Encryption and Anonymization in Hadoop

Current and Future needs

Sept-28-2015 ApacheCon, Budapest
Agenda

- Need for data protection – Encryption and Anonymization
- Current State of Encryption in Hadoop
- Demo
- Future focus areas for the community
Speakers

bosco@apache.org
Chief Security Architect,
Hortonworks
Committer - Apache Ranger
and Apache Hawq

bganesan@apache.org
Sr Director, Enterprise Security
Hortonworks
Committer - Apache Ranger
## Security today in Hadoop

### Hadoop Ecosystem

<table>
<thead>
<tr>
<th>Centralized Security Administration w/ Ranger</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authentication</strong> Who am I/prove it?</td>
</tr>
<tr>
<td>• Kerberos</td>
</tr>
<tr>
<td>• API security with Apache Knox</td>
</tr>
<tr>
<td><strong>Authorization</strong> What can I do?</td>
</tr>
<tr>
<td>• Fine grain access control with Apache Ranger</td>
</tr>
<tr>
<td><strong>Audit</strong> What did I do?</td>
</tr>
<tr>
<td>• Centralized audit reporting w/ Apache Ranger</td>
</tr>
<tr>
<td><strong>Data Protection</strong> Can data be encrypted at rest and over the wire?</td>
</tr>
<tr>
<td>• Wire encryption in Hadoop</td>
</tr>
<tr>
<td>• HDFS, Hbase encryption</td>
</tr>
</tbody>
</table>
Data Protection

Encryption and Anonymization
Why is Encryption at Rest required?

- Sensitive data could be stored in Hadoop
- Compliance or external regulation may mandate encryption, example PCI (Retail, Consumer) or HIPAA (Healthcare)
- Cost of not encrypting is increasing
- Enhanced Security
  - Added layer on top of authentication (passwords) and authorization (ACLs)
- Protect certain rogue administrators from accessing sensitive data
Available Hadoop Encryption Options

- Hbase
- Custom
- HDFS
- OS
OS Level Encryption – LUKS/DM-CRYPT

Why it helps?
- Encrypts entire disk volume – all data is encrypted
- Simpler setup, native OS and Vendor solutions available

Cons
- Performance challenges
- Admin can still see raw data
HDFS Transparent Encryption Solution

Why it helps?
• Encrypt only specific data
• Different access control levels
• Transparent to end application, little changes needed
• Auditing of Key Access
HDFS Encryption – Protect Application Data

Guidelines
- Encrypt Hive, Hbase data stored in HDFS
- Specific changes in Hive to ensure scratch dir is encrypted
- Separate admins in HDFS, Yarn, Oozie
- Spark application logs should be in EZ
Ranger KMS – Centralized Key Management

Key Detail

Key Name: audit

Key Management

Select Service: cl1_kms

Search for your keys...

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Cipher</th>
<th>Version</th>
<th>Attributes</th>
<th>Length</th>
<th>Created Date</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit</td>
<td>AES/CTR/NoPadding</td>
<td>1</td>
<td>key.acl.name → audit</td>
<td>128</td>
<td>09/15/2015 03:28:08 PM</td>
<td></td>
</tr>
<tr>
<td>sensitivefolder</td>
<td>AES/CTR/NoPadding</td>
<td>1</td>
<td>key.acl.name → SensitiveFolder</td>
<td>128</td>
<td>08/06/2015 01:30:44 PM</td>
<td></td>
</tr>
</tbody>
</table>
HDFS TDE Workflow

1. **Client**
2. **NN, DN**
3. **Ranger KMS**

- **Create Encryption Zone**
- **NN marks folder as EZ**
- **Provide EZ Keys**
- **Create EZ Keys**
HDFS TDE Workflow – Write a File

1. **Client**
   - Client request to write to EZ

2. **NN, DN**
   - NN does access check.

3. **Ranger KMS**
   - Create DEK and encrypt with EZ Key

---

4. **Receive EDEK. Request DEK**

---

5. **Encrypt data and write to DN.**

---

6. **Send block information to client. EDEK stored with file**

---

7. **Decrypt EDEK, provide DEK**
HDFS TDE Workflow – Read a File

Client

Client request to read from EZ

NN, DN

NN does access check. Provide data, EDEK

Ranger KMS

Decrypt EDEK, provide DEK

Receive EDEK. Request DEK

Use DEK to read file data

Request DEK

Decrypt EDEK

Provide DEK

Read file data
Hbase Encryption in 0.98

Why it helps?

- Hfile encrypted and stored in disk
- Per CF configuration
- Keys stored in Java keystore
Demo

Don Bosco Durai
Future Work

Focus areas for the community
Encryption and Anonymization - Future Focus Areas

- Hive Column Encryption
- Solidifying Hbase Encryption
- Kafka and Solr Encryption
- Need for Tokenization/Masking
Hive Column Encryption

- Being discussed in the community. Apache JIRA # ORC-14
- Handled at the ORC layer
- Elegant solution. Encryption done after ORC compression.
- Each columns are different files and they can be encrypted with different key
- Leverage keyprovider API. Potentially can use Hadoop/Ranger KMS

How it will help?

- Encrypt fields instead of file
- Data protected in HDFS as well as OS layer
Kafka Encryption

- Discussion going on in Kafka community
- Two possible approaches
  - Broker encrypts and stores the data
  - Client(s) encrypt/decrypt the data
- Pros with client side encrypt/decrypt
  - No encryption/decryption overhead on Broker side
  - Keys not available on Broker, so data safe from everyone
  - No need for wire encryption
- Cons with client side encrypt/decrypt
  - Compaction/compression not effective with encrypted data.
  - Needs protocol change and update client libraries.
Solr Encryption

- No active discussion currently
- Will be good to have native support
- Index files could be encrypted/decrypted just like ORC
- Could be integrated with external KMS (Hadoop/Ranger)

How it will help?
- Sensitive data could be stored in indexes, may need to be encrypted
- Higher granularity than OS or HDFS encryption
Beyond Encryption... Anonymization¹?

- Tokenization – Replace a sensitive field (eg: card number) with some other value. Could be format preserving or random unique value.

- Redaction - Mask sensitive data (eg: card numbers can be changed to xxxx xxxx xxxx 1234)


How it helps?

- Protect sensitive data beyond access control
- Field level control
- Enable compliance to privacy laws
Where is it applicable?

- Sensitive data in HDFS file
- Column values in Hive or Hbase
- Field values in Solr
- Messages in Kafka or NiFi
How?

- **Tokenize on source**
  - Tokenize while ingesting data (Flume, NiFi, Sqoop, etc.)
  - Data stored tokenized, so safe to give access to others.
  - Selective users can de-tokenize if needed

- **Tokenize/Mask on read**
  - E.g. select name, mobile_number from customer;
  - Based on policy, if user is Data Scientist, then tokenize/mask data before returning

<table>
<thead>
<tr>
<th>Name</th>
<th>Returned (Format Preserved)</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>415-123-4567</td>
<td>415-682-5638</td>
</tr>
<tr>
<td>Jane Smith</td>
<td>408-123-4567</td>
<td>408-802-4027</td>
</tr>
<tr>
<td>Mary Pick</td>
<td>650-123-4567</td>
<td>650-865-6921</td>
</tr>
</tbody>
</table>
Questions ??