeteru, abyste měli přehled o tom, jak operace probíhá:

```
dd if=/dev/urandom | pipemeter | dd of=/dev/hda4
```

V mém případě trvalo zaplnění 200GB dat asi 20 hodin (Athlon 64 3200+, 2,0GHz). Když je příprava dokončena, můžeme vytvořit LUKS oddíl:

```
cryptsetup luksFormat /dev/hda4
```

Občas ale bývá lepší specifikovat jednotlivé parametry pro šifru, hash a velikost klíče ručně. S jádrem 2.6.20 a novějším máme podporu šifrovacího módu LRW, který je bezpečnější než CBC-ESSIV.

Informace o šifrovacích módech a jejich bezpečnosti na diskové šifrování probírám ve své [seminární práci](#) [9]. LRW tedy použijeme takto:

```
cryptsetup -c aes-lrw-benbi -h sha256 -s 384 -y luksFormat /dev/sdc2
```

Na starších jádrech použijeme alespoň CBC-ESSIV:

```
cryptsetup -c aes-cbc-essiv:sha256 -h sha256 -s 256 -y luksFormat /dev/sdc2
```

Jak vidíme, parametr `-c` nám umožňuje specifikovat šifru (aes, serpent, twofish, cast6, apod.), šifrovací mód (ecb, cbc, lrw) a způsob generování inicializačního vektoru (plain, essiv:hash, benbi), v tomto pořadí. Z hlediska bezpečnosti je nejhorším řešením pro diskové šifrování mód ecb a plain inicializační vektor (to už je snad lepší nešifrovat, abyste nepropadli falešnému pocitu bezpečí) a nejlepším lrw a benbi (pozor, benbi je určen pro "narrow block" módy, jako LRW, XEX, apod., není určen pro CBC). Obecně lze doporučit použití LRW s tím, že pokud jej nemůžete z nějakého důvodu použít, použijte alespoň CBC-ESSIV.

Následně vytvoříme příslušné zařízení device mapperu:

```
cryptsetup luksOpen /dev/hda4 zasifrovano
```

Druhý parametr určuje název vytvořeného zařízení v `/dev/mapper`, lze použít cokoliv, co uznáte za vhodné. Nyní na zařízení vytvoříme souborový systém:

```
mkfs.ext3 /dev/mapper/zasifrovano
```

Nyní již existující souborový systém připojíme:

```
mount /dev/mapper/zasifrovano /mnt/zasifrovano
```

A práce je skoro hotová. Pokud budeme chtít takto zašifrovat třeba `/home` nebo něco klíčového (třeba kořenový oddíl), buď nám v tom distribuce vyjde vstříc již předem upravenými startovacími skripty, nebo budeme muset příslušné změny provést sami.

Odpojení oddílu včetně odstranění zařízení device mapperu lze provést takto:

```
umount /dev/mapper/zasifrovano  
cryptsetup luksClose zasifrovano
```

» Správa klíčů

Jak už jsem naznačil, je možné použít až osm klíčů, přičemž každý je schopen dešifrovat oddíl. Jak ale přidat či ubrat klíč? Snadno. Zkusíme přidat nový klíč:

```
cryptsetup luksAddKey /dev/hda4
```

Pokud zpozorníte u výpisů `cryptsetup`, všimnete si zprávy "key slot 0 unlocked.". Tato zpráva vám říká, kterým klíčem jste si odemkli. V tomto případě prvním (resp. nultým) klíčem. Pokud jste přidali druhý klíč, následujícím příkazem ho odstraníte:

```
cryptsetup luksDelKey /dev/hda4 1
```

» Záloha hlavičky

Pokud nemáte RAID-1, hlavička je "single point of failure". Tu však lze snadno zálohovat pomocí `dd`. Nejprve si ale necháme vypsat informace z LUKS hlavičky oddílu:

```
cryptsetup luksDump /dev/hda4
```

Ve výpisu si všimněte čísla označeného jako "Payload offset:". Toto číslo pak zadejte jako parametr count= pro dd:

```
dd if=/dev/hda4 of=luks_header.hda4 count=cislo
```

» Použití USB flash disku jako tokenu

Není nic jednoduššího. Připojte flashku a vygenerujte příslušný klíč:

```
dd if=/dev/random of=/mnt/flashdisk/key.hda4 bs=1 count=256
```

Následně přidejte klíč do "klíčenky":

```
cryptsetup luksAddKey /dev/hda4 /mnt/flashdisk/key.hda4
```

A vytvořte zařízení device mapperu:

```
cryptsetup -d /mnt/flashdisk/key.hda4 luksOpen /dev/hda4 zasifrovano
```

» Zajištění swapu pro vyšší bezpečnost

Pokud budete používat zašifrované oddíly, bývá dobrým nápadem zajistit swapovací oddíl tak, aby z něj v případě, že si do něj linuxové jádro dočasně uloží obsah paměti i s vaším původním heslem, nebylo možné později tuto informaci vytáhnout. Úpravu konfigurace pro zajištění swapu při každém bootu nechám na vás, neb u každé distribuce bude mírně odlišná.

Swap zajistíme tak, že necháme jeho obsah šifrovat náhodným heslem při každém startu systému:

```
cryptsetup -d /dev/urandom create cswap /dev/hda2  
mkswap /dev/mapper/cswap  
swapon /dev/mapper/cswap
```

» Přenositelnost

Přenositelnost kontejnerů šifrovaných pomocí LUKS zajišťuje na platformě Windows projekt [FreeOTFE](#) [10]. Pro jiné platformy (např. *BSD systémy) zatím o žádném řešení zpřístupnění LUKS kontejnerů nevím.

Zdroje:

- oficiální [wiki](#) [11] projektu LUKS
- [Using DM-Crypt](#) [12] na [ArchWiki](#) [13]

URL článku:

<https://security-portal.cz/clanky/%C5%A1ifrov%C3%A1n%C3%AD-odd%C3%AD%C5%AF-v-gnulinuxu-pomoc%C3%AD-luks>

Odkazy:

- [1] <https://security-portal.cz/users/shadow>
- [2] <https://security-portal.cz/category/tagy/encryption>
- [3] <https://security-portal.cz/category/tagy/gnu/linux-bsd>
- [4] <https://security-portal.cz/category/tagy/security>
- [5] <http://luks.endorphin.org/>
- [6] <http://clemens.endorphin.org/publications>

- [7] <http://luks.endorphin.org/dm-crypt>
- [8] <http://luks.endorphin.org/spec>
- [9] <http://shadow.cz/wiki/seminarky/otfe>
- [10] <http://www.freeotfe.org/>
- [11] <http://www.saout.de/tikiwiki/tiki-index.php?page=LUKS>
- [12] http://wiki.archlinux.org/index.php/Using_DM-Crypt
- [13] <http://wiki.archlinux.org>