GridGain

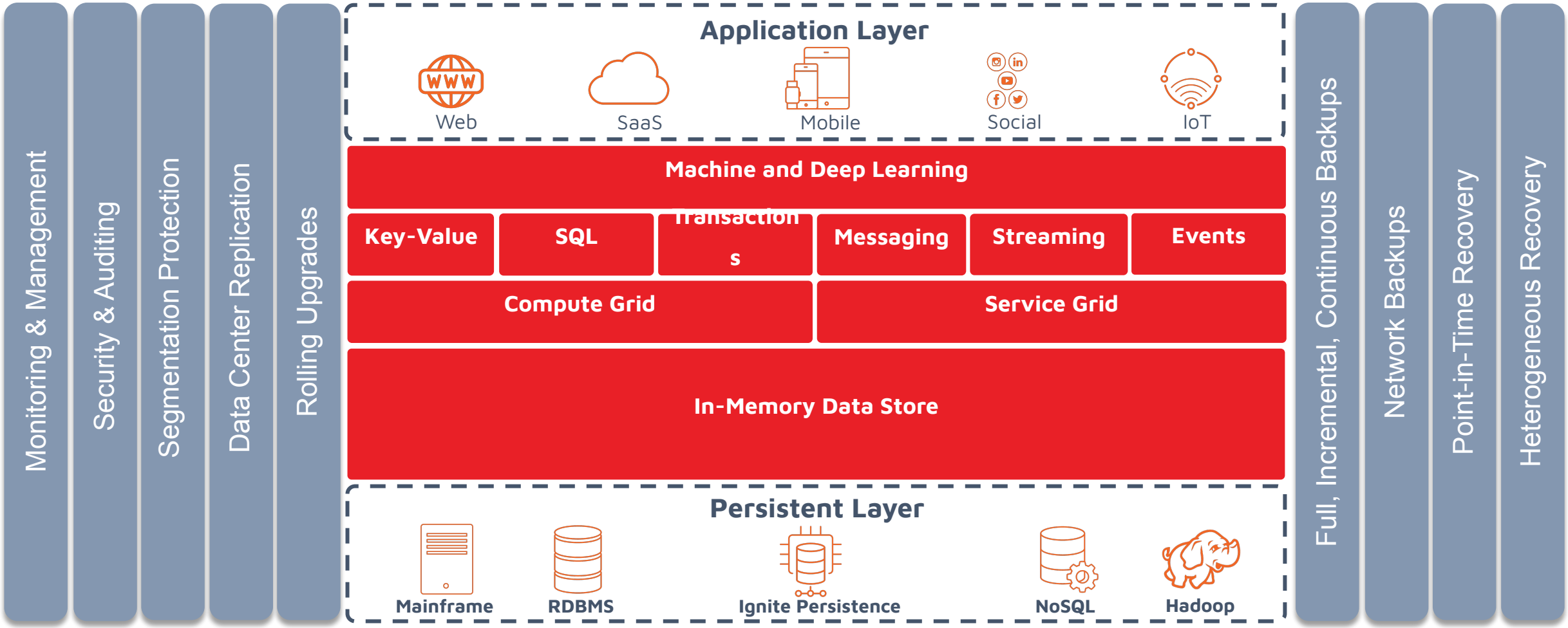
Apache Ignite and GridGain Security Guide

Alexandrov Andrei

About me

Name: Andrei Alexandrov
Company: GridGain since 2017
Team: Customer Solutions
Email: aealexsandrov@gmail.com

GridGain In-Memory Computing Platform



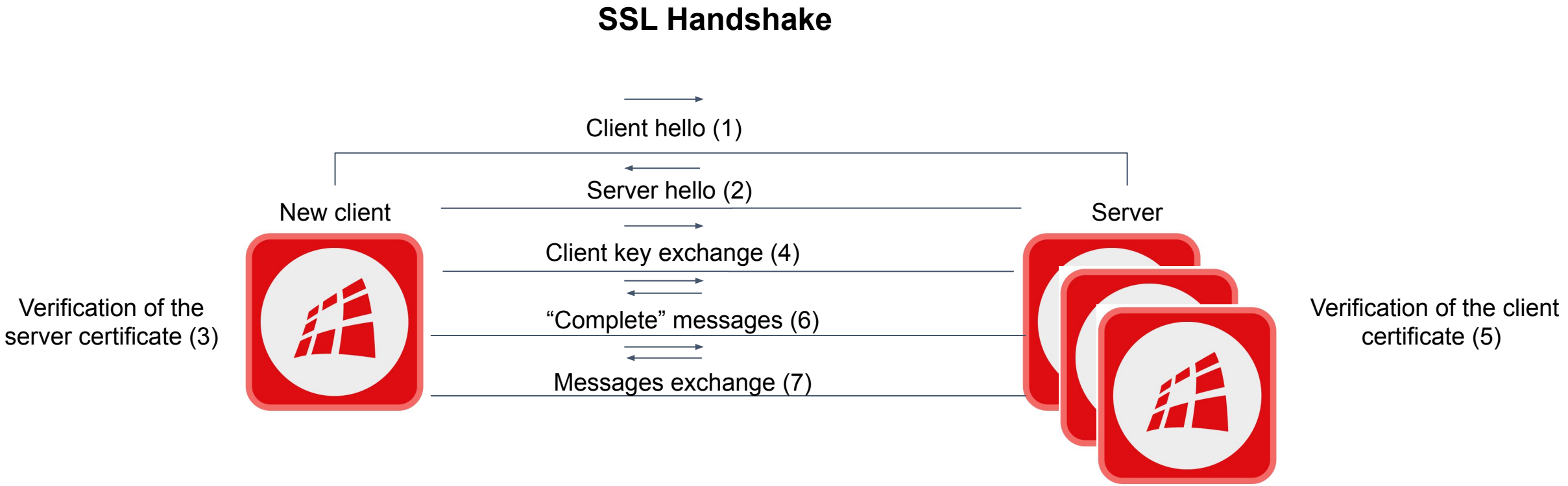
Presentation Plan

1. Apache Ignite and Gridgain security overview
2. SSL/TLS for Apache Ignite and GridGain
   a. How SSL Handshake Works
   b. How SSL/TLS Certificates Work
   c. SSL/TLS for Cluster Nodes and Clients
   d. SSL/TLS for SQL Thin Clients (ODBC/JDBC)
   e. SSL/TLS for Other Thin Clients
   f. Configuration of the HTTPS
   g. SSL/TLS for Visor CMD
   h. SSL/TLS for Standalone Web Console

Presentation Plan

3. Apache Ignite advanced security
4. GridGain security plugin
   a. GridGain security plugin overview
   b. GridGain security plugin - user permissions
   c. GridGain Username/Password authenticator
   d. GridGain Java Authentication and Authorization Service authenticator
5. Audit via Events
   a. GridGain security events for authentication and authorization

(2.a) How SSL Handshake Works



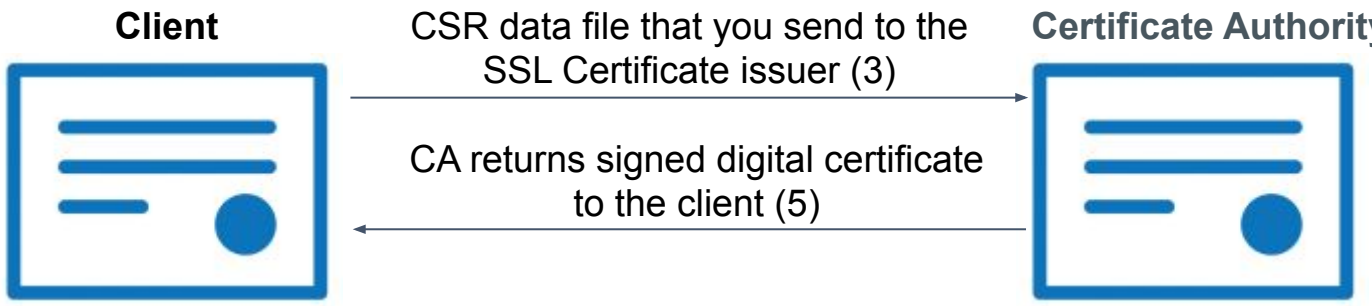
(1) SSL/TLS version, CipherSuites supported by the client, compression methods supported by the client.
(2) CipherSuite chosen by the server from the list provided by the client, the session ID, server digital certificate, client certificate request.
(3) Digital signature, certificate chain, expiry and activation dates and the validity period of the server certificate checks.
(4) Secret key will be sent to the server. It will be signed by the server public key. Client sends its own certificate if client authentication is required.
(5) Digital signature, certificate chain, expiry and activation dates, and the validity period of the client certificate checks.
(6) Client and Server send each other the "finish" message encrypted with the secret key signaling that the client handshake is complete.
(7) Server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

(2.b) How SSL/TLS Certificates Work

Private and Public key pair for the client were generated (1)

Certificate Signing Request was created (2)

You should add the digital certificate to some store (**keystore**). Also, you should add CA certificates that can be used for validation of remote certificates (**truststore**) (6)



CA uses the CSR data file to create a **digital identity certificate** to match your private key without compromising the key itself (4)

(1) Public keys can be used widely. Private keys must be known only to the owner.
(2) CSR contains the public key for which the certificate should be issued, identifying the information (such as a domain name) and integrity protection (e.g., a digital signature).
(3) You can use some trusted providers or generate self-signed certificates.
(4) (5) (6) Self-signed certificates, CA and stores can be created for free using a wide variety of tools like OpenSSL.

(2.c) SSL/TLS for Cluster Nodes and Clients

Configure **sslContextFactory** of **IgniteConfiguration**:

```
<property name="sslContextFactory">
        <bean class="org.apache.ignite.ssl.SslContextFactory">
          <property name="keyStoreFilePath" value="node.jks" />
          <property name="keyStorePassword" value="password" />
          <property name="trustStoreFilePath" value="trust.jks" />
          <property name="trustStorePassword" value="password" />
        </bean>
</property>
```

(2.d) SSL/TLS for SQL Thin Clients (ODBC/JDBC)

Configure **clientConnectorConfiguration** of **IgniteConfiguration**:

```
<property name="clientConnectorConfiguration">
  <bean
class="org.apache.ignite.configuration.ClientConnectorConfiguration">
      <property name="host" value="127.0.0.1" />
      <property name="port" value="10800"/>
      <property name="portRange" value="10"/>
      <property name="sslEnabled" value="true" />
      <property name="sslClientAuth" value="true" />
  </bean>
</property>
```

(2.d) JDBC Example

```
// Register JDBC driver.
Class.forName("org.apache.ignite.IgniteJdbcThinDriver");
// Open the JDBC connection.
try (Connection conn =
DriverManager.getConnection("jdbc:ignite:thin://127.0.0.1:10800?" +
    "sslMode=require&" +
    "sslClientCertificateKeyStoreUrl=client.jks&" +
    "sslClientCertificateKeyStorePassword=password&" +
    "sslTrustCertificateKeyStoreUrl=trust.jks&" +
    "sslTrustCertificateKeyStorePassword=password")) {
    conn.createStatement().executeQuery("select 1");}

catch (SQLException e) { e.printStackTrace(); }
```

(2.d) ODBC Example

Connection string:

DRIVER={Apache Ignite};ADDRESS=127.0.0.1:10800;SCHEMA=PUBLIC;SSL_MODE=require;SSL_KEY_FILE=client.pem;SSL_CERT_FILE=client.pem;SSL_CA_FILE=ca_odbc.pem;

- **ssl_cert_file** - path to the file containing a PEM-encoded SSL/TLS certificate.
- **ssl_key_file** - path to the file containing a PEM-encoded private key.
- **ssl_ca_file** - Specifies the name of the file containing the SSL server certificate authority (CA)

!!! SSL_KEY_FILE and SSL_CERT_FILE may point to the same file.

(2.e) SSL/TLS for Other Thin Clients

Java Thin Client	https://www.gridgain.com/docs/latest/developers-guide/thin-clients/java-thin-client#ssltls
.Net Thin Client	https://www.gridgain.com/docs/latest/developers-guide/thin-clients/dotnet-thin-client#ssltls
C++ Thin Client	https://www.gridgain.com/docs/latest/developers-guide/thin-clients/cpp-thin-client#ssltls
Python Thin Client	https://www.gridgain.com/docs/latest/developers-guide/thin-clients/python-thin-client#ssltls
PHP Thin Client	https://www.gridgain.com/docs/latest/developers-guide/thin-clients/php-thin-client#ssltls
Node.js Thin Client	https://www.gridgain.com/docs/latest/developers-guide/thin-clients/nodejs-thin-client#ssltls

(2.f) Configuration of the HTTPS

Add **ignite-rest-http** dependency

```
<dependency>
  <groupId>org.gridgain</groupId>
  <artifactId>ignite-rest-http</artifactId>
  <version>${ignite.version}</version>
</dependency>
```

Configure **connectorConfiguration** of **IgniteConfiguration**:

```
<property name="connectorConfiguration">
  <bean class="org.apache.ignite.configuration.ConnectorConfiguration">
    <property name="sslEnabled" value="true" />
    <property name="sslClientAuth" value="true" />
    <property name="jettyPath" value="jetty.xml" />
  </bean>
</property>
```

(2.g) SSL/TLS for Visor CMD

To start Visor CMD you need to perform the following steps:

- Set the **IGNITE_HOME** variable to folder with your GridGain binaries
- Copy your **client.xml** with configured sslContextFactory to **IGNITE_HOME/config** folder. It will help you to see this file in the command line interface
- Start **$IGNITE_HOME/bin/ignitevisorcmd.sh**
- Type “open”
- Choose your configuration file from the list
- Visor CMD will be started

(2.h) SSL/TLS for Standalone Web Console

To start Web Console you need to perform the following steps:

- Download the archive and unzip somewhere
- Create **application.properties** file in the root of the archive.
- Add required properties (next slide)
- Run **web-console.sh** script.
- Go to web console UI and download the web agent.
- Copy your security token from Web Console
- Unzip the web agent archive and run it (next slide)

(2.h) SSL/TLS for Stand-alone Web Console

Properties:

server.port=443

server.ssl.protocol=TLS

server.ssl.key-store-type=JKS

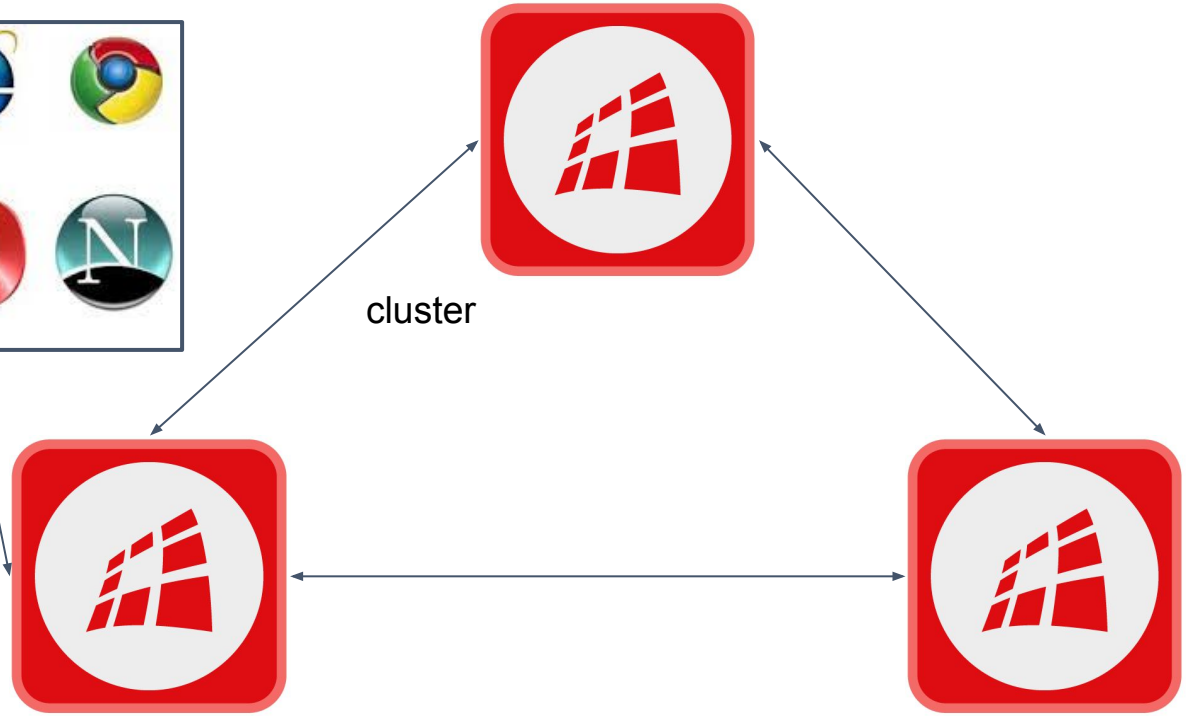
server.ssl.key-store=node.jks

server.ssl.key-store-password=password

server.ssl.trust-store-type=JKS

server.ssl.trust-store=trust.jks

server.ssl.trust-store-password=password



Run web agent command:

start ignite-web-console-agent.bat --node-key-store node.jks --node-key-store-password password --node-trust-store trust.jks --node-trust-store-password password --server-key-store wc.jks --server-key-store-password password --server-trust-store trust.jks --server-trust-store-password password -t 9fae7710-3a6c-475f-84f6-2c760eed67a6 -s https://localhost:443 -n https://localhost:8443

SSL/TLS Configuration Demo

Goals:

- Start server and client nodes with configured SSL
- Start Visor CMD with configured SSL
- Start Web Console with configured SSL

Where We Are?

1. Apache Ignite and Gridgain security overview
2. SSL/TLS for Apache Ignite and GridGain
3. Apache Ignite Advanced Security
4. GridGain Security Plugin
5. Audit via Events

(3) Apache Ignite Advanced Security

Note that this is just username/password authentication for thin clients!

To configure:

- Set the **authenticationEnabled** property of **IgniteConfiguration** to **true**
- Set the **persistenceEnabled** property of **defaultDataRegionConfiguration** to **true**

After that a user with the name **ignite** and password **ignite** will be created.

You can manage your users using the following commands:

- CREATE USER userName WITH PASSWORD 'password';
- ALTER USER userName WITH PASSWORD 'newPassword';
- DROP USER userName;

Note that **userName** will be created in **UPPER_CASE** by default but you can use "**userName**" and it will be created in **CAMEL_CASE**.

(3) Apache Ignite Advanced Security

Limitations:

- No authentication for cluster nodes
- No authorization
- Persistence is required

(4) GridGain Security Plugin Overview

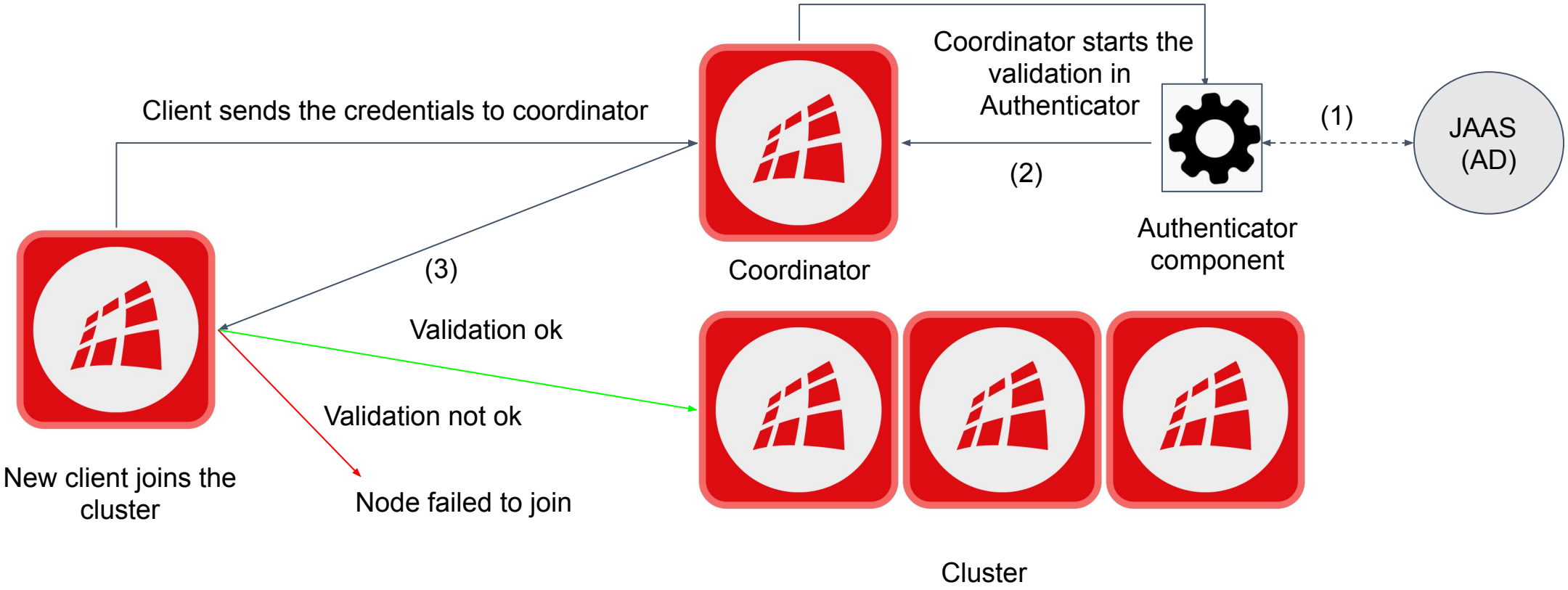
GridGain provides the following:

- Username/Password authenticator
- Java Authentication and Authorization Service authenticator
- Support for customized security authenticator that will override default implementation

Where We Are?

1. Apache Ignite and Gridgain security overview
2. SSL/TLS for Apache Ignite and GridGain
3. Apache Ignite Advanced Security
4. GridGain Security Plugin
5. Audit via Events

(4.a) GridGain Security Plugin Overview



(1) Authenticator validates the credentials. In case of JAAS, it will go to the specified AD and validate the user.
(2) After validating the user permissions, authenticator returns the unique SecuritySubject for the new client, which can be used for identifying this new client.
(3) Coordinator send the result of validation to the client. After that client can continue the join process or fail.

(4.b) GridGain Security Plugin - User Permissions

Cache permissions:

- CACHE_READ - cache read operations
- CACHE_PUT - cache put operations
- CACHE_REMOVE - cache remove operations

Task permissions:

- TASK_EXECUTE - task execution
- TASK_CANCEL - task cancellation

Service permissions:

- SERVICE_DEPLOY - service deployment
- SERVICE_INVOKE - service invocation
- SERVICE_CANCEL - service cancellation

System permissions:

- JOIN_AS_SERVER - node joining the cluster as server
- EVENTS_ENABLE - enabling events in runtime
- EVENTS_DISABLE - disabling events in runtime
- ADMIN_OPS - operations executed from the Web Console
- ADMIN_VIEW - viewing cluster statistics (metrics, graphs, cache sizes, etc.) in the Web Console
- ADMIN_QUERY - executing SQL queries from the Web Console
- ADMIN_CACHE - cache operations executed from the Web Console (data loading, manual rebalancing, etc.)
- CACHE_CREATE - creating new caches (including ones specified in the node configuration)
- CACHE_DESTROY - destroying existing caches

(4.c) GridGain Username/Password Authenticator

Server configuration should contain all expected credentials because they will be used by the Authenticator for validation of remote nodes and clients:

```
<!-- Server node credentials. -->
<bean id="server.cred" class="org.apache.ignite.plugin.security.SecurityCredentials">
  <constructor-arg value="server" />
  <constructor-arg value="password" />
</bean>

<!-- Client node credentials. -->
<bean id="client.cred" class="org.apache.ignite.plugin.security.SecurityCredentials">
  <constructor-arg value="client" />
  <constructor-arg value="password" />
</bean>
```

(4.c) GridGain Username/Password Authenticator

You should add **PasscodeAuthenticator** to the **authenticator** property of **GridGainConfiguration:**

```
<property name="authenticator">
  <bean class="org.gridgain.grid.security.passcode.PasscodeAuthenticator">
    <property name="aclProvider">
      <bean class="org.gridgain.grid.security.passcode.AuthenticationAclBasicProvider">
        <constructor-arg>
          <map>
            <!-- Allow all operations on server nodes. -->
            <entry key-ref="server.cred" value="{defaultAllow:true}" />
            <!-- Allow only cache reads on client nodes. -->
            <entry key-ref="client.cred" value="{defaultAllow:false,{cache:'*',permissions:['CACHE_READ']}}" />
          </map>
        </constructor-arg>
      </bean>
    </property>
  </bean>
</property>
```

(4.c) GridGain Username/Password Authenticator

You should add **SecurityCredentialsBasicProvider** to the **securityCredentialsProvider** property of **GridGainConfiguration** to provide the credential that should be used:

```
<property name="securityCredentialsProvider">
  <bean
class="org.apache.ignite.plugin.security.SecurityCredentialsBasicProvider">
    <!-- Specify credentials for the current node -->
    <constructor-arg ref="server.cred" />
  </bean>
</property>
```

It should be done on every node.

GridGain Username/Password Authenticator Demo

Goals:

- Start server and client nodes with configured SSL and Security
- Start Web Console with configured SSL and Security

(4.d) GridGain JAAS Authenticator

Every cluster node should have correct JAAS configuration file like the following:

```
GridJaasLoginContext {
    com.sun.security.auth.module.LdapLoginModule SUFFICIENT
    userProvider="ldap://127.0.0.1:389/CN=Users,DC=example,DC=com"
    authIdentity="{USERNAME}@EXAMPLE.COM"
    userFilter="(&(sAMAccountName={USERNAME})(memberOf=CN=superuser,CN=Users,DC=example,DC=com))"
    useSSL=false
    authzIdentity="superuser";

    com.sun.security.auth.module.LdapLoginModule SUFFICIENT
    userProvider="ldap://127.0.0.1:389/CN=Users,DC=example,DC=com"
    authIdentity="{USERNAME}@EXAMPLE.COM"
    userFilter="(&(sAMAccountName={USERNAME})(memberOf=CN=client,CN=Users,DC=example,DC=com))"
    useSSL=false
    authzIdentity="client";
};
```

After the module class name, you can see the special flag REQUIRED. Here are the flags that you can use:

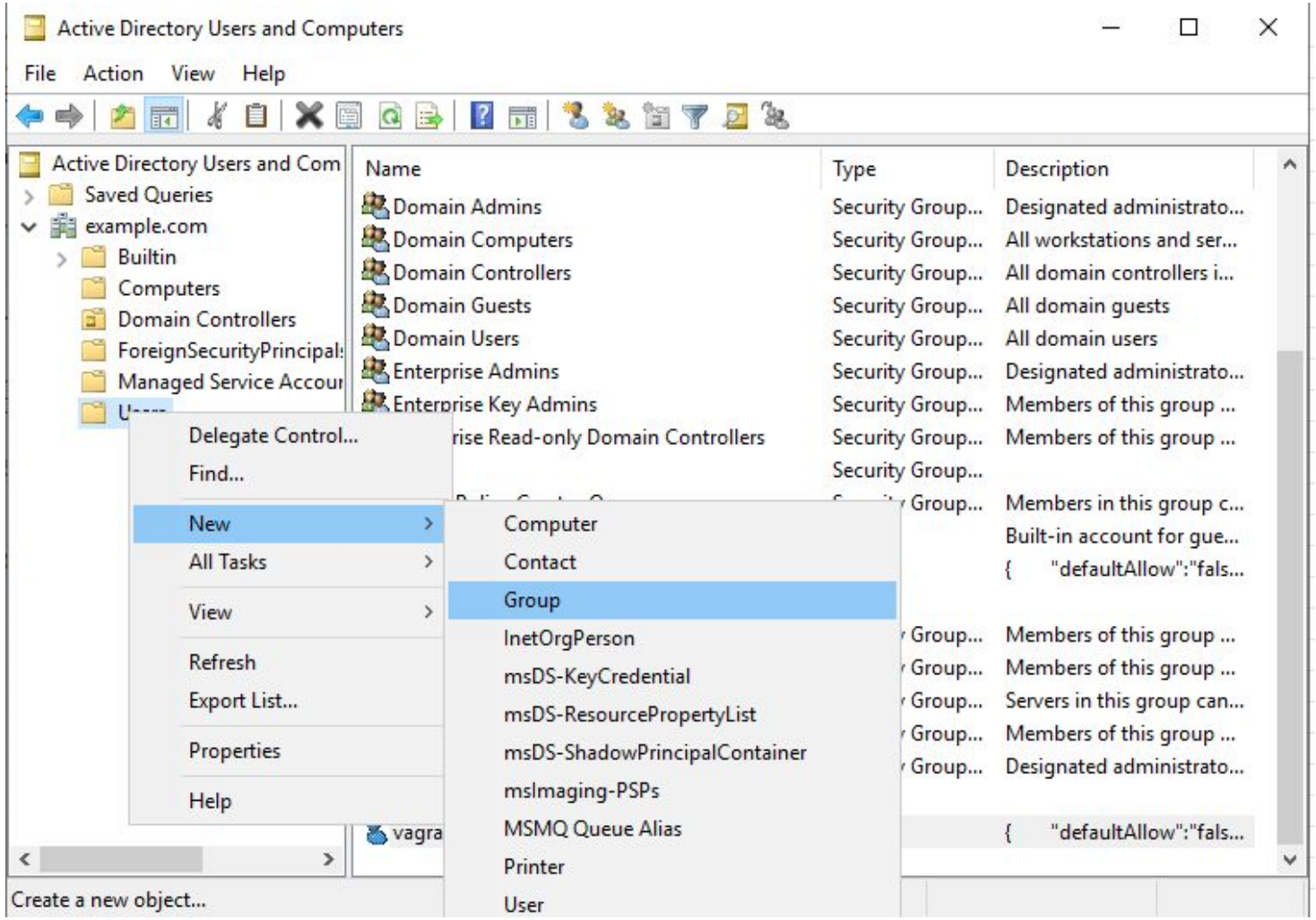
- **Required** - The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.
- **Requisite** - The LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).
- **Sufficient** - The LoginModule is not required to succeed. If it fails, authentication continues down the LoginModule list.
- **Optional** - The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

(4.d) GridGain JAAS Authenticator

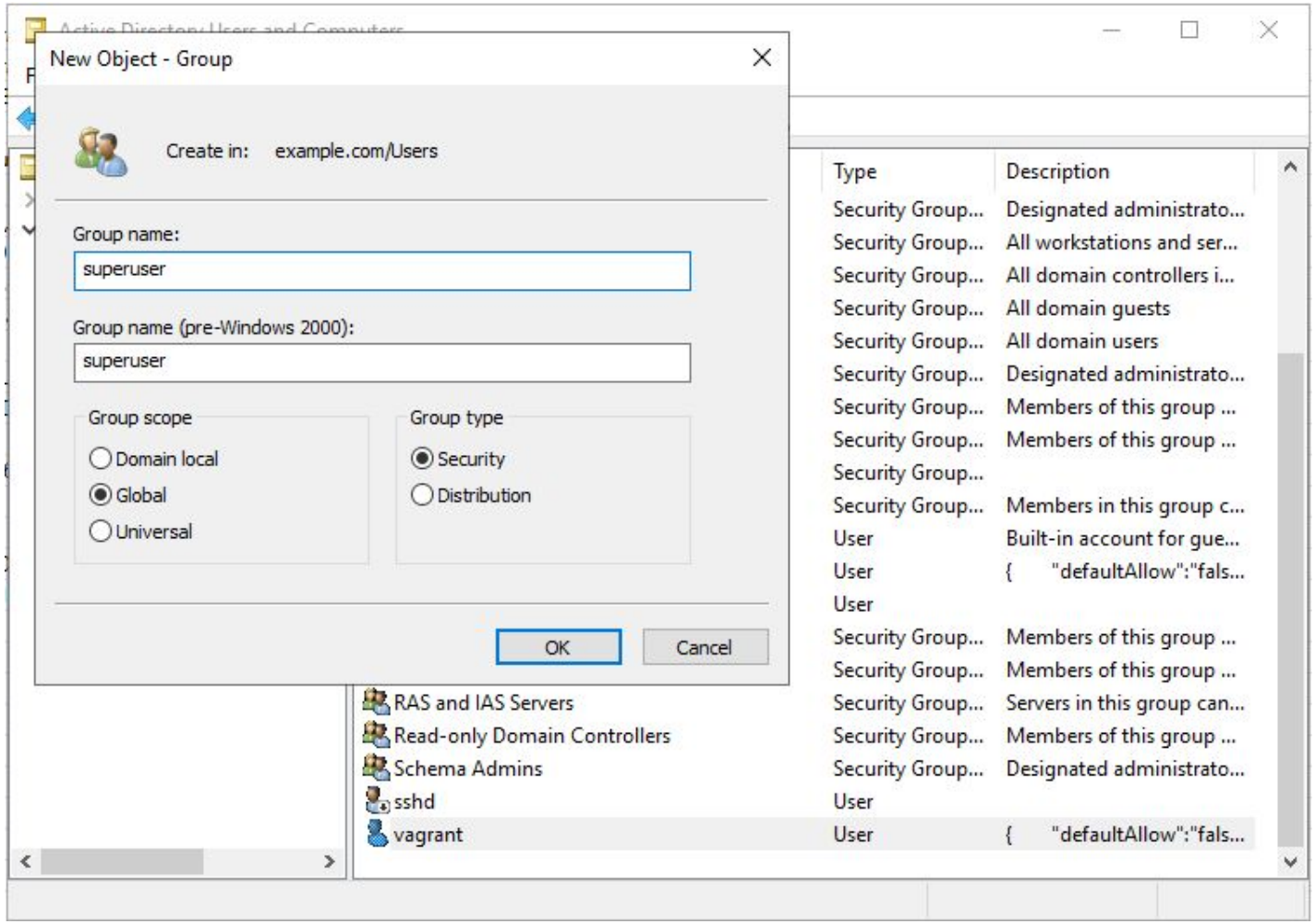
You should add **JaasAuthenticator** to the **authenticator** property of **GridGainConfiguration:**

```
<property name="authenticator">
    <bean class="org.gridgain.grid.security.jaas.JaasAuthenticator">
      <property name="permissionsProvider">
        <bean class="org.gridgain.grid.security.jaas.JaasBasicPermissionsProvider">
       <constructor-arg>
            <map>
              <entry key="superuser">
                <bean class="org.apache.ignite.plugin.security.SecurityBasicPermissionSet">
                  <property name="defaultAllowAll" value="true"/>
                </bean>
              </entry>
            </map>
          </constructor-arg>
        </bean>
 </property>
```

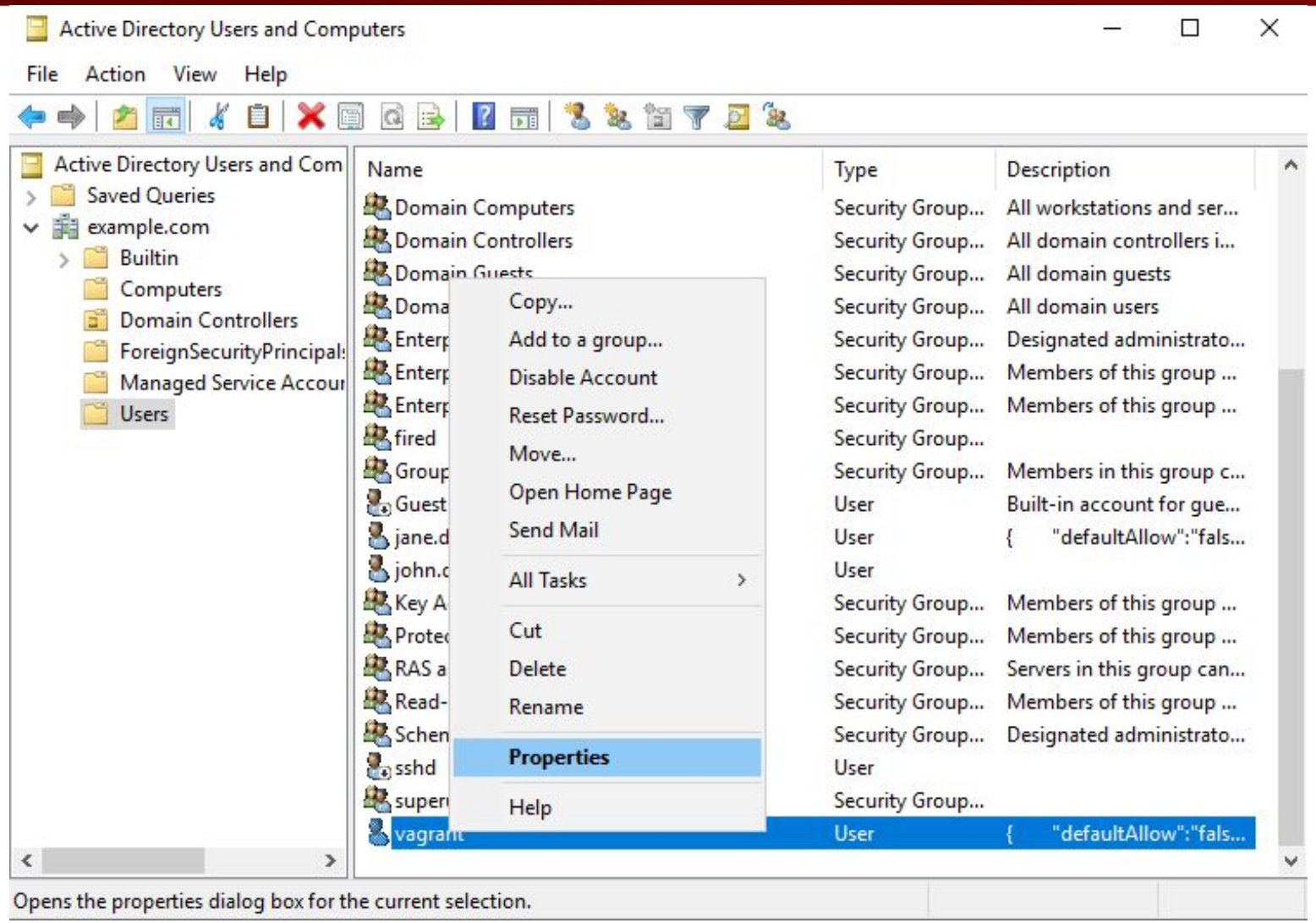
(4.d) GridGain JAAS Authenticator



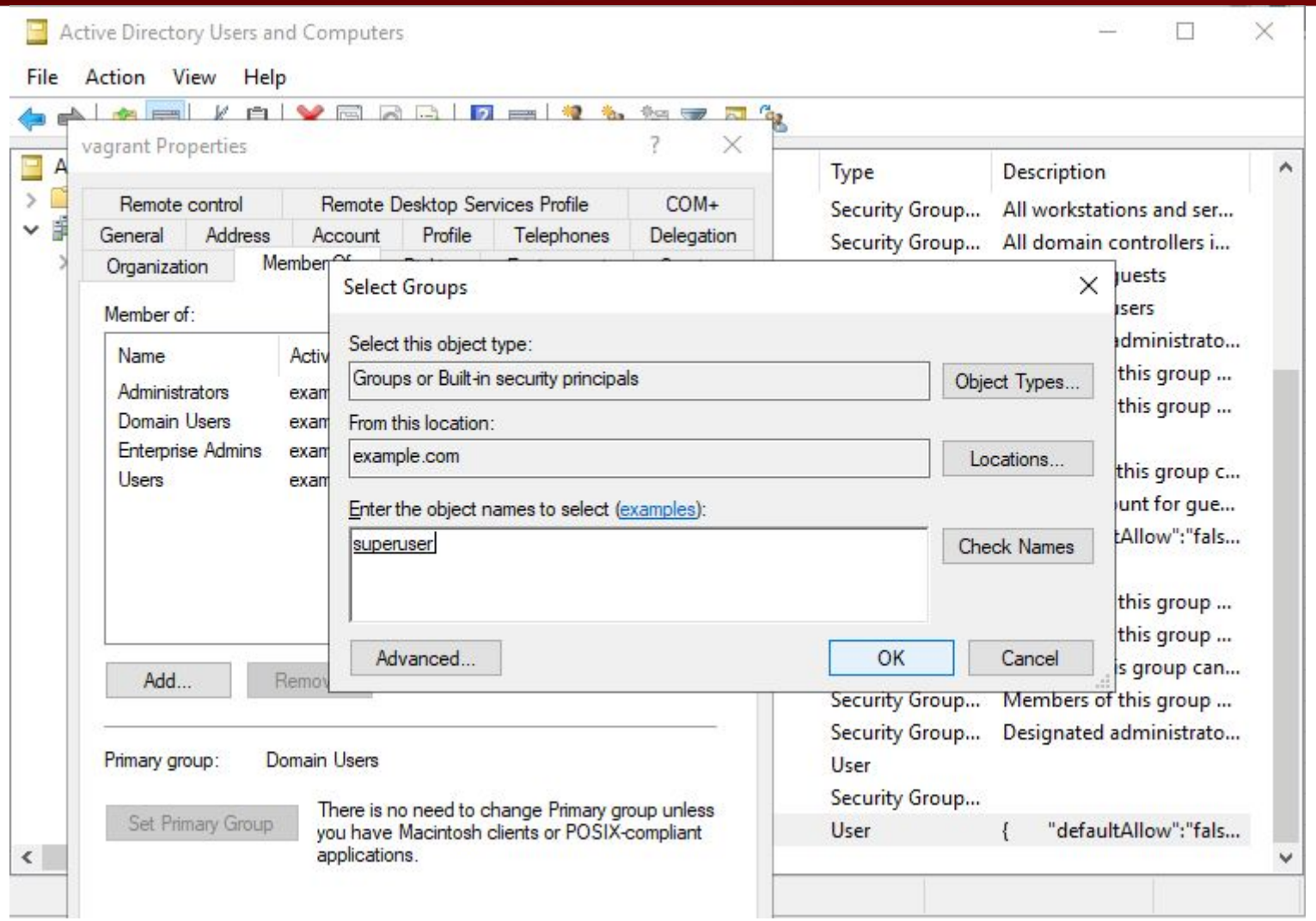
(4.d) GridGain JAAS Authenticator



(4.d) GridGain JAAS Authenticator



(4.d) GridGain JAAS Authenticator



GridGain JAAS authenticator live example

Goals:

- Start server and client nodes with configured JAAS Security
- Implement special callback to get the password from a java code

Where We Are?

1. Apache Ignite and Gridgain security overview
2. SSL/TLS for Apache Ignite and GridGain
3. Apache Ignite Advanced Security
4. GridGain Security Plugin
5. Audit via Events

(5) Audit via Events

```
@IgniteSpiMultipleInstancesSupport(true)
public class MgmtAuditEventStorageSpi extends IgniteSpiAdapter implements EventStorageSpi {
...
    public void record(Event evt) throws IgniteSpiException {
        if (evt.type() == EVT_MANAGEMENT_TASK_STARTED) {
            TaskEvent taskEvent = (TaskEvent) evt;
            SecuritySubject subj = taskEvent.subjectId() != null
                    ? getSpiContext().authenticatedSubject(taskEvent.subjectId())
                    : null;
            log.info("Management task started: [" +
                    "name=" + taskEvent.taskName() + ", " +
                    "eventNode=" + taskEvent.node() + ", " +
                    "timestamp=" + taskEvent.timestamp() + ", " +
                    "info=" + taskEvent.message() + ", " +
                    "subjectId=" + taskEvent.subjectId() + ", " +
                    "secureSubject=" + subj +
                    "]");}}}
```

Events documentation:

https://www.gridgain.com/docs/latest/developers-guide/events

(5.a) GridGain Security Events for Authentication and Authorization

GridGain provides additional security events:

- AuthorizationEvent
  - EVT_AUTHORIZATION_SUCCEEDED
  - EVT_AUTHORIZATION_FAILED
- AuthenticationEvent
  - EVT_AUTHENTICATION_FAILED
  - EVT_AUTHENTICATION_SUCCEEDED

Using the above listed events, you can map every operation to any user using a subject ID

QA

SSL/TLS article: https://www.gridgain.com/docs/tutorials/ssl_guide/ssl-guide

Demo's source code: https://github.com/aealeksandrov/security-webinar

GridGain documentation:
https://www.gridgain.com/docs/latest/getting-started/what-is-ignite

Ignite documentation: https://apacheignite.readme.io/docs/what-is-ignite

Custom authenticator example:
https://www.gridgain.com/docs/latest/administrators-guide/security/custom-authenticators