Pluggable Security in Hadoop ... and beyond
About presenters

Yuliya Feldman (yuliya@dremio.com)
- S/W Engineer at Dremio
- Lead MapR security initiative
- Started working in Big Data field at 2009
- 20+ years experience in S/W development
- Advanced degree in Applied Mathematics from Kharkov University (Ukraine)

Bill O’Donnell (bodonnell@mapr.com)
- Strategic Security Architect at MapR
- Previously Chief Security Architect for IBM WebSphere
- 20+ years experience in S/W development
- Computer Science degree from University of Wisconsin
Agenda

1. 4 pillars of security

2. Double Click on Authentication
   - Current State of affairs - Kerberos
   - Need to support multiple authentication mechanisms, why?
   - How SASL, JAAS, PAM are used
   - What needs to be implemented to achieve fully pluggable authentication
   - Real world examples: MapR

3. Lessons from building security features at Dremio
4 Pillars of Security

1. Ubiquitous Data Protection
   - Encryption for Data in Motion
     - Within a Cluster
     - Between Client and Cluster
   - Encryption for Data at Rest

2. Flexible Authentication
   - Wire-level authentication for all services in the cluster
   - Integration with LDAP, etc
   - Basic Auth, Certificate authentication, etc

3. Robust Auditing
   - All events recorded immediately in audit files
   - Includes data access and administrative actions

4. Granular Authorization
   - Protect files, tables, columns, and management objects
Authentication

“you are who you say you are”
Hadoop and security - pluggable from early days

- Original version of Hadoop did not have security

- Necessity to process sensitive data (PII, PCI), data that was owned by different organizations within the same framework and network drove security requirements

- Slides presented at Kerberos Conference in 2010 by then Hadoop team from Yahoo
  
  - From “Solution” slide: “Integrate Hadoop with Kerberos”
  
  - From “Out Of Scope” slide: “Non-Kerberos Authentication – Much easier now that framework is available”
Hadoop and security

Recap:

- Security was introduced to Hadoop in 2010

- Security was introduced to vast majority of Hadoop ecosystem projects such as Hive, Oozie, HBase, etc, as well as complementary projects like Zookeeper.

Current state of affairs (2017):

- “Out Of Scope” slide: “Non-Kerberos Authentication – Much easier now that framework is available”
Consequences of no pluggable security mechanisms

- Handling security is a rare skill, so lots of “copy/paste” patterns
- Many projects treat security == Kerberos
- Lots of hard-coding security methods occur to create spaghetti of “if/elseif/…/else” code
- No way to add new mechanism to handle security without code changes
Examples of security == Kerberos

```java
boolean kerberosAuthOn = ConfigurationService.getBoolean(conf, KERBEROS_AUTH_ENABLED);

LOG.info("Oozie Kerberos Authentication [{0}]", (kerberosAuthOn) ? "enabled" : "disabled");
if (kerberosAuthOn) {
    kerberosInit(conf);
} else {
    Configuration ugiConf = new Configuration();
    ugiConf.set("hadoop.security.authentication", "simple");
    UserGroupInformation.setConfiguration(ugiConf);
}
```

(https://github.com/apache/oozie/blob/9c0516608bdd0787b1fac3a69ccfe9d36cbb2d15/core/src/main/java/org/apache/oozie/service/HadoopAccessorService.java#L139)
The pros and cons of Kerberos

• Pros
  • One of the most secure protocols available
  • Supports interoperability across platform and applications

• Cons
  • **Nothing is simple about Kerberos**
  • Requires Kerberos expertise across security team for entire ecosystem
  • Difficult to debug
  • Requires high availability
  • Kerberos has very strict time requirements that can cause problems
  • Complications arise with each network attached service, requiring different host names and different sets of keys
The reality of Kerberos

Kerberos – Requires deep technical depth, a deep learning curve, and configuring can be extremely complex.

Scripts and Documentation – Can make initial configuring Kerberos easier, but making mistakes following instructions or making changes down the road can be very challenging in debugging and troubleshooting.

Alternatives – Provide plug-ability to allow for alternative solutions to be developed through open source or 3rd party solutions.
What can we leverage that exists in Hadoop today to provide Kerberos alternatives
Security frameworks used in Hadoop today

Frameworks:

- SASL (Simple Authentication and Security Layer)
  - To provide authentication and/or encryption across the wire

- JAAS (Java Authentication and Authorization Service)
  - To provide identity of a user attempting to run Java code

- PAM (Pluggable Authentication Module)
  - Used under umbrella of JAAS to authenticate user identity
SASL (Simple Authentication and Security Layer)

“The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms. It provides a structured interface between protocols and mechanisms. The resulting framework allows new protocols to reuse existing mechanisms and allows old protocols to make use of new mechanisms. The framework also provides a protocol for securing subsequent protocol exchanges within a data security layer.”

JAAS (Java Authentication and Authorization Service)

The Java Authentication and Authorization Service (JAAS) is a set of application program interfaces (APIs) that can determine the identity of a user or computer attempting to run Java code and ensure that the entity has the right to execute the functions requested.

JAAS authentication is performed in a pluggable fashion. This permits applications to remain independent from underlying authentication technologies. New or updated authentication technologies can be plugged under an application without requiring modifications to the application itself. Applications enable the authentication process by instantiating a LoginContext object, which in turn references a Configuration to determine the authentication technology(ies), or LoginModule(s), to be used in performing the authentication. Typical LoginModules may prompt for and verify a username and password. Others may read and verify a voice or fingerprint sample.
A pluggable authentication module (PAM) is a mechanism to integrate multiple low-level authentication schemes into a high-level API. It allows programs that rely on authentication to be written independently of the underlying authentication scheme.
DREMIO

SASL reminder

Provider

Concrete Provider

SaslClientFactory

ConcreteSaslClientFactory

SaslServerFactory

ConcreteSaslServerFactory

SASL
- createSaslClient
- createSaslServer

SaslClient createSaslClient(
  Providers Security.getProviders();
  Foreach Provider {
    ConcreteSaslClientFactory = loadFactory(Provider,FactoryClassname)
    ConcreteSaslClientFactory.createSaslClient(...)
  }
)

SaslServer createSaslServer(
  Providers Security.getProviders();
  Foreach Provider {
    ConcreteSaslServerFactory = loadFactory(Provider,FactoryClassname)
    ConcreteSaslServerFactory.createSaslServer(...)
  }
)
SASL Interactions

**Client Actions**

- ClientSASLInit
- SASL Authentication
- SASL Privacy and Integrity
- SASL session release (end connection)

**Server Actions**

- ServerSASLInit
- SASL Authentication
- SASL Privacy and Integrity
- SASL session release (end connection)

- Authentication Challenge Exchange
- Session Key creation
- Encoded messages
SASLClient

String getMechanismName()
byte[] evaluateChallenge(byte[] challenge)
boolean isComplete()
Object getNegotiatedProperty(String propName)
byte[] wrap(byte[] outgoing, int offset, int len)
byte[] unwrap(byte[] incoming, int offset, int len)
void dispose()
boolean hasInitialResponse()

SASLServer

String getMechanismName()
byte[] evaluateResponse(byte[] response)
boolean isComplete()
Object getNegotiatedProperty(String propName)
byte[] wrap(byte[] outgoing, int offset, int len)
byte[] unwrap(byte[] incoming, int offset, int len)
void dispose()
String getAuthorizationID()
SASL in Hadoop RPC

RpcClient (Client) ──────────────────── RpcProtocol ──────────────────── RpcServer (Server)

Connection

RPC Header:
- "hrpc" 4 bytes
- Version (1 byte)
- Service Class (1 byte)
- AuthProtocol (1 byte)
SASL in Hadoop RPC

Main Classes:

SaslRpcServer

- AuthMethod

SaslRpcClient

```java
public static enum AuthMethod {
    SIMPLE((byte) 80, ""),
    KERBEROS((byte) 81, "GSSAPI"),
    @Deprecated
    DIGEST((byte) 82, "DIGEST-MD5"),
    TOKEN((byte) 82, "DIGEST-MD5"),
    PLAIN((byte) 83, "PLAIN"),
```

SASL in Hadoop RPC (SaslRpcClient: createSaslClient)

```java
final AuthMethod method = AuthMethod.valueOf(authType.getMethod());
switch (method) {
    case TOKEN: {
        ...
        break;
    }
    case KERBEROS: {
        ...
        break;
    }
    default:
        throw new IOException("Unknown authentication method " + method);
}
String mechanism = method.getMechanismName();

return Sasl.createSaslClient(
    new String[] { mechanism }, saslUser, saslProtocol, saslServerName, 
    saslProperties, saslCallback);
```

WHY???
SASL in Hadoop RPC (SaslRpcServer::SaslServer create())

switch (authMethod) {
    case "TOKEN": {
        ...
        break;
    }
    case "KERBEROS": {
        ...
        break;
    }
    default:
        // we should never be able to get here
        throw new AccessControlException(
            "Server does not support SASL " + authMethod);
}

saslServer = saslFactory.createSaslServer(mechanism, protocol, serverId, saslProperties, callback);
Wait, there is more ...

UserGroupInformation
JAAS - UserGroupInformation relationship

UserGroupInformation represents User Identity and to achieve that it implements and/or uses following:

- Common Classes
  - Subject, Principal, Credential
- Authentication Classes and Interfaces
  - LoginContext, LoginModule, CallbackHandler, Callback,
  - Configuration ([http://docs.oracle.com/javase/6/docs/api/javax/security/auth/login/Configuration.html](http://docs.oracle.com/javase/6/docs/api/javax/security/auth/login/Configuration.html))
JAAS - UserGroupInformation relationship

javax.security.auth.login.Configuration

A Configuration object is responsible for specifying which LoginModules should be used for a particular application, and in what order the LoginModules should be invoked.

UserGroupInformation class has shy of 100 lines of code to hardcode what Configuration object suppose to represent

**Pros:**
- No need for external configuration file

**Cons:**
- No way to add new LoginModule or modify existing
- A lot of lines of code that need to be modified with every change of thought
Recap

Great things:

1. Using JAAS to create user identities
2. Using SASL to do mutual client/server authentication

Still missing:

1. JAAS Configuration is hardcoded
2. Authentication methods and subsequently SASL client/server implementations are limited to TOKEN (MD5) and KERBEROS
What could we do to make things better?
JAAS - UserGroupInformation relationship

Cons:

- No way to add new LoginModule or modify old
- A lot of lines of code that need to be modified with every change of thought
Configuration file example

hadoop_kerberos {
  com.sun.security.auth.module.Krb5LoginModule required
  useTicketCache=true
  renewTGT=true
  doNotPrompt=true;
  org.apache.hadoop.security.login.GenericOSLoginModule required;
  org.apache.hadoop.security.login.HadoopLoginModule required;
};

hadoop_kerberos_keytab {
  com.sun.security.auth.module.Krb5LoginModule required
  refreshKrb5Config=true
  doNotPrompt=true
  useKeyTab=true
  storeKey=true;
  org.apache.hadoop.security.login.GenericOSLoginModule required;
  org.apache.hadoop.security.login.HadoopLoginModule required;
};
Dealing with AuthMethod, AuthenticationMethod and others like them

- A lot of first class citizen classes, utility classes to deal with creating user identity, on wire authentication
- How about trying to consolidate quite a few of those or at least give more flexible way of dealing with them
Truly Pluggable way

- **RpcAuthMethod** is abstract class

  - We really already kind of have implementations for KERBEROS, DIGEST, SIMPLE - just scattered around in different classes

  - Anybody can add implementation for their own extension of RpcAuthMethod and use it for their method of authentication and beyond

- **RpcAuthRegistry** class

  - To register RpcAuthMethods

  - Preregister ones for KERBEROS, DIGEST, SIMPLE

  - **public static synchronized void** addRpcAuthMethod(RpcAuthMethod authMethod)

  - Can register your own
public abstract class RpcAuthMethod {

public String getMechanismName() ;

public AuthenticationMethod getAuthenticationMethod();

public CallbackHandler createCallbackHandler();

public String[] loginModules();

public UserGroupInformation getUser(UserGroupInformation ticket);

public void writeUGI(UserGroupInformation ugi, Builder ugiProto);

public UserGroupInformation getAuthorizedUgi(String authorizedId, SecretManager secretManager) throws IOException;

public boolean shouldReLogin() throws IOException;

public void reLogin() throws IOException;

public boolean isProxyAllowed();

public boolean isNegotiable();

public boolean isSasl();

public String getProtocol() throws IOException;

public String getServerId() throws IOException;

public SaslClient createSaslClient(Map<String, Object> saslProperties) throws IOException;

public SaslServer createSaslServer(Connection connection, Map<String, Object> saslProperties) throws IOException, InterruptedException;
final AuthMethod method = AuthMethod.valueOf(authType.getMethod());
switch (method) {
    case TOKEN: {
        ...
        break;
    }
    case KERBEROS: {
        ...
        break;
    }
    default: {
        throw new IOException("Unknown authentication method " + method);
    }
}
String mechanism = method.getMechanismName();

return Sasl.createSaslClient(
    new String[] { mechanism }, saslUser, saslProtocol, saslServerName, saslProperties, saslCallback);

WHY???
Let’s reformat that code a bit with what we know now

```java
private SaslClient createSaslClient(SaslAuth authType) {
    final RpcAuthMethod method = RpcAuthRegistry.getAuthMethod(authType.getMethod());
    switch (method.getAuthenticationMethod()) {
        case TOKEN: {
            ...
            saslProperties.put(SaslRpcServer.SASL_AUTH_TOKEN, token);
            break;
        }
        case KERBEROS: {
            ...
            saslProperties.put(SaslRpcServer.SASL_KERBEROS_PRINCIPAL, serverPrincipal);
            break;
        }
    }

    return method.createSaslClient(saslProperties);
}
```
One more recap - SASL in Hadoop RPC (server)

```java
switch (authMethod) {
    case TOKEN: {
        ...
        break;
    }
    case KERBEROS: {
        ...
        break;
    }
    default:
        // we should never be able to get here
        throw new AccessControlException("Server does not support SASL "+ authMethod);
}

saslServer = saslFactory.createSaslServer(mechanism, protocol, serverId, 
    saslProperties, callback);
```
Let's reformat that code a bit with what we know now:

```java
public SaslServer create(final Connection connection,  
                        Map<String, Object> saslProperties,  
                        SecretManager<TokenIdentifier> secretManager)  
                        throws IOException, InterruptedException {
    if (secretManager != null) {
        saslProperties.put(SaslRpcServer.SASL_AUTH_SECRET_MANAGER, secretManager);
    }

    return authMethod.createSaslServer(connection, saslProperties);
}
```

authMethod is of type RpcAuthMethod.
What does it take to have your security plugin to be really plugged in :) 

Let’s look at the inventory:

1. FooBarLoginModule implements LoginModule
2. FooBarRpcAuthMethod extends RpcAuthMethod
3. FooBarSaslClient implements SaslClient
4. FooBarSaslServer implements SaslServer
5. FooBarSaslProvider extends Provider
6. SaslFooBarClientFactory extends SaslClientFactory
7. SaslFooBarServerFactory extends SaslServerFactory
Wait, there is still more ...

How about web based authentication?

- AuthenticationHandler interface (server side)
- Authenticator interface (client side)

You can create your own implementations or use existing ones.

You don’t have much choices with browsers besides SPNEGO and BasicAuth though

But with the client apps where you can provide additional headers - why not.
What is done so far

- MapR’s non-Kerberos security mechanism follows paradigm described in previous slides. In production 3+ years.

- JIRA submitted to Zookeeper community to follow the same paradigm, as Zookeeper is not using any of Hadoop constructs (https://issues.apache.org/jira/browse/ZOOKEEPER-2159)

- JIRA submitted to Hadoop community (https://issues.apache.org/jira/browse/HADOOP-14182)
Dremio’s security capabilities

- Dremio is a stealth data analytics company
- Capable of running on dedicated hardware or in Hadoop cluster
- All security topics are relevant, including authentication
- We are extending Hadoop’s Impersonation capabilities to non-Hadoop sources
- We have a security model layered on Hadoop’s security for fine-grained control (e.g., grant granular access to non-authenticated sources)
- See me after the talk to learn more :-)}
Sources to look at

https://issues.apache.org/jira/browse/ZOOKEEPER-2159

https://github.com/mapr/hadoop-common/tree/release-2.7.0-mapr-1602

https://issues.apache.org/jira/browse/HADOOP-14182
Aknowledgements

All this work would not be possible without significant contribution of the following people

Keys Botzum
Aditya Kishore
Santosh Marella
Venki Korukanti
Mehant Baid
Na Yang
And quite a few others folks
Bibliography


https://docs.oracle.com/javase/8/docs/technotez/guides/security/jaas/JAASLMDevGuide.html

http://docs.oracle.com/javase/6/docs/technotez/guides/security/jaas/JAASRefGuide.html

http://docs.oracle.com/javase/6/docs/api/javax/security/auth/login/Configuration.html

https://github.com/apache/oozie

https://github.com/apache/hadoop/tree/trunk/hadoop-common-project

https://issues.apache.org/jira/browse/ZOOKEEPER-2159

https://github.com/mapr/hadoop-common/tree/release-2.7.0-mapr-1602
Thank you
Q&A