Apache HAWQ TDE security integration

hma@pivotal.io 2017-07

Pivotal

Agenda

- TDE introduction
- HAWQ-TDE integration
- Demo
- Future
- Reference
- Q & A

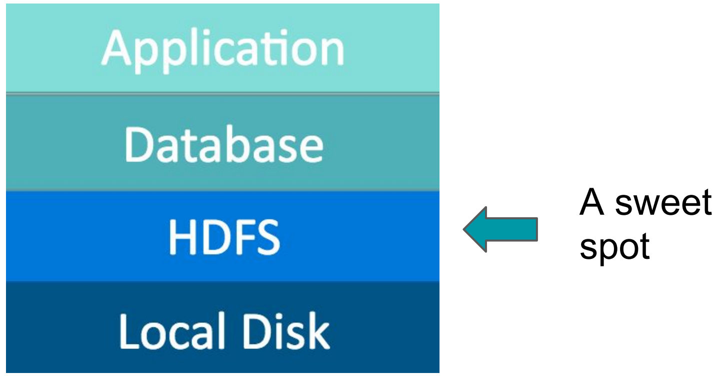
TDE introduction - Attack Vectors

- Physical access
  - Data at-rest on hard drives
- Network sniffing
  - Data in-transit on the network
- Rogue user or admin
  - Insider access to encrypted data, encryption keys

```
22:15:10.354486 arp who-has 192.168.1.254 (00:1b:a9:27:33:32 (oui Unknown))
22:15:10.655673 IP 192.168.3.4.netbios-ns > 192.168.3.255.netbios-ns: NBT U
22:15:11.374842 arp who-has 192.168.1.254 (00:1b:a9:27:33:32 (oui Unknown))
22:15:11.378352 IP 192.168.3.4.netbios-ns > 192.168.3.255.netbios-ns: NBT U
22:15:12.194193 IP 192.168.3.4.netbios-ns > 192.168.3.255.netbios-ns: NBT U
22:15:12.397805 arp who-has 192.168.1.254 (00:1b:a9:27:33:32 (oui Unknown))
22:15:13.217965 IP 192.168.3.4.netbios-ns > 192.168.3.255.netbios-ns: NBT U
22:15:13.320140 arp who-has 192.168.1.254 (00:1b:a9:27:33:32 (oui Unknown))
22:15:13.625631 aa:aa:aa:bb:bb:bb (oui Unknown) > 33:33:00:00:00:01 (oui Un
        0x0000:  6000 0000 0040 3aff fe80 0000 0000 0000  `....@:.........
        0x0010:  0218 4cff feaf bb95 ff02 0000 0000 0000  ..L.............
        0x0020:  0000 0000 0000 0001 8600 0181 ff08 ffff  ................
        0x0030:  00fa 0000 001e                           ......
22:15:13.933856 IP 192.168.3.4.netbios-ns > 192.168.3.255.netbios-ns: NBT U
22:15:14.344078 arp who-has 192.168.1.254 (00:1b:a9:27:33:32 (oui Unknown))
22:15:14.649595 aa:aa:aa:bb:bb:bb (oui Unknown) > 33:33:00:00:00:01 (oui Un
        0x0000:  6000 0000 0040 3aff fe80 0000 0000 0000  `....@:.........
        0x0010:  0218 d4ff fe8b 84d4 ff02 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0001 8600 a0e6 ff08 ffff  ................
        0x0030:  00fa 0000 001e                           ......
22:15:14.753839 IP 192.168.3.4.netbios-ns > 192.168.3.255.netbios-ns: NBT U
22:15:15.368218 arp who-has 192.168.1.254 (00:1b:a9:27:33:32 (oui Unknown))
22:15:15.673580 aa:aa:aa:bb:bb:bb (oui Unknown) > 33:33:00:00:00:01 (oui Un
        0x0000:  6000 0000 0040 3aff fe80 0000 0000 0000  `....@:.........
        0x0010:  0218 82ff fe4b ab26 ff02 0000 0000 0000  .....K.&........
        0x0020:  0000 0000 0000 0001 8600 a234 ff08 ffff  ............4....
        0x0030:  00fa 0000 001e                           ......
22:15:22.640401 arp who-has 192.168.3.4 tell 192.168.3.3
22:15:22.642734 arp who-has 192.168.3.3 tell 192.168.3.4
22:15:26.225302 IP 192.168.3.4.17500 > 192.168.3.255.17500: UDP, length 105
c^C
68 pa
68 pa
0 pac
sams-iPad:~ root#
```

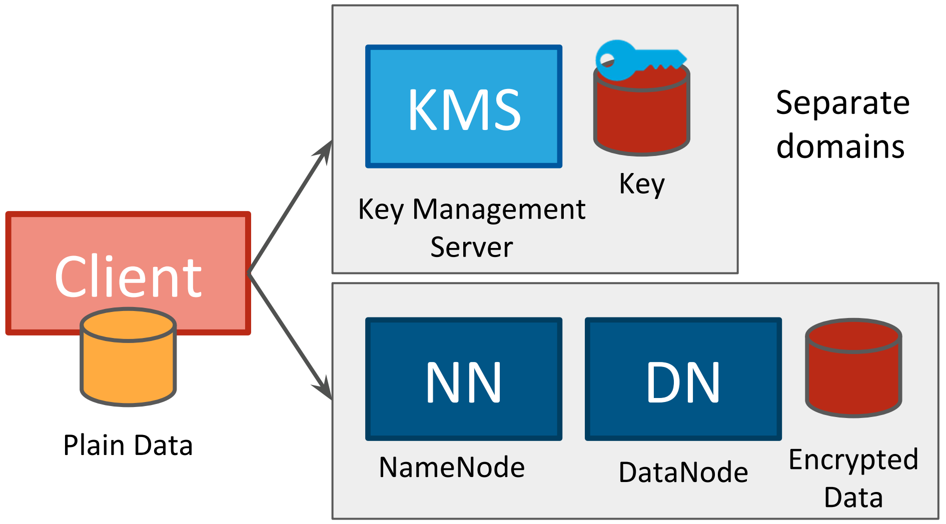
TDE introduction - Background

- Transparent Data Encryption in HDFS
- https://issues.apache.org/jira/browse/HDFS-6134
- *HDFS implements* ***transparent****,* ***end-to-end*** *encryption. Once configured, data read from and written to special HDFS directories is transparently encrypted and decrypted without requiring changes to user application code.*
- Why Encrypt in HDFS?
  - Excellent performance
  - Safe from OS attacks
  - App transparency
  - Easy to deploy



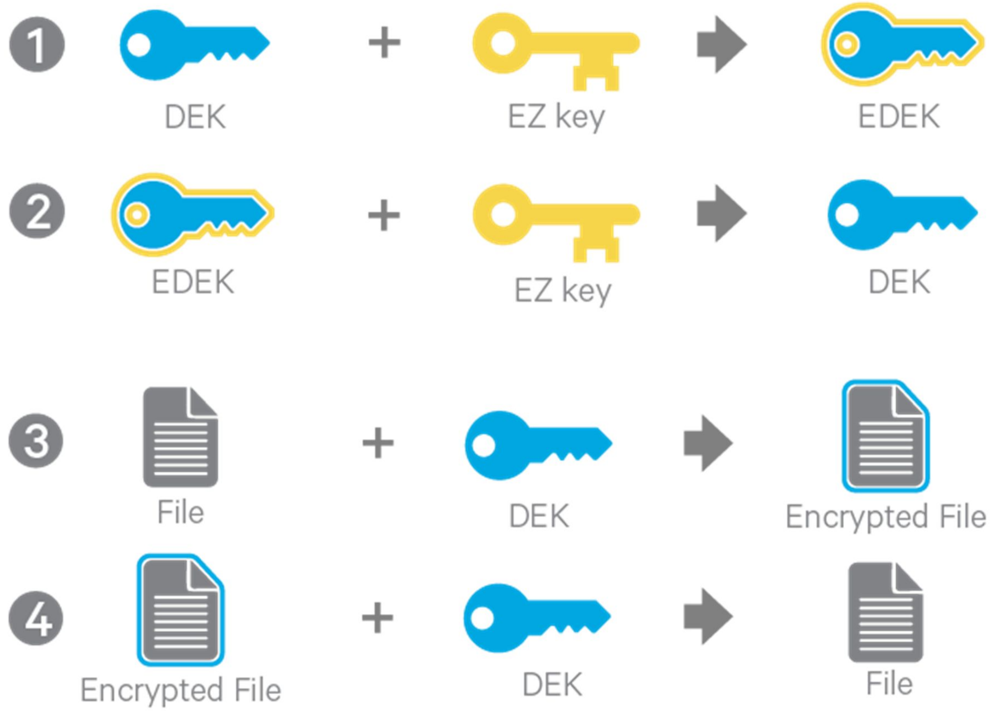
TDE introduction - Components

- Hadoop KMS
  - The key server manage encryption keys.
  - Granular per-file encryption keys and per-key ACLs
    - Rogue user can only leak their own data.
  - Encryption algorithm
    - AES-CTR 128/256
    - High-performance
- HDFS
  - Only stores the encrypted data.
  - Encryption Zone, a HDFS directory
- Separate administrative domains for KMS and HDFS
  - Rogue admins do not have access to both keys and data
- API transparency for existing applications



TDE introduction - Keys

- Encryption Zone Key (**EZK**)
  - Unique key per encryption zone
  - key material is stored in the KMS
- Data Encryption Key (**DEK**)
  - **Unique DEK per file**
  - Used by client to encrypt and decrypt data in HDFS
  - Only handled by client and KMS, never HDFS
  - DataNodes never handle plaintext, only ciphertext
- Encrypted Data Encryption Key (**EDEK**)
  - DEK encrypted with corresponding EZK
  - Generated by KMS for NameNode
  - Stored in HDFS metadata as a file xattr



TDE introduction - Back to Attack Vectors

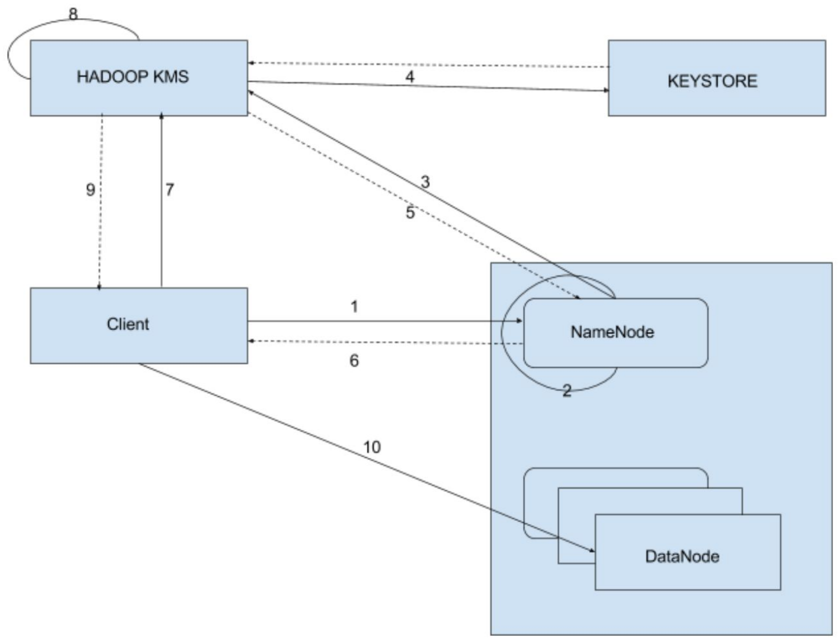
- Compromised EZK
  - Also requires EDEK and ciphertext
  - Only leaks a single zone
- Compromised EDEK
  - Also requires EZK and ciphertext
  - Only leaks a single file
- Compromised ciphertext
  - Also requires corresponding EZ key and EDEK
  - Only leaks a single file
- Attack vectors
  - Physical access: ciphertext, EDEK, etc.
  - Network sniffing: ciphertext, EDEK, etc.
  - Rogue admin: only one domain.

HAWQ-TDE integration - Introduction

- https://issues.apache.org/jira/browse/HAWQ-1193
- HAWQ data is stored in HDFS
  - `hdfs://namenode:9000/hawq_default`
  - The whole directory(filespace) as an Encryption Zone
- libhdfs3
  - **A C/C++ native HDFS client**, written by Pivotal
  - HAWQ visit HDFS by it, need to upgrade to support TDE
  - https://github.com/apache/incubator-hawq/tree/master/depends/libhdfs3
- Other HAWQ components don't need to be modified
  - API transparency

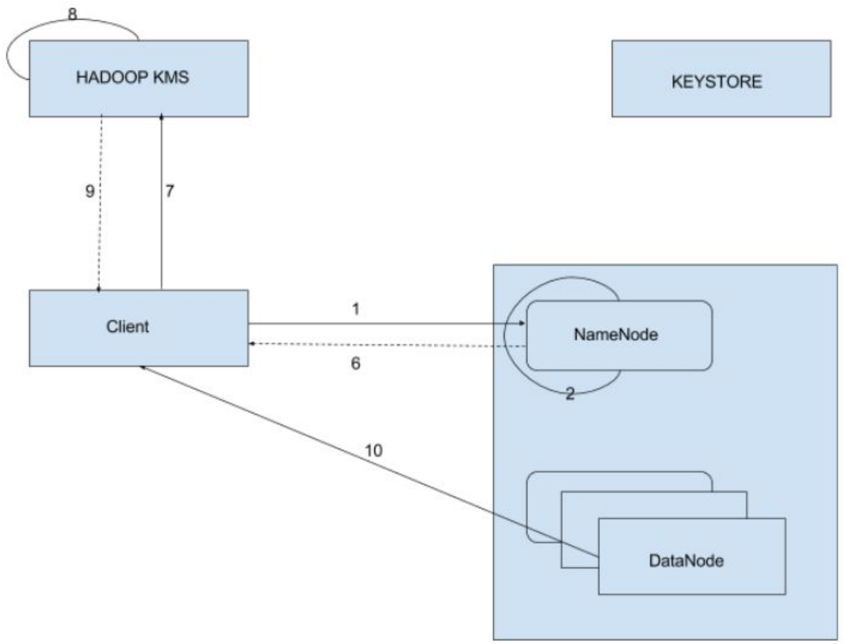
HAWQ-TDE integration - Write process

1. The client issues a command to store a new file.
2. The NameNode checks if the file is under the Encryption Zone.
3. The NameNode passes the EZK name to KMS and requests KMS to create a new DEK.
4. The KMS retrieves the EZK from the key store and encrypts the DEK using EZK to generate the EDEK.
5. The KMS provides the EDEK to the NameNode and NameNode persists the EDEK as an extended attribute for the file metadata.
6. The NameNode provides the EDEK to the HDFS client.
7. The HDFS client sends the EDEK to the KMS, requesting the DEK.
8. The KMS checks if the user running the HDFS client has access to the EZK.
9. The KMS decrypts the EDEK using the EZK and provides the DEK to the HDFS client.
10. The HDFS client encrypts data using the DEK and writes the encrypted data blocks to HDFS.



HAWQ-TDE integration - Read process

1. The client issues a command to read a new file.
2. The NameNode checks if the file is under the Encryption Zone.
3. No needed
4. No needed
5. No needed
6. The NameNode provides the EDEK to the HDFS client.
7. The HDFS client sends the EDEK to the KMS, requesting the DEK.
8. The KMS checks if the user running the HDFS client has access to the EZK.
9. The KMS decrypts the EDEK using the EZK and provides the DEK to the HDFS client.
10. The HDFS client encrypts data using the DEK and writes the encrypted data blocks to HDFS.



HAWQ-TDE integration - Write/Read summary

- Writing a file
  - Client asks NameNode to create a file in an Encryption Zone
  - NameNode gets a new EDEK from KMS
  - NameNode stores EDEK in file’s xattr and returns EDEK to client
  - Client asks KMS to decrypt EDEK
  - KMS decrypts EDEK and returns DEK to client
  - Client uses DEK to encrypt file data
- Reading a file
  - Client asks NameNode for file’s EDEK
  - Client asks KMS to decrypt EDEK
  - KMS decrypts EDEK and returns DEK to client
  - Client uses DEK to decrypt data

HAWQ-TDE integration - Conclusion

- Hadoop TDE bring end-to-end, client-side encryption to HAWQ
  - Data protected at-rest and in-transit
- The whole HAWQ data directory as an Encryption Zone
  - It cannot transfer between Encryption Zone and Normal Zone
- Transparent
  - API transparent for existing applications
  - Performance transparent
    - Less 10% performance impact
- Separate administrative domains for key management and HDFS

Demo

```
// as the key administrator user

$ hadoop key create mykey

// as the HDFS administrator user

$ hadoop fs —mkdir /users/hongxu/my-ez

$ hadoop fs —chown hongxu /users/hongxu/my-ez

$ hadoop fs —chmod 700 /users/hongxu/my-ez

$ hadoop crypto —createZone —keyName mykey —path /users/hongxu/my-ez
```

Demo (cont.)

```
// as the encryption user

$ echo “hello world” > hello-world.txt

$ hadoop fs −put hello-world.txt /users/hongxu/my-ez/

$ hadoop fs −cat /users/hongxu/my-ez/hello-world.txt

    hello world

// as the HDFS admin

$ hadoop fs −cat /.reserved/raw/users/hongxu/my-ez/hello-world.txt

    ��嶗]□N^���+���
```

Future & Reference

- TDE with HDP 2.6 support will be released in HAWQ 2.3.0.0
- Future
  - KMS security: Kerberos, HTTPS
  - Finer granularity (tablespace, table)
- Reference
  - http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/TransparentEncryption.html
  - https://www.slideshare.net/Hadoop_Summit/transparent-encryption-in-hdfs
  - https://issues.apache.org/jira/browse/HDFS-6134
  - https://issues.apache.org/jira/browse/HAWQ-1193

Thanks
Q & A

Pivotal is hiring: GPDB, HAWQ, **pivotalrnd_china_jobs@pivotal.io**

Welcome to join!

Pivotal