Hazelcast Security Overview

Agenda

- Communication Security
- Authentication
- Authorization / Client Permissions
- Persistence Encryption
- Auditlog
- Secrets Strength Validation
- Pluggable Security

Communication Security - Background

For attackers, it's easy to capture unprotected network communication.

- Hazelcast members support different application layer protocols - all TCP-based
  - Hazelcast Member Protocol (`HZC`)
  - Hazelcast Open Binary Client Protocol (`CP2`)
  - HTTP REST
  - memcached protocol (disabled by default)

Communication Security - Background

For attackers, it's easy to capture unprotected network communication.

- Hazelcast members support different application layer protocols - all TCP-based
  - Hazelcast Member Protocol (`HZC`)
  - Hazelcast Open Binary Client Protocol (`CP2`)
  - HTTP REST
  - memcached protocol (disabled by default)
- Members support two networking modes
  - `default` - member networking accepts all protocols on a single port (default: `5701`)
  - `advanced` - each protocol binds to its socket address - the ports are different

Communication Security - Background

For attackers, it's easy to capture unprotected network communication.

- Hazelcast members support different application layer protocols - all TCP-based
  - Hazelcast Member Protocol (`HZC`)
  - Hazelcast Open Binary Client Protocol (`CP2`)
  - HTTP REST
  - memcached protocol (disabled by default)
- Members support two networking modes
  - `default` - member networking accepts all protocols on a single port (default: `5701`)
  - `advanced` - each protocol binds to its socket address - the ports are different
- Members support multiple discovery strategies to find each other
  - Multicast discovery is the default in some Hazelcast packages

Communication Security - Basics - OS/Community

- Advanced networking
  - specify which protocols will be accessible from outside
  - allows employing firewalls
- Management operation protection on Client protocol
  - specify from which IP addresses cluster management operations can be executed
- Untrusted deserialization protection
  - allows configuring allow-lists/deny-lists for deserializing Java classes

Communication Security - Basics - TLS

- TLS/SSL (Transport Layer Security) - based on public-key cryptography; asymmetric encryption used for key exchange; symmetric encryption used for data transmission
  - by default, server doesn't check client's identity; mutual authentication has to be enabled to do so;
  - Hazelcast supports native Java TLS implementation (JSSE) and netty bindings to other engines (OpenSSL & BoringSSL)
- Hazelcast Symmetric Encryption (deprecated feature)
  - only available for Hazelcast Member protocol
  - compared to TLS, it encrypts also the datagrams in Multicast discovery strategy

Communication Security - Rule of thumb

*If a Hazelcast feature communicates **over network**, it **allows configuring encryption** (usually TLS).*

Sample areas where the rule is applied

- supported application protocols
- Hazelcast discovery strategies
- Jet connectors to external sources and sinks
- API calls to Kubernetes and other HTTP client calls

Authentication

- Hazelcast Authentication - part of Hazelcast Security suite
  - security has to be enabled
  - supported on protocols: Member, Client, REST (partly)
- Mutual TLS authentication
  - available for all protocols
  - configured in the networking configuration sections

Authentication - Hazelcast Authentication

*Hazelcast authentication is responsible for **verifying identity of the connecting party** (client-side on the given application protocol). It also **assigns roles** to the party. Roles can be later used during the authorization.*

Out-of-the-box provided mechanisms:

- **`simple`** - users and their roles directly configured in the Hazelcast member configuration file
- **`ldap`** - users and roles configured in an LDAP server
- **`kerberos`** (`+ldap`) - authentication based on kerberos ticket verification, roles can be loaded from underlying LDAP server (if there is any)
- **`tls`** - can be used when the mutual TLS authentication is configured to load roles from the client's X.509 certificate

*There is also a simple **default**/fallback mechanism. It's used when no explicit authentication is configured and security is enabled.*

Authorization / Client permissions

Role-Based Access Control (RBAC) is available in Hazelcast. It can be configured for the Client Protocol.

- Permissions are mapped (granted) to roles in the Member configuration.
- Clients get the roles assigned during authentication.
- Before a task requested by a client is executed, the permission required for the given task is checked.

Simplicity vs flexibility

When users first meet the Client permissions, they can struggle with the configuration complexity. It's a price for greater flexibility of the permission-based solution. It allows

- using custom role names (e.g. from LDAP object "uid=jduke,ou=Accounting,o=Hazelcast" use "Accounting" as the role name)
- assigning permission for certain data structure types and object names (e.g. the `"Accounting"` role has `read & write` access to an `IMap` structure named "Invoices")
- executing some operation from certain remote addresses only;

Persistence Encryption

> *Persistence allows members to recover data by persisting map entries, JCache data, and streaming job snapshots on disk. Members can use persisted data to recover from a planned shutdown, a sudden cluster-wide crash, or a single member failure.*

Hazelcast supports encryption of the data stored in a filesystem by the Persistence feature.

Auditlog

- allows observing some important events, E.g.
  - network connections lifecycle
  - authentications
  - cluster lifecycle
- default implementation uses Hazelcast logging

Secrets Strength Validation

- validate passwords in a member configuration
  - minimal secret length
  - large keyspace
  - no dictionary words
- prints warning banner during member startup
- may be enforced (i.e. prevents the member from starting)

Pluggable Security

Several Hazelcast features contain an SPI that allows users to plug in their implementations.

Security-related integration points/interfaces:

- `LoginModule` (Java interface) - for custom credentials verification and role assignment
- `ICredentialFactory` - enables programmatically generating credentials (username/password combinations or tokens);
- `SocketIntercetptor` - allows custom negotiations when a new connection to a member is established;
- `SecurityInterceptor` - enables implementation of finer-grained authorization (ABAC - Attribute-based access control);
- `Auditlog` - allows different message formatting and logging to various external log systems;
- `SecretStrengthPolicy` - for custom secrets validator;
- `SSLContextFactory` and `SSLEngineFactory` - for custom TLS implementations;
- `IPermissionPolicy` - for custom permission mapping implementation.

Roadmap

- **New REST** with better security implementation
- **Deny permissions**
- support **external secret stores**

Resources

- https://github.com/hazelcast/hazelcast/blob/master/docs/design/security/README.adoc
- https://github.com/memcached/memcached/blob/master/doc/protocol.txt
- https://github.com/hazelcast/hazelcast/blob/v5.3.5/hazelcast/src/main/resources/hazelcast-security-hardened.xml
- https://github.com/hazelcast/hazelcast/blob/v5.3.5/hazelcast/src/main/resources/hazelcast-full-example.xml

Questions?

Thank you