ZFS-Encryption

Sichere Datenhaltung in der Cloud

Tübix 2018 | Daniel Kobras

Native ZFS-Verschlüsselung

- Neue Funktion in OpenZFS
- Nicht kompatibel zu Verschlüsselung in Oracle ZFS
- Ursprünglich entwickelt für ZFS on Linux, Portierungen auf übrige OpenZFS-Architekturen:

	ENTWICKLUNGSSTATUS
macOS	in aktuellem Release verfügbar
Linux	in Entwicklungszweig/nächstem Major-Release (0.8) verfügbar
Illumos	Portierung in Arbeit
FreeBSD	Portierung in Arbeit (FreeNAS)
Windows	?

Status Quo

- Bislang:
  - Verschlüsselung unterhalb ZFS-Pool mit OS-Bordmitteln (Leaf-vdevs verschlüsselt mit dm-crypt, geli, etc.)
  - Vorteil: vollständige Verschlüsselung aller Nutz- und Metadaten
  - Nachteile: keine Verschlüsselung von Teilbereichen des Pools, keine unterschiedlichen Schlüssel für Teilbereiche, Konsistenzprüfung nur auf unverschlüsseltem Pool möglich, Replikation nur mit unverschlüsselten Daten möglich
- oder
  - zvol verschlüsselt mit OS-Bordmitteln, darauf beliebiges Dateisystem
  - Vorteil: Scrub und Replikation verschlüsselt möglich, unterschiedliche Schlüssel für einzelne zvols möglich
  - Nachteil: ZFS-Features werden nur bedingt genutzt, unnötiger Overhead durch zusätzliches Dateisystem, eingeschränkte Flexibilität

Architektur

- Native Verschlüsselung:
  - In Pool-Funktionalität integriert
  - Steuerbar auf Dataset-Ebene
  - Vererbung an Kind-Datasets möglich
    - Nutz- und Metadaten des Datasets (Dateisystems) verschlüsselt
    - Dataset-Properties unverschlüsselt
    - Pool-Metadaten unverschlüsselt
- Scrub und Snapshots auf verschlüsselten Pool möglich
- Deduplizierung gleicher Daten möglich (aber: prinzipbedingtes Information-Leak)
- Neue Funktion: raw send/receive verschlüsselter Datasets

Anwendungszweck

- **Secure Erase:** verwerfen des Schlüssels macht sämtliche Daten sicher und unmittelbar unbrauchbar (Schnelligkeit, defekte Medien, remapped Blocks)

- **System Authentication:** verhindert unerwünschte Zugriffe von beliebigen Maschinen (Shared Storage, transportable Medien, Replikation)

- **Aber:** keine Nutzerauthentisierung (ist Schlüssel bekannt gemacht, kann OS/root zugreifen); Nutzerautorisierung bleibt Aufgabe der Rechteverwaltung des OS (Encryption at rest ist keine Encryption in use)

- **Sorgfältige Schlüsselverwaltung entscheidend, andernfalls tritt ungewollt Secure Erase-Funktion in Kraft!**

Was hat das mit Cloud zu tun?

- Nonplusultra für Cloud-Backup!
- Effiziente Replikation über inkrementelle Snapshots
- Kompatibel mit verschlüsselten Datasets
- Backup-Site muss Schlüssel nicht kennen
- Backup-Site kann trotzdem Datenintegrität sicherstellen (Scrub)
- Im Desaster-Fall entweder Schlüssel an Backup-Site übermitteln (schnelle Verfügbarkeit) oder Daten verschlüsselt zurück replizieren (hohe Vertraulichkeit)
- Haken: Backup-Site muss ZFS-Pool (mit raw send/receive) bereitstellen, S3-Storage reicht nicht
- D.h. eigene Infrastruktur nötig (rsync.net auf FreeBSD-Basis, noch keine Verschlüsselung verfügbar)
- Haken 2: Backup des Schlüssels?

Technische Umsetzung

- Symmetrische Verschlüsselung
- Mehrstufige Verschlüsselung, Wechsel des nach außen sichtbaren Schlüssels erzwingt kein Neuschreiben aller Daten, restliche Schlüssel zufällig erzeugt, werden mit übergeordnetem Schlüssel verschlüsselt und mit den Daten gespeichert
- Bislang nur ein Schlüssel pro Dataset implementiert (Design unterstützt prinzipiell mehr Schlüssel)
- Konzept lässt sich aber bei Bedarf selbst um weitere Stufe erweitern
  - Z.B. neuen Schlüssel erzeugen, mit mehreren Public Keys verschlüsseln und in Dataset-Property speichern
  - Private Keys liegen in Smartcard, lokalem Storage, Key Management System, Bankschließfach, Safe, etc.
  - Fehlerfreie Umsetzung der Schlüsselverwaltung wichtig, sonst Tritt wieder Secure-Erase-Funktion in Kraft!

Demo: Encrypted Datasets (1/2)

```
# create pool
user@host:~> sudo zpool create -f tuebix /dev/disk2
Password: ****************

# encryption feature is available (but currently unused)
user@host:~> zpool get feature@encryption tuebix
NAME    PROPERTY            VALUE               SOURCE
tuebix  feature@encryption  enabled             local

# create encryption dataset (prompting for passphrase)
user@host:~> sudo zfs create -o encryption=on -o keyformat=passphrase tuebix/encrypted
Enter passphrase: tuebix2018
Re-enter passphrase: tuebix2018

# encryption feature now indicates that it is actively used
user@host:~> zpool get feature@encryption tuebix
NAME    PROPERTY            VALUE               SOURCE
tuebix  feature@encryption  active              local

# display various encryption properties
user@host:~> zfs list -ro name,encryption,keyformat,keylocation,keystatus tuebix
NAME               ENCRYPTION   KEYFORMAT  KEYLOCATION    KEYSTATUS
tuebix                    off        none  none                    -
tuebix/encrypted  aes-256-ccm  passphrase  prompt         available

# unload encryption key (dataset must be unmounted)
user@host:~> sudo zfs umount tuebix/encrypted
Running process: '/usr/sbin/diskutil' 'unmount' '/Volumes/tuebix/encrypted'
Unmount successful for /Volumes/tuebix/encrypted
user@host:~> sudo zfs unload-key tuebix/encrypted

# trying to mount the dataset now fails (as it should) because the key is missing
user@host:~> sudo zfs mount tuebix/encrypted
cannot mount '/Volumes/tuebix/encrypted': encryption key not loaded
```

Demo: Encrypted Datasets (2/2)

```
# we can still snapshot and replicate the dataset without knowing the key!
user@host:~> sudo zfs create -o mountpoint=none tuebix/receive
user@host:~> sudo zfs snap tuebix/encrypted@snap1
user@host:~> sudo zfs send -R tuebix/encrypted@snap1 | sudo zfs recv -e tuebix/receive
cannot send tuebix/encrypted@snap1: encrypted dataset tuebix/encrypted may not be sent with properties without the raw flag
cannot receive: failed to read from stream

# oops, a regular send doesn't work without the key—need new raw send (option -w) to send encrypted stream
user@host:~> sudo zfs send -Rw tuebix/encrypted@snap1 | sudo zfs recv -e tuebix/receive

# the replicated dataset is still encrypted (using the same key)
user@host:~> zfs list -o name,encryption,keyformat,keylocation,keystatus tuebix/receive/encrypted
NAME                       ENCRYPTION   KEYFORMAT  KEYLOCATION    KEYSTATUS
tuebix/receive/encrypted  aes-256-ccm  passphrase  prompt       unavailable

# note how we could still access the dataset properties even without the key
# checking the pool integrity doesn't require any keys, either, meaning we can ensure data integrity without knowing the data
user@host:~> sudo zpool scrub tuebix
user@host:~> zpool status
  pool: tuebix
 state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Sat Jun  9 15:03:39 2018
config:

        NAME        STATE     READ WRITE CKSUM
        tuebix      ONLINE       0     0     0
          disk2     ONLINE       0     0     0

errors: No known data errors

# normal send (with key loaded) creates an unencrypted replica
user@host:~> sudo zfs load-key tuebix/encrypted
Enter passphrase for ,tuebix/encrypted': tuebix2018
user@host:~> sudo zfs send -R tuebix/encrypted@snap1 | sudo zfs recv tuebix/unencrypted
cannot send tuebix/encrypted@snap1: encrypted dataset tuebix/encrypted may not be sent with properties without the raw flag
cannot receive: failed to read from stream

# oops, cannot send an encrypted dataset as an unencrypted stream including properties—just sending the data works, though
user@host:~> sudo zfs send tuebix/encrypted@snap1 | sudo zfs recv tuebix/unencrypted
Mount successful
```

Demo: Key Management (1/2)

```
# To share a key between multiple parties, we can store a copy in the
# dataset's properties that is encrypted to multiple gpg keypairs. zfs@host
# identifies the local machine, zfs@securebackup's secret key is kept offsite
# in a safe, disaster-proof location
user@host:~> gpg --list-keys
/Users/user/.gnupg/pubring.kbx
------------------------------
pub   rsa2048 2018-06-08 [SC] [verfällt: 2020-06-07]
      3AE09346935399BDB17BCD3FA4171D8300CC4857
uid         [ ultimativ ] zfs@host
sub   rsa2048 2018-06-08 [E]

pub   rsa2048 2018-06-08 [SC] [verfällt: 2020-06-07]
      463E1CCFC01F920A97C4036DA0DA1F7DFF5BDBA6
uid         [ ultimativ ] zfs@securebackup
sub   rsa2048 2018-06-08 [E]

user@host:~> gpg --list-secret-keys
/Users/user/.gnupg/pubring.kbx
------------------------------
sec   rsa2048 2018-06-08 [SC] [verfällt: 2020-06-07]
      3AE09346935399BDB17BCD3FA4171D8300CC4857
uid         [ ultimativ ] zfs@host
ssb   rsa2048 2018-06-08 [E]
```

Demo: Key Management (2/2)

```
# let’s create and stage a new passphrase (tuebix2019), and store it as a property within the dataset itself
# we encrypt one copy for the system’s GPG key pair (stored locally), and another copy for the offsite key
user@host:~> echo tuebix2019 | gpg –e –r zfs@myhost –r zfs@securebackup | sudo zfs set myhost:key.staging="$(base64 /dev/stdin)"
tuebix/encrypted
user@host:~> zfs list –Ho myhost:key.staging tuebix/encrypted
hQEMA1TsyXUXyLcnAQgAqmnBIBWIrdoJEfXzNpSjEHX+2cri+TJdkeOM3xZACroGmhVrDm6CCYVeLvFoIH1pibxFmAh1pzVLYmmKkbl1mbXvL4GFJCZSXsXpYgPCGaYi5kL6z8
JKLTIQ0qAa1okfodZoDHpl1HPpqc5W8GOZd2elMxfauxUw4QRgVDbiKNq2evrAcTahPOjT4F/
jeqC7QXxTWXQv85OcdIV4HHJImWDKiWLd7jqu+fHW7vbuASym7lJOJc6FeeiBKLKnwwnDIFqNDbNj1j+UzNJbceEBMD8MbLHto1LjcDPNRNr+iwKMV7gckipnqaueQ/
G+XPgn5j0Z1/07wsaHtzzRfsBZzIUBDAOcKWq441C1cwEH/ja6C1hhbyow2HfQkNmS+j0cpy9lti86+U23ZE/7b5UcyH/KcBNFFUCK4k8jmK0hgL9GqLZb/
Scb7v7ft9kvMpj+rtJESMpSyFhoJ9mdi01vo+MMj/4ihNCBm1zeSjQv3KIVMnq0kMOQkqd0+IIqe4hpzws18CBrtNUB/
1D1nsFpTdFMotHYSptqleu751Q88M1suz9GhNggdF3xSE4l/KWwv1nUg5/86TOJmzG1hdU8giW+Ygd99oCWCs/
WNducsd86wf85TXFXy1CCLazW6H70X6NdPSl9PBlqRIoSPHb0xPF3P/
akctXQBuubEvFXYPzpv6azy7iYNa4ETLtrFJDSRgHWojf2k217Fn4I5Ym6PA0bDRtRJr5jalWAt7JY8MHTim5ip84Q7j52077YAkD/caP54F3+lDfPEOFfkJd3J1it6tYkwn8=

# double check that we can still retrieve the correct passphrase from the property using the system’s GPG key pair
# (better to be safe than sorry at this stage!)
user@host:~> zfs list –Ho myhost:key.staging tuebix/encrypted | base64 –D | gpg –d 2> /dev/null
tuebix2019

# everything ok, so change to the new passphrase/key for real
user@host:~> zfs list –Ho myhost:key.staging tuebix/encrypted | base64 –D | gpg –d 2> /dev/null | sudo zfs change–key tuebix/encrypted

# and copy the passphrase/key from staging to the property hosting the live key
user@host:~> zfs list –Ho myhost:key.staging tuebix/encrypted | sudo zfs set myhost:key.active=$(cat /dev/stdin) tuebix/encrypted

# access to the system’s secret GPG key now suffices to access the encrypted filesystem
user@host:~> sudo zfs unload–key tuebix/encrypted
user@host:~> zfs list –Ho myhost:key.active tuebix/encrypted | base64 –D | gpg –d 2> /dev/null | sudo zfs load–key tuebix/encrypted
```

Zusammenfassung

- Native ZFS-Verschlüsselung bietet Sicherheitsfeature, ohne auf Komfortfunktionen verzichten zu müssen
- Schränkt Datenzugriff auf autorisierte Systeme ein
- Ermöglicht Datensicherung auf nicht (voll) vertrauenswürdigen Systemen
- Besonders geeignet für Off-Site-Backups in der Cloud
- Sinnvoll anwendbar von Heimbereich bis Enterprise