Keep Your Secrets Secret

Kerberos for Java developers

@jckwart

hazelcast

Josef Cacek

- Security Engineer / Java Developer at Hazelcast
- Former Red Hatter
- Father
- Runner
- Open-source contributor

@jckwart

hazelcast

Agenda

- Kerberos protocol basics
- GSS-API and its Java binding
- Tools and Debugging

bit.ly/devconf_2020_kerberos

Icebreakers

1. Are you using Kerberos for authentication?

Icebreakers

II. Do you know JAAS API (LoginModule, ...)?

Basics / Terminology

Kerberos protocol

- Network authentication protocol
- roles
  - Key Distribution Center (KDC)
  - Client
  - Server/Service
- Single-sign-on approach
  - old, but still popular in many companies
- Kerberos v5 - RFC-4120

Protocol key points

- passwords and secret keys are not sent over the network
  - **tickets** are used instead
- supports credentials delegation (impersonation)
- clients and services share their secrets with KDC
  - symmetric cryptography
- protection against replay attacks
  - replay cache
  - clock skew check
- server doesn't need to talk to KDC to authenticate clients

Protocol key points

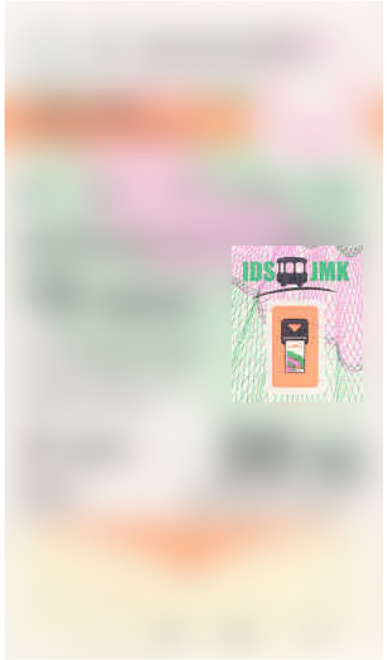
- passwords and secret keys are not sent over the network
  - **tickets** are used instead
- supports credentials delegation (impersonation)
- clients and services share their secrets with KDC
  - symmetric cryptography
- protection against replay attacks
  - replay cache
  - clock skew check
- server doesn't need to talk to KDC to authenticate clients

Traps

- time on all 3 parties has to be synchronized
- KDC is a single point of failure
- handling service hostnames is error prone

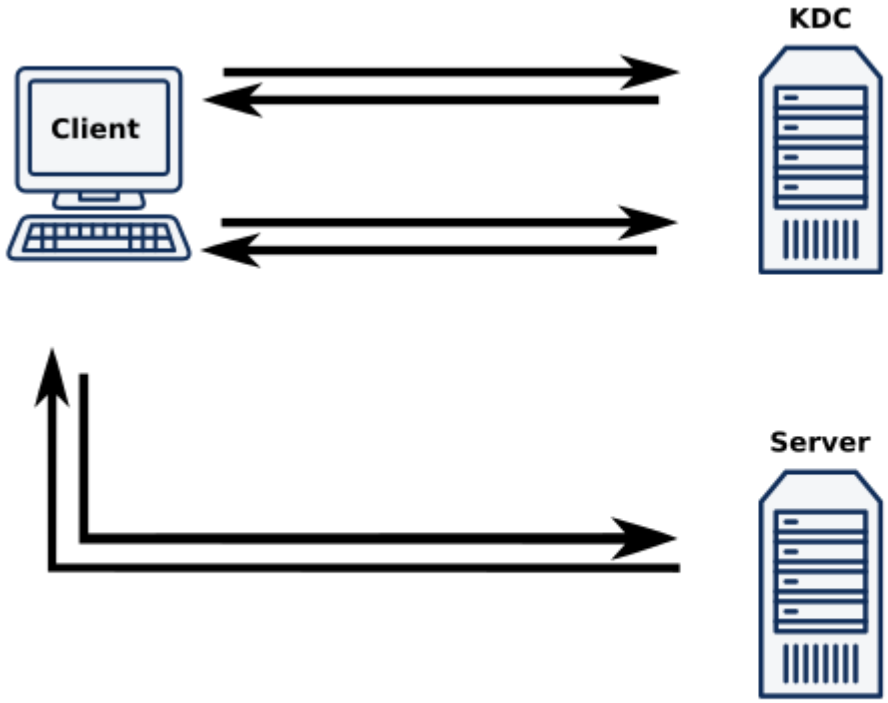
Ticket

- data structure - partly encrypted
- plain
  - realm name
  - server name
- encrypted (using server's secret key)
  - client name
  - session key
  - validity period
  - flags and additional information



Protocol overview

- I. Client authenticates in KDC
- II. Client requests from KDC a ticket for a server
- III. Client authenticates to the server by presenting a valid ticket



Protocol overview - I. Client authenticates in KDC

```
AS_REQ
C->K: Hi, C is here. Could we talk together?


AS_REP
K->C: Nice to see you. I remember you. I have 2 packages for you:
        * your session key - encrypted, of course
        * ticket to talk to me next time - it's called TGT



C=Client
K=Key Distribution Center
S=Server
```

@jckwart

hazelcast

Protocol overview - II. Client requests ticket for accessing a server

```
TGS_REQ
C->K: Hi again, it's me your lovely C.
      I have this TGT ticket and I know our session key.
      Give me a ticket for server S. Please!

TGS_REP
K->C: Let me check. Sure, here you are. You need these 2 packages
        * your client-server session key - encrypted by Client/TGS session key, of course
        * ticket for the server S - it's called the service ticket (ST)



C=Client
K=Key Distribution Center
S=Server
```

Protocol overview - III. Client authenticates to the Server

```
AP_REQ
C->S: Hi S, here am I, your dear C.
      I have this service ticket ST and I know our session key.
      Talk to me!

AP_REP
S->C: Let me check. Sure. I was able to decrypt the ST and verify you know the session key.
      Now I know the session key too, and you can verify it too. Just if you want.
```

C=Client
K=Key Distribution Center
S=Server

Tools and APIs

@jckwart

hazelcast

Tools

- Unix/Linux
  - `kinit` - obtain and cache Kerberos ticket-granting tickets
  - `klist` - display the entries in the local credentials cache and keytab
  - `kdestroy` - destroy user's active tickets by removing credentials cache
  - and more: `kvno`, `kadmin`
- Windows:
  - `klist`
  - other: `ktpass`, `ksetup`
  - Active Directory module in PowerShell

Tools

- Unix/Linux
  - `kinit` - obtain and cache Kerberos ticket-granting tickets
  - `klist` - display the entries in the local credentials cache and keytab
  - `kdestroy` - destroy user's active tickets by removing credentials cache
  - and more: `kvno`, `kadmin`
- Windows:
  - `klist`
  - other: `ktpass`, `ksetup`
  - Active Directory module in PowerShell

Servers

- MIT Kerberos
- MS Active Directory
- Heimdal

Tools

- Unix/Linux
  - `kinit` - obtain and cache Kerberos ticket-granting tickets
  - `klist` - display the entries in the local credentials cache and keytab
  - `kdestroy` - destroy user's active tickets by removing credentials cache
  - and more: `kvno`, `kadmin`
- Windows:
  - `klist`
  - other: `ktpass`, `ksetup`
  - Active Directory module in PowerShell

Servers

- MIT Kerberos
- MS Active Directory
- Heimdal

Kerberos related tools in Java runtime on Windows

- `kinit`, `klist`
- `ktab` - manage the principal names and service keys stored in a local keytab

Configuration in /etc/krb5.conf

```
[libdefaults]
    default_realm = ACME.COM

[realms]
    ACME.COM = {
        kdc = kwartad.eastus.cloudapp.azure.com
    }
    TEST.REALM = {
        kdc = localhost:10088
    }
```

@jckwart

hazelcast

Java KDC implementations

- 2 subprojects in the Apache Directory Project
  - Apache Kerby
  - ApacheDS
- can run embedded - testing simplified

Apache Kerby / Show me the Code

```
SimpleKdcServer kdc = new SimpleKdcServer();
kdc.setKdcHost("localhost");
kdc.setKdcRealm("TEST.REALM");
kdc.setKdcPort(10088);
kdc.init();

kdc.createPrincipal("jduke", "theduke");
kdc.start();
```

Maven

```
<dependency>
    <groupId>org.apache.kerby</groupId>
    <artifactId>kerb-simplekdc</artifactId>
    <version>2.0.0</version>
</dependency>
```

@jckwart

hazelcast

ApacheDS / Show me the Code

```
@CreateDS(name = "HazelcastDS",
        partitions = {
            // @CreatePartition ...
            },
            additionalInterceptors = { KeyDerivationInterceptor.class })
@CreateKdcServer(primaryRealm = "HAZELCAST.ORG",
        kdcPrincipal = "krbtgt/HAZELCAST.ORG@HAZELCAST.ORG",
        searchBaseDn = "dc=hazelcast,dc=org")
public static void startKdc() throws Exception {
    DirectoryService directoryService = DSAnnotationProcessor.getDirectoryService();
    KdcServer kdc = ServerAnnotationProcessor.getKdcServer(directoryService, 10088);
}
```

Maven

```
<dependency>
  <groupId>org.apache.directory.server</groupId>
  <artifactId>apacheds-all</artifactId>
  <version>2.0.0.AM25</version>
</dependency>
```

@jckwart

hazelcast

DEMO

github.com/kwart/demo-kerberos-in-java

@jckwart

hazelcast

Demo I

- start Java-based KDC (Kerby) in IDE
- capture network traffic in Wireshark (port 10088)
- use native (MIT) Kerberos commands

```
kinit jduke
klist
kvno gsstest/localhost
klist
```

- review Kerberos messages in WireShark:
  - AS_REQ, AS_REP, TGS_REQ, TGS_REP

JAAS authentication against KDC - `Krb5LoginModule`

- vendor specific implementation classes
  - `com.sun.security.auth.module.Krb5LoginModule`
  - `com.ibm.security.auth.module.Krb5LoginModule`

JAAS authentication against KDC - `Krb5LoginModule`

- vendor specific implementation classes
  - `com.sun.security.auth.module.Krb5LoginModule`
  - `com.ibm.security.auth.module.Krb5LoginModule`
- referenced from a JAAS configuration file

```
KerberosLogin {
com.sun.security.auth.module.Krb5LoginModule required
debug=true;
};
```

JAAS authentication against KDC - `Krb5LoginModule`

- vendor specific implementation classes
  - `com.sun.security.auth.module.Krb5LoginModule`
  - `com.ibm.security.auth.module.Krb5LoginModule`
- referenced from a JAAS configuration file

```
KerberosLogin {
com.sun.security.auth.module.Krb5LoginModule required
debug=true;
};
```

- configuration referenced by JAAS API call

```
LoginContext lc = new LoginContext("KerberosLogin",
new NamePasswordCbHandler("jduke@TEST.REALM", "theduke".toCharArray()));
lc.login();
Subject subj = lc.getSubject();
```

@jckwart

hazelcast

System properties for Kerberos authentication

- path to `krb5.conf` file
  - `java.security.krb5.conf`
- realm and KDC host defaults
  - `java.security.krb5.realm`
  - `java.security.krb5.kdc` *(doesn't support custom ports)*
- debug output
  - `sun.security.krb5.debug=true`
  - `com.ibm.security.krb5.Krb5Debug=all`

GSS-API (RFC 2743)

- Generic Security Service Application Program Interface
- provides uniform access to security services
- doesn't provide security itself
  - delegates to an underlying security mechanism

GSS-API (RFC 2743)

- Generic Security Service Application Program Interface
- provides uniform access to security services
- doesn't provide security itself
  - delegates to an underlying security mechanism

Relationship to Kerberos (RFC 4121)

- Kerberos is dominant GSS-API mechanism
- GSS-API provides standardized API for Kerberos

GSS-API (RFC 2743)

- Generic Security Service Application Program Interface
- provides uniform access to security services
- doesn't provide security itself
  - delegates to an underlying security mechanism

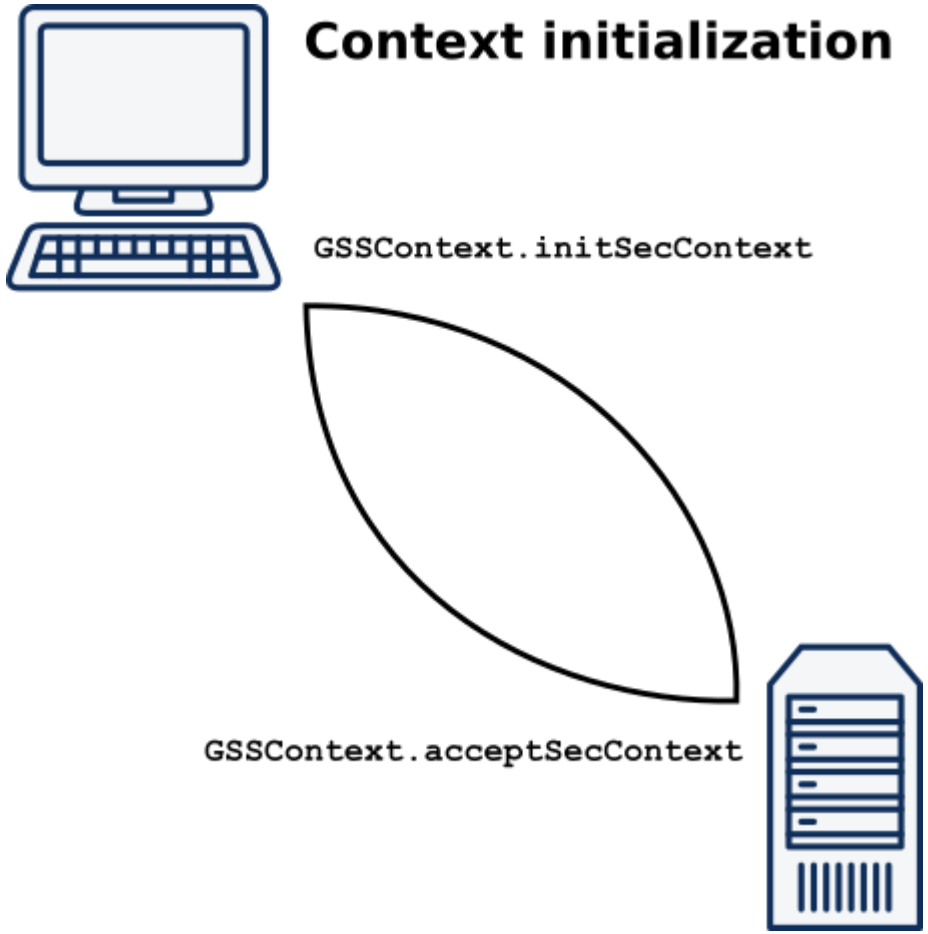
Relationship to Kerberos (RFC 4121)

- Kerberos is dominant GSS-API mechanism
- GSS-API provides standardized API for Kerberos

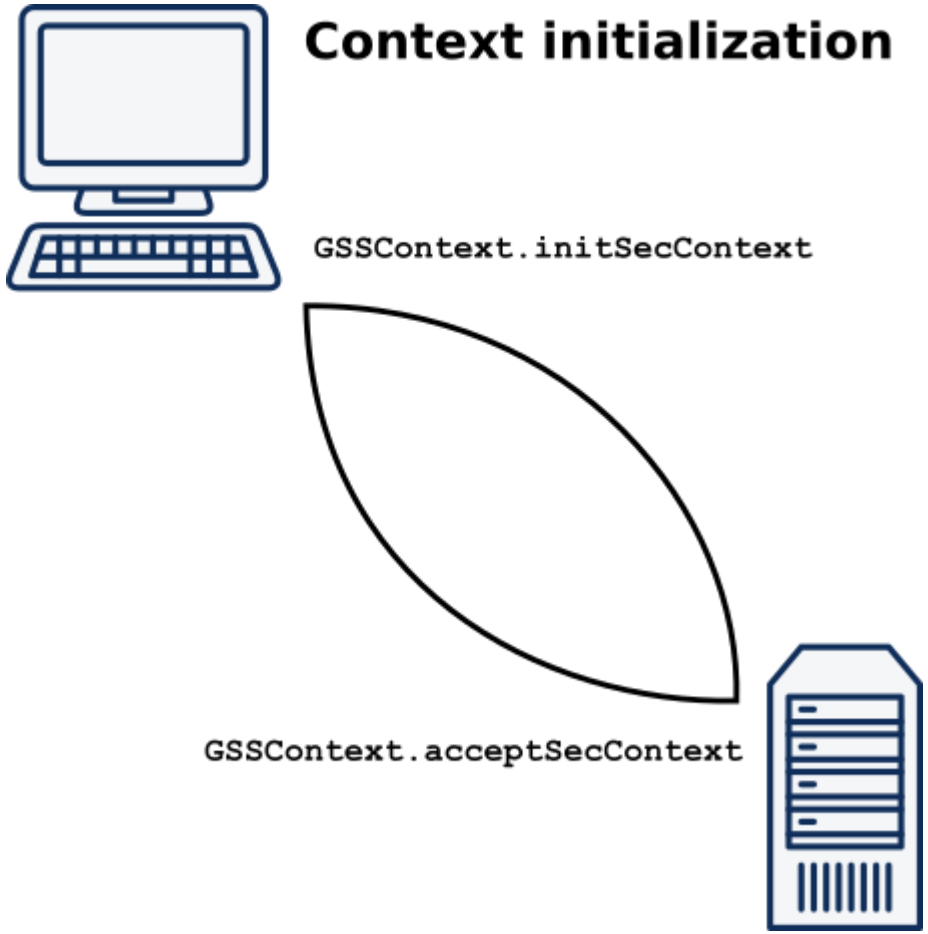
Not enough APIs? Let's SPNEGO (RFC 4178)

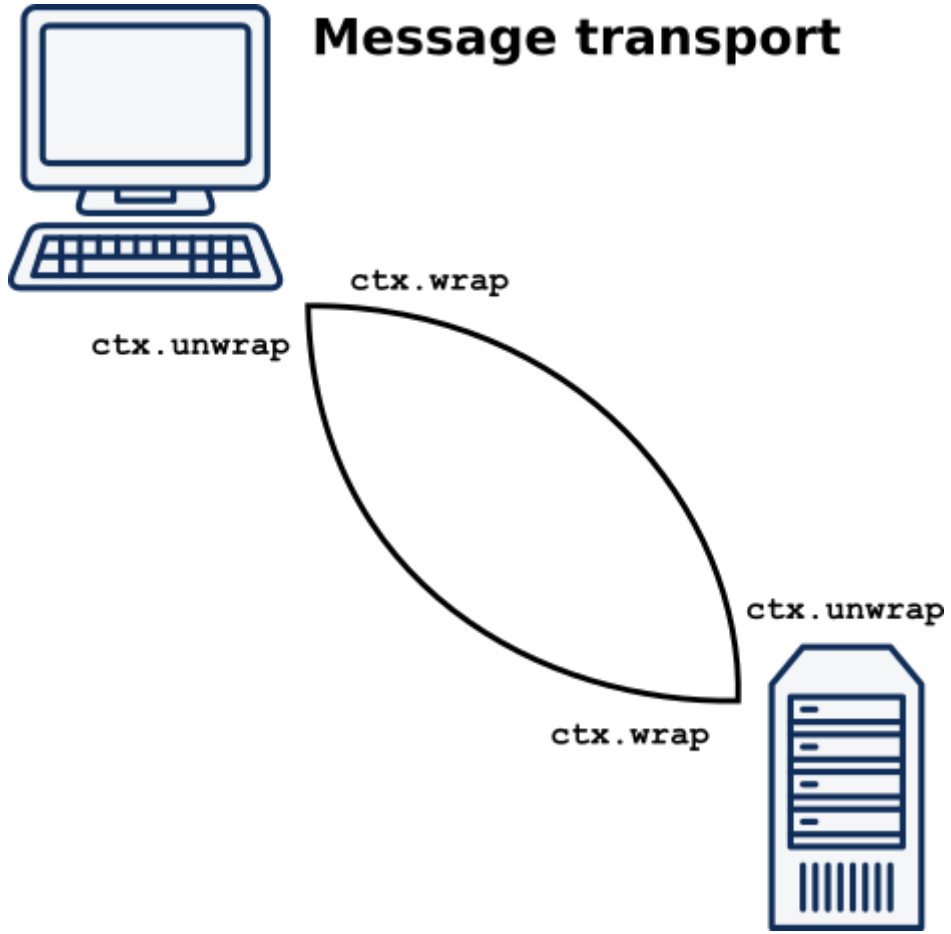
- Simple and Protected GSSAPI Negotiation Mechanism
- pseudo-mechanism - allows negotiation of a real mechanism

GSS-API overview



GSS-API overview





GSS-API in Java (RFC 2853)

- package `org.ietf.jgss`
  - `GSSManager`
  - `GSSContext`
  - `GSSName`
  - `GSSCredential`
  - `MessageProp`
  - `Oid`
- retrieves Kerberos credentials from the JAAS `Subject`
- can also proceed with JAAS authentication by itself
  - login config entries: `com.sun.security.jgss.krb5.initiate`, `com.sun.security.jgss.krb5.accept`
  - system property: `javax.security.auth.useSubjectCredsOnly=false`

Kerberos in Java on Windows

- you can reuse authentication in Active Directory domain
  - allow access to session key in Local Security Authority (LSA)

```
Reg Add
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters
/v AllowTGTSessionKey /t REG_DWORD /d 1
```

  - PowerShell workaround for missing native `kinit`

```
Import-Module ActiveDirectory
```

- create playground from a simple template in MS Azure
  - https://github.com/maxskunkworks/TLG/tree/master/tlg-base-config_3-vm

@jckwart

hazelcast

Sample: Kerberos in JAAS enabled client-server application

Client

- Authenticate to KDC through JAAS `Krb5LoginModule` login
- request service ticket by initializating `GSSContext`
  - `initSecContext` returns GSS-API token which includes the service ticket
- send the token to server as client credentials

Sample: Kerberos in JAAS enabled client-server application

Client

- Authenticate to KDC through JAAS `Krb5LoginModule` login
- request service ticket by initializating `GSSContext`
  - `initSecContext` returns GSS-API token which includes the service ticket
- send the token to server as client credentials

Server

- Use `Krb5LoginModule` to load keytab file into Subject credentials
- Receive GSS-API token from client
- use custom login module and try to accept the GSS-API token in a new `GSSContext`

Sample: Kerberos in JAAS enabled client-server application

Hazelcast IMDG

- Apache Licensed In-Memory Data Grid
- main use-case - distributed cache which scales
- not limited by heap or single JVM
- Goals: performance and simplicity

Sample: Kerberos in JAAS enabled client-server application

Hazelcast IMDG

- Apache Licensed In-Memory Data Grid
- main use-case - distributed cache which scales
- not limited by heap or single JVM
- Goals: performance and simplicity

- Security is an Enterprise feature
  - **doesn't support Kerberos** (yet)
  - supports JAAS login modules

@jckwart

hazelcast

Sample: Kerberos in JAAS enabled client-server application

Hazelcast IMDG

- Apache Licensed In-Memory Data Grid
- main use-case - distributed cache which scales
- not limited by heap or single JVM
- Goals: performance and simplicity

- Security is an Enterprise feature
  - **doesn't support Kerberos** (yet)
  - supports JAAS login modules

Code sample

```
HazelcastInstance hz = HazelcastClient.newHazelcastClient();

Map<Integer, User> userCache = hz.getMap("users");
User user = userCache.get(id);
if (user == null) {
    user = dbUtil.loadUser(id);
    userCache.put(id, user);
}
```

@jckwart

hazelcast

Sample: Client side code

Sample: Client side code

JAAS login config (automatic GSS-API login)

```
com.sun.security.jgss.initiate  {
  com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
};
```

@jckwart

hazelcast

Sample: Client side code

JAAS login config (automatic GSS-API login)

```
com.sun.security.jgss.initiate  {
  com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
};
```

Retrieve service token by initializeing GSSContext

```
GSSManager manager = GSSManager.getInstance();
GSSName servicePrincipalName = manager.createName(
      "hazelcast/test.hazelcast.org@HAZELCAST.ORG", null);
GSSContext gssContext = manager.createContext(
      servicePrincipalName, KRB5_OID, null, GSSContext.DEFAULT_LIFETIME);
gssContext.requestMutualAuth(false);
byte[] token = gssContext.initSecContext(new byte[0], 0, 0);
```

@jckwart

hazelcast

Sample: Client side code

JAAS login config (automatic GSS-API login)

```
com.sun.security.jgss.initiate  {
  com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
};
```

Retrieve service token by initializeing GSSContext

```
GSSManager manager = GSSManager.getInstance();
GSSName servicePrincipalName = manager.createName(
      "hazelcast/test.hazelcast.org@HAZELCAST.ORG", null);
GSSContext gssContext = manager.createContext(
      servicePrincipalName, KRB5_OID, null, GSSContext.DEFAULT_LIFETIME);
gssContext.requestMutualAuth(false);
byte[] token = gssContext.initSecContext(new byte[0], 0, 0);
```

Use the token for authentication

```
ClientConfig clientConfig = new ClientConfig();
clientConfig.getSecurityConfig().setCredentials(new SimpleTokenCredentials(token));
clientConfig.getNetworkConfig().addAddress("test.hazelcast.org");
HazelcastInstance hz = HazelcastClient.newHazelcastClient(clientConfig);
```

@jckwart

hazelcast

Sample: Server side code - GSSApiLoginModule

```
public boolean login() throws LoginException {
  // try-catch blocks removed for readability
  CredentialsCallback cc = new CredentialsCallback();
  callbackHandler.handle(new Callback[] { cc });
  TokenCredentials creds = (TokenCredentials) cc.getCredentials();

  byte[] token = creds.getToken();
  GSSContext gssContext = GSSManager.getInstance().createContext((GSSCredential) null);
  token = gssContext.acceptSecContext(token, 0, token.length);

  if (!gssContext.isEstablished()) {
      throw new FailedLoginException("Multi-step negotiation is not supported by this login module");
  }
  name = gssContext.getSrcName().toString();
  return true;
}
```

@jckwart

hazelcast

DEMO

github.com/kwart/demo-kerberos-in-java

@jckwart

hazelcast

Demo II

- Using `Krb5LoginModule` - initiator & acceptor
- Custom Kerberos authentication in Hazelcast
- Java GSS-API Client/Server

TLS and Kerberos (JSSE vs Java GSS-API)

- Client-Server authentication
  - TLS authenticates the server by default
  - Kerberos authenticates the client by default
- encryption and data integrity
- data structures described in ASN.1

TLS and Kerberos (JSSE vs Java GSS-API)

- Client-Server authentication
  - TLS authenticates the server by default
  - Kerberos authenticates the client by default
- encryption and data integrity
- data structures described in ASN.1

Kerberos

- Single-sign-On
- Credentials delegation
  - server impersonates client when accessing other services
- Token-based communication
  - Selective Encryption
- Transport: UDP, TCP

TLS

- public key cryptography
- socket-based API
- authentication doesn't depend on central server availability
- doesn't have strict time requirements (clock synchronization)
- Transport: TCP

Tools for Debugging Kerberos

- network analyzer: WireShark / tcpdump
- ASN.1 parser:
  - `dumpasn1` tool
  - `org.apache.kerby.asn1.Asn1`
- system properties
  - `sun.security.krb5.debug=true`
  - `sun.security.jgss.debug=true`
  - `sun.security.spnego.debug=true`
  - `com.ibm.security.krb5.Krb5Debug=all`
  - `com.ibm.security.jgss.debug=all`
  - `java.security.debug=gsslogincfg`
- `Krb5LoginModule` option `debug=true`

@jckwart

hazelcast

Resources

- https://github.com/kwart/demo-kerberos-in-java
- https://docs.oracle.com/javase/8/docs/technotes/guides/security/index.html
- https://web.mit.edu/kerberos/krb5-1.17/doc/
- RFC-4120 The Kerberos Network Authentication Service (V5)
- RFC-2743 GSS-API
- RFC-2853 GSS-API: Java Bindings
- https://directory.apache.org/kerby/
- https://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface

Thank you

github.com/kwart
twitter.com/jckwart
javlog.cacek.cz

@jckwart

hazelcast