

Spring Kafka Security: Part 3 - Best Practices, CLI Operations & Production Guide

Final part of the comprehensive Spring Kafka Security guide covering security best practices, CLI operations, troubleshooting, and production deployment patterns.

Comparisons & Trade-offs

Security Protocol Comparison

Protocol	Encryption	Authentication	Performance	Use Case
PLAINTEXT	✗ None	✗ None	★★★★★ Highest	Development only
SSL	☑ TLS/SSL	☑ mTLS	★★★ Medium	Simple production
SASL_PLAINTEXT	✗ None	☑ SASL	★★★★ High	Internal networks
SASL_SSL	☑ TLS/SSL	☑ SASL + mTLS	★★ Lower	Full security

Authentication Mechanism Comparison

Mechanism	Security Level	Complexity	Performance	Best For
PLAIN	★★ Low	★★★★★ Simple	★★★★★ Fast	Development
SCRAM-SHA-256	★★★★ Good	★★★ Medium	★★★★ Good	Production
SCRAM-SHA-512	★★★★★ High	★★★ Medium	★★★ Medium	High security
GSSAPI/Kerberos	★★★★★ Highest	★ Complex	★★ Lower	Enterprise

Performance Impact Analysis

Security Overhead Comparison (relative to PLAINTEXT baseline):

PLAINTEXT (Baseline):

SSL:

SASL_PLAINTEXT:

SASL_SSL:

SASL_SSL + ACLs:

100%

80%

95%

70%

65%

Throughput Impact:

Security Level	Throughput	Latency	CPU Usage
PLAINTEXT	100,000/sec	1ms	20%

SSL	80,000/sec	3ms	35%
SASL_PLAINTEXT	95,000/sec	1.5ms	25%
SASL_SSL	70,000/sec	4ms	45%

Memory Usage Impact:

SSL Context: +50MB per application

SASL Authentication: +10MB per application

ACL Processing: +5MB per 1000 ACLs

Common Pitfalls & Best Practices

Critical Security Anti-Patterns to Avoid

✗ SSL/TLS Configuration Mistakes

```
// DON'T - Disabling hostname verification in production
Map<String, Object> props = new HashMap<>();
props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, ""); // BAD!
// This makes you vulnerable to man-in-the-middle attacks

// DON'T - Using self-signed certificates in production
// Self-signed certs provide encryption but no identity verification

// DON'T - Hardcoding passwords in configuration files
props.put(SslConfigs.SSL_KEYSTORE_PASSWORD_CONFIG, "hardcoded-password"); // BAD!
// Use environment variables or secret management systems

// DON'T - Using weak TLS protocols
props.put(SslConfigs.SSL_PROTOCOL_CONFIG, "TLSv1.1"); // BAD!
// Use TLSv1.3 or at least TLSv1.2

// DON'T - Ignoring certificate expiration
// Implement certificate monitoring and renewal processes
```

✗ SASL Authentication Anti-Patterns

```
// DON'T - Using SASL/PLAIN without SSL encryption
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT"); // BAD!
props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
// Credentials are sent in plaintext over the network

// DON'T - Sharing SASL credentials across multiple services
String jaasConfig = "PlainLoginModule required username=\"shared-user\"
password=\"shared-pass\"";
// Each service should have its own credentials
```

```
// DON'T - Storing SASL passwords in plaintext
@Value("${kafka.sasl.password}") // BAD if password is in plaintext in config
private String password;
// Use encrypted configuration or secret management

// DON'T - Using weak passwords for SCRAM
// SCRAM passwords should be strong and unique
```

✗ ACL Configuration Mistakes

```
// DON'T - Overly permissive ACLs
adminClient.createAcls(Arrays.asList(
    new AclBinding(
        new ResourcePattern(ResourceType.TOPIC, "*", PatternType.LITERAL),
        new AccessControlEntry("User:service", "*", AclOperation.ALL,
AclPermissionType.ALLOW)
    )
)); // BAD - gives access to all topics

// DON'T - Forgetting to set allow.everyone.if.no.acl.found=false
// This allows unrestricted access when ACLs are not configured

// DON'T - Not implementing least privilege principle
// Grant only the minimum permissions required for functionality
```

Production Security Best Practices

☑ Comprehensive SSL/TLS Security Configuration

```
/**
 * ☑ GOOD - Production-ready SSL configuration
 */
@Configuration
@lombok.extern.slf4j.Slf4j
public class ProductionSSLSecurityConfiguration {

    @Bean
    public ProducerFactory<String, Object> productionSecureProducerFactory() {
        Map<String, Object> props = new HashMap<>();

        // Basic configuration
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
            System.getenv().getOrDefault("KAFKA_BOOTSTRAP_SERVERS",
"localhost:9093"));
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
JsonSerializer.class);
```

```

// GOOD: Use strongest security protocol
props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");

// GOOD: Use latest TLS version
props.put(SslConfigs.SSL_PROTOCOL_CONFIG, "TLSv1.3");
props.put(SslConfigs.SSL_ENABLED_PROTOCOLS_CONFIG, "TLSv1.3,TLSv1.2");

// GOOD: Enable hostname verification
props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG,
"https");

// GOOD: Use environment variables for sensitive data
props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG,
    System.getenv("KAFKA_SSL_TRUSTSTORE_PATH"));
props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG,
    System.getenv("KAFKA_SSL_TRUSTSTORE_PASSWORD"));
props.put(SslConfigs.SSL_KEYSTORE_LOCATION_CONFIG,
    System.getenv("KAFKA_SSL_KEYSTORE_PATH"));
props.put(SslConfigs.SSL_KEYSTORE_PASSWORD_CONFIG,
    System.getenv("KAFKA_SSL_KEYSTORE_PASSWORD"));
props.put(SslConfigs.SSL_KEY_PASSWORD_CONFIG,
    System.getenv("KAFKA_SSL_KEY_PASSWORD"));

// GOOD: Specify strong cipher suites
props.put(SslConfigs.SSL_CIPHER_SUITES_CONFIG,
"TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256,TLS_AES_128_GCM_SHA256");

// GOOD: Use SCRAM-SHA-512 for strongest SASL security
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-512");

String jaasConfig = String.format(
    "%s required username=\"%s\" password=\"%s\";",
    ScramLoginModule.class.getName(),
    System.getenv("KAFKA_SASL_USERNAME"),
    System.getenv("KAFKA_SASL_PASSWORD")
);
props.put(SaslConfigs.SASL_JAAS_CONFIG, jaasConfig);

// GOOD: Production-optimized performance settings
props.put(ProducerConfig.ACKS_CONFIG, "all");
props.put(ProducerConfig.RETRIES_CONFIG, Integer.MAX_VALUE);
props.put(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG, true);
props.put(ProducerConfig.MAX_IN_FLIGHT_REQUESTS_PER_CONNECTION, 5);

log.info("Configured production-secure producer factory");

return new DefaultKafkaProducerFactory<>(props);
}
}

/**
 * ☒ GOOD - Certificate monitoring and validation

```

```

*/
@Component
@lombok.extern.slf4j.Slf4j
public class CertificateMonitoringService {

    @Value("${kafka.ssl.keystore.location}")
    private String keystorePath;

    @Value("${kafka.ssl.keystore.password}")
    private String keystorePassword;

    @Scheduled(cron = "0 0 6 * * *") // Daily at 6 AM
    public void checkCertificateExpiration() {

        log.info("Checking SSL certificate expiration");

        try {
            KeyStore keystore = KeyStore.getInstance("JKS");
            try (FileInputStream fis = new FileInputStream(keystorePath)) {
                keystore.load(fis, keystorePassword.toCharArray());
            }

            Enumeration<String> aliases = keystore.aliases();
            while (aliases.hasMoreElements()) {
                String alias = aliases.nextElement();
                Certificate cert = keystore.getCertificate(alias);

                if (cert instanceof X509Certificate) {
                    X509Certificate x509Cert = (X509Certificate) cert;
                    Date expirationDate = x509Cert.getNotAfter();
                    Date currentDate = new Date();

                    long daysUntilExpiration = (expirationDate.getTime() -
currentDate.getTime())
                        / (1000 * 60 * 60 * 24);

                    if (daysUntilExpiration <= 30) {
                        log.warn("Certificate {} expires in {} days: {}",
alias, daysUntilExpiration, expirationDate);

                        // Send alert to monitoring system
                        sendCertificateExpirationAlert(alias, daysUntilExpiration,
expirationDate);
                    } else {
                        log.info("Certificate {} is valid until {} ({} days)",
alias, expirationDate, daysUntilExpiration);
                    }
                }
            }
        } catch (Exception e) {
            log.error("Failed to check certificate expiration", e);
        }
    }
}

```

```

        private void sendCertificateExpirationAlert(String alias, long daysLeft, Date
expirationDate) {
            // Implementation would send alerts to monitoring system
            log.error("🚨 CERTIFICATE EXPIRATION ALERT: {} expires in {} days ({})",
                alias, daysLeft, expirationDate);
        }
    }

/**
 * ☒ GOOD - Secure credential management
 */
@Component
@lombok.extern.slf4j.Slf4j
public class SecureCredentialManager {

    private final AESUtil aesUtil;

    public SecureCredentialManager() {
        this.aesUtil = new AESUtil();
    }

    /**
     * Get decrypted SASL password from encrypted configuration
     */
    public String getSaslPassword() {

        String encryptedPassword = System.getenv("KAFKA_SASL_PASSWORD_ENCRYPTED");
        String encryptionKey = System.getenv("KAFKA_ENCRYPTION_KEY");

        if (encryptedPassword == null || encryptionKey == null) {
            throw new IllegalStateException("Encrypted credentials not
configured");
        }

        try {
            String decryptedPassword = aesUtil.decrypt(encryptedPassword,
encryptionKey);
            log.debug("Retrieved and decrypted SASL password");
            return decryptedPassword;
        } catch (Exception e) {
            log.error("Failed to decrypt SASL password", e);
            throw new RuntimeException("Credential decryption failed", e);
        }
    }

    /**
     * Rotate SASL credentials periodically
     */
    @Scheduled(cron = "0 0 2 1 * *") // First day of each month at 2 AM
    public void rotateCredentials() {

        log.info("Starting credential rotation process");
    }
}

```

```

    try {
        // Generate new password
        String newPassword = generateSecurePassword();

        // Update SCRAM credentials in Kafka
        updateScramCredentials(getCurrentUsername(), newPassword);

        // Update encrypted password in configuration store
        updateEncryptedPassword(newPassword);

        log.info("Credential rotation completed successfully");

    } catch (Exception e) {
        log.error("Credential rotation failed", e);
        // Send alert to monitoring system
    }
}

private String generateSecurePassword() {
    SecureRandom random = new SecureRandom();
    StringBuilder password = new StringBuilder();

    String chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*";

    for (int i = 0; i < 32; i++) {
        password.append(chars.charAt(random.nextInt(chars.length())));
    }

    return password.toString();
}

private void updateScramCredentials(String username, String newPassword) {
    // Implementation would update SCRAM credentials in Kafka
    log.debug("Updated SCRAM credentials for user: {}", username);
}

private void updateEncryptedPassword(String newPassword) {
    // Implementation would update encrypted password in secure store
    log.debug("Updated encrypted password in configuration store");
}

private String getCurrentUsername() {
    return System.getenv("KAFKA_SASL_USERNAME");
}
}

/**
 * ☒ GOOD - Comprehensive ACL management
 */
@Service
@lombok.extern.slf4j.Slf4j
public class ProductionACLService {

```

```

@Autowired
private AdminClient adminClient;

/**
 * Initialize ACLs for a new application service
 */
public void initializeServiceACLs(ServiceACLRequest request) {

    log.info("Initializing ACLs for service: {}", request.getServiceName());

    try {
        List<AclBinding> aclBindings = new ArrayList<>();

        // Producer permissions
        for (String topic : request.getProducerTopics()) {

aclBindings.addAll(createProducerACLs(request.getServicePrincipal(), topic));
        }

        // Consumer permissions
        for (ConsumerACLRequest consumerRequest :
request.getConsumerRequests()) {
            aclBindings.addAll(createConsumerACLs(
                request.getServicePrincipal(),
                consumerRequest.getTopic(),
                consumerRequest.getConsumerGroup()
            ));
        }

        // Create all ACLs in batch
        CreateAclsResult result = adminClient.createAcls(aclBindings);
        result.all().get(60, TimeUnit.SECONDS);

        log.info("Service ACLs initialized successfully: service={}, ACLs={}",
            request.getServiceName(), aclBindings.size());

        // Audit log
        auditACLCreation(request.getServiceName(), aclBindings);

    } catch (Exception e) {
        log.error("Failed to initialize service ACLs: service={}",
request.getServiceName(), e);
        throw new RuntimeException("Service ACL initialization failed", e);
    }
}

/**
 * Apply principle of least privilege
 */
private List<AclBinding> createProducerACLs(String principal, String topic) {
    return Arrays.asList(
        // Minimal permissions for producer
        createACL(principal, ResourceType.TOPIC, topic, AclOperation.WRITE),

```



```

        createACL(principal, ResourceType.TOPIC, topic, AclOperation.DESCRIBE)
        // Note: CREATE permission not granted by default for security
    );
}

private List<AclBinding> createConsumerACLs(String principal, String topic,
String consumerGroup) {
    return Arrays.asList(
        // Minimal permissions for consumer
        createACL(principal, ResourceType.TOPIC, topic, AclOperation.READ),
        createACL(principal, ResourceType.TOPIC, topic,
AclOperation.DESCRIBE),
        createACL(principal, ResourceType.GROUP, consumerGroup,
AclOperation.READ)
    );
}

private AclBinding createACL(String principal, ResourceType resourceType,
String resourceName, AclOperation operation) {
    return new AclBinding(
        new ResourcePattern(resourceType, resourceName, PatternType.LITERAL),
        new AccessControlEntry(principal, "*", operation,
AclPermissionType.ALLOW)
    );
}

private void auditACLCreation(String serviceName, List<AclBinding>
aclBindings) {
    log.info("ACL_AUDIT: Created {} ACLs for service: {}", aclBindings.size(),
serviceName);

    for (AclBinding acl : aclBindings) {
        log.info("ACL_AUDIT: {} - {}:{} - {} - {}",
            serviceName,
            acl.pattern().resourceType(),
            acl.pattern().name(),
            acl.entry().operation(),
            acl.entry().permissionType()
        );
    }
}

/**
 * Regular ACL audit and cleanup
 */
@Scheduled(cron = "0 0 3 * * SUN") // Every Sunday at 3 AM
public void auditAndCleanupACLs() {

    log.info("Starting weekly ACL audit and cleanup");

    try {
        // Get all ACLs
        DescribeAclsResult result =
adminClient.describeAcls(AclBindingFilter.ANY);

```

```

        Collection<AclBinding> allAcls = result.values().get(60,
TimeUnit.SECONDS);

        // Group by principal for analysis
        Map<String, List<AclBinding>> aclsByPrincipal = allAcls.stream()
            .collect(Collectors.groupingBy(acl -> acl.entry().principal()));

        // Analyze and report
        for (Map.Entry<String, List<AclBinding>> entry :
aclsByPrincipal.entrySet()) {
            String principal = entry.getKey();
            List<AclBinding> principalAcls = entry.getValue();

            analyzeUserACLs(principal, principalAcls);
        }

        log.info("ACL audit completed: total principals={}, total ACLs={}",
            aclsByPrincipal.size(), allAcls.size());

    } catch (Exception e) {
        log.error("ACL audit failed", e);
    }
}

private void analyzeUserACLs(String principal, List<AclBinding> acls) {
    log.info("ACL_ANALYSIS: Principal={}, ACL_Count={}", principal,
acls.size());

    // Check for overly permissive ACLs
    long wildcardACLs = acls.stream()
        .filter(acl -> "*".equals(acl.pattern().name()))
        .count();

    if (wildcardACLs > 0) {
        log.warn("ACL_WARNING: Principal {} has {} wildcard ACLs", principal,
wildcardACLs);
    }

    // Check for ALL operation ACLs
    long allOperationACLs = acls.stream()
        .filter(acl -> acl.entry().operation() == AclOperation.ALL)
        .count();

    if (allOperationACLs > 0) {
        log.warn("ACL_WARNING: Principal {} has {} ALL operation ACLs",
principal, allOperationACLs);
    }
}
}

```

Kafka Security CLI Commands

```
#!/bin/bash

# =====
# SSL Certificate Management
# =====

# Check SSL connection to Kafka broker
openssl s_client -connect kafka.example.com:9093 -servername kafka.example.com

# Verify certificate details
keytool -list -v -keystore kafka.client.keystore.jks -storepass password

# Check certificate expiration
keytool -list -keystore kafka.client.keystore.jks -storepass password | grep
"Valid until"

# =====
# SASL/SCRAM User Management
# =====

# Create SCRAM-SHA-256 user
kafka-configs.sh --bootstrap-server localhost:9093 \
  --alter \
  --add-config 'SCRAM-SHA-256=[password=user-password]' \
  --entity-type users \
  --entity-name alice \
  --command-config client.properties

# Create SCRAM-SHA-512 user (more secure)
kafka-configs.sh --bootstrap-server localhost:9093 \
  --alter \
  --add-config 'SCRAM-SHA-512=[iterations=8192,password=strong-password]' \
  --entity-type users \
  --entity-name bob \
  --command-config client.properties

# List SCRAM users
kafka-configs.sh --bootstrap-server localhost:9093 \
  --describe \
  --entity-type users \
  --command-config client.properties

# Delete SCRAM user
kafka-configs.sh --bootstrap-server localhost:9093 \
  --alter \
  --delete-config 'SCRAM-SHA-256' \
  --entity-type users \
  --entity-name alice \
  --command-config client.properties
```

```
# =====
# ACL Management
# =====

# Create producer ACLs
kafka-acls.sh --bootstrap-server localhost:9093 \
  --add \
  --allow-principal User:producer-service \
  --operation Write \
  --operation Describe \
  --topic orders \
  --command-config client.properties

# Create consumer ACLs
kafka-acls.sh --bootstrap-server localhost:9093 \
  --add \
  --allow-principal User:consumer-service \
  --operation Read \
  --operation Describe \
  --topic orders \
  --group order-processors \
  --command-config client.properties

# Create admin ACLs
kafka-acls.sh --bootstrap-server localhost:9093 \
  --add \
  --allow-principal User:admin \
  --operation All \
  --cluster \
  --command-config client.properties

# List all ACLs
kafka-acls.sh --bootstrap-server localhost:9093 \
  --list \
  --command-config client.properties

# List ACLs for specific principal
kafka-acls.sh --bootstrap-server localhost:9093 \
  --list \
  --principal User:alice \
  --command-config client.properties

# Delete ACLs
kafka-acls.sh --bootstrap-server localhost:9093 \
  --remove \
  --allow-principal User:alice \
  --operation Read \
  --topic orders \
  --command-config client.properties

# =====
# Security Testing
# =====
```

```

# Test producer with SASL/SSL
kafka-console-producer.sh --bootstrap-server localhost:9093 \
  --topic test-topic \
  --producer.config client-sasl-ssl.properties

# Test consumer with SASL/SSL
kafka-console-consumer.sh --bootstrap-server localhost:9093 \
  --topic test-topic \
  --from-beginning \
  --consumer.config client-sasl-ssl.properties

# Test connection with wrong credentials (should fail)
kafka-console-producer.sh --bootstrap-server localhost:9093 \
  --topic test-topic \
  --producer.config wrong-credentials.properties

# =====
# Security Monitoring
# =====

# Monitor failed authentication attempts
kafka-run-class.sh kafka.tools.JmxTool \
  --object-name
kafka.server:type=BrokerTopicMetrics,name=FailedAuthenticationTotal \
  --jmx-url service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi

# Monitor successful authentication rate
kafka-run-class.sh kafka.tools.JmxTool \
  --object-name
kafka.server:type=BrokerTopicMetrics,name=SuccessfulAuthenticationRate \
  --jmx-url service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi

# Check SSL handshake failures
kafka-run-class.sh kafka.tools.JmxTool \
  --object-name kafka.server:type=socket-server-
metrics,listener=SSL,networkProcessor=* \
  --jmx-url service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi

# =====
# Configuration Files
# =====

# client-sasl-ssl.properties
cat > client-sasl-ssl.properties << EOF
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-256
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required
\
  username="client-user" \
  password="client-password";

ssl.truststore.location=/path/to/kafka.client.truststore.jks
ssl.truststore.password=truststore-password
ssl.endpoint.identification.algorithm=https

```

```

EOF

# server.properties (broker security configuration)
cat > server-security.properties << EOF
# Listeners
listeners=PLAINTEXT://localhost:9092,SASL_SSL://localhost:9093
advertised.listeners=PLAINTEXT://localhost:9092,SASL_SSL://localhost:9093
security.inter.broker.protocol=SASL_SSL
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SASL_SSL:SASL_SSL

# SASL Configuration
sasl.mechanism.inter.broker.protocol=SCRAM-SHA-512
sasl.enabled.mechanisms=SCRAM-SHA-256,SCRAM-SHA-512

# SSL Configuration
ssl.keystore.location=/path/to/kafka.server.keystore.jks
ssl.keystore.password=keystore-password
ssl.key.password=key-password
ssl.truststore.location=/path/to/kafka.server.truststore.jks
ssl.truststore.password=truststore-password
ssl.endpoint.identification.algorithm=
ssl.client.auth=none

# ACL Configuration
authorizer.class.name=kafka.security.authorizer.AclAuthorizer
allow.everyone.if.no.acl.found=false
super.users=User:admin;User:kafka

# JAAS Configuration
listener.name.sasl_ssl.scram-sha-
256.sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required \
    username="kafka" \
    password="kafka-secret";
EOF

```

Security Health Check Scripts

```

#!/bin/bash

# =====
# Comprehensive Kafka Security Health Check
# =====

BOOTSTRAP_SERVERS="localhost:9093"
CLIENT_CONFIG="client-sasl-ssl.properties"
LOG_FILE="kafka-security-health.log"

echo "Starting Kafka Security Health Check..." | tee $LOG_FILE
echo "Timestamp: $(date)" | tee -a $LOG_FILE

```

```
# Test 1: SSL Connection Test
echo "=== SSL Connection Test ===" | tee -a $LOG_FILE
timeout 10 openssl s_client -connect ${BOOTSTRAP_SERVERS} -servername kafka
</dev/null 2>&1 | \
    grep -E "(CONNECTED|Verify return code)" | tee -a $LOG_FILE

# Test 2: SASL Authentication Test
echo "=== SASL Authentication Test ===" | tee -a $LOG_FILE
echo "test-message" | timeout 10 kafka-console-producer.sh \
    --bootstrap-server $BOOTSTRAP_SERVERS \
    --topic health-check-topic \
    --producer.config $CLIENT_CONFIG 2>&1 | \
    tee -a $LOG_FILE

# Test 3: ACL Verification
echo "=== ACL Verification ===" | tee -a $LOG_FILE
kafka-acls.sh --bootstrap-server $BOOTSTRAP_SERVERS \
    --list \
    --command-config $CLIENT_CONFIG 2>&1 | \
    head -20 | tee -a $LOG_FILE

# Test 4: Certificate Expiration Check
echo "=== Certificate Expiration Check ===" | tee -a $LOG_FILE
KEYSTORE_PATH="/path/to/kafka.client.keystore.jks"
KEYSTORE_PASSWORD="keystore-password"

if [ -f "$KEYSTORE_PATH" ]; then
    keytool -list -keystore $KEYSTORE_PATH -storepass $KEYSTORE_PASSWORD
2>/dev/null | \
    grep "Valid until" | tee -a $LOG_FILE
else
    echo "Keystore not found: $KEYSTORE_PATH" | tee -a $LOG_FILE
fi

# Test 5: SCRAM User Check
echo "=== SCRAM Users Check ===" | tee -a $LOG_FILE
kafka-configs.sh --bootstrap-server $BOOTSTRAP_SERVERS \
    --describe \
    --entity-type users \
    --command-config $CLIENT_CONFIG 2>&1 | \
    head -10 | tee -a $LOG_FILE

echo "Security health check completed. Results saved to $LOG_FILE"
```

Version Highlights

Spring Kafka Security Evolution

Version	Release	Key Security Features
3.2.x	2024	SSL Bundle integration, enhanced credential management

Version	Release	Key Security Features
3.1.x	2024	SSL Bundle support , improved certificate handling
3.0.x	2023	Native compilation security support, GraalVM compatibility
2.9.x	2022	OAuth 2.0/OIDC authentication support
2.8.x	2022	Enhanced SASL configuration, better error handling
2.7.x	2021	Delegated authentication support, improved SSL
2.6.x	2021	SASL/OAUTHBEARER mechanism support
2.5.x	2020	Improved ACL integration with Spring Security
2.4.x	2020	Enhanced SSL configuration options
2.3.x	2019	Basic SASL/SCRAM support in Spring Boot

Apache Kafka Security Milestones

Kafka 3.6+ (2024):

- **Enhanced KRaft Security:** Improved metadata security in KRaft mode
- **OAuth 2.0 Improvements:** Better JWT token handling
- **Certificate Rotation:** Hot certificate reload capabilities

Kafka 3.0+ (2021):

- **KRaft Security:** Security support in ZooKeeper-free mode
- **Improved ACL Performance:** Faster ACL evaluation
- **Enhanced Audit Logging:** Better security event tracking

Kafka 2.8+ (2021):

- **KRaft Mode Introduction:** Foundation for ZooKeeper-free security
- **ACL Improvements:** Enhanced authorization performance

Kafka 2.0+ (2018):

- **Delegation Tokens:** Lightweight authentication mechanism
- **ACL CLI Improvements:** Better tooling for ACL management
- **Security Protocol Validation:** Stronger configuration validation

Production Security Checklist

Essential Security Configuration

- ☒ **Use SASL_SSL protocol** for all client connections
- ☒ **Enable TLSv1.3** or at least TLSv1.2 for SSL
- ☒ **Implement certificate monitoring** and rotation
- ☒ **Use SCRAM-SHA-256/512** instead of PLAIN authentication

- ☒ **Enable hostname verification** (ssl.endpoint.identification.algorithm=https)
- ☒ **Set allow.everyone.if.no.acl.found=false** for ACL enforcement
- ☒ **Implement least privilege ACLs** for all services
- ☒ **Use environment variables** for sensitive configuration
- ☒ **Enable audit logging** for security events
- ☒ **Implement credential rotation** procedures

Security Monitoring Setup

- ☒ **Monitor failed authentication attempts**
- ☒ **Track SSL handshake failures**
- ☒ **Alert on certificate expiration**
- ☒ **Monitor ACL violations**
- ☒ **Log security configuration changes**
- ☒ **Set up security health checks**

Operational Security Procedures

- ☒ **Regular security audits** of ACLs and users
- ☒ **Incident response plan** for security breaches
- ☒ **Backup and recovery** procedures for security configurations
- ☒ **Security training** for development and operations teams

Key Security Takeaways

Essential Security Hierarchy

1. **Network Security:** Use SASL_SSL protocol with TLSv1.3
2. **Authentication:** Prefer SCRAM-SHA-512 over PLAIN
3. **Authorization:** Implement fine-grained ACLs with least privilege
4. **Encryption:** Ensure end-to-end encryption of sensitive data
5. **Monitoring:** Implement comprehensive security monitoring and alerting

Security vs Performance Trade-offs

- **SASL_SSL:** 20-30% performance overhead, but essential for production
- **Strong Encryption:** Higher CPU usage, but critical for data protection
- **ACL Enforcement:** Minimal overhead, provides essential authorization
- **Certificate Validation:** Small latency increase, prevents MITM attacks

Critical Security Principles

1. **Defense in Depth:** Multiple security layers (network, authentication, authorization)
2. **Least Privilege:** Grant only required permissions
3. **Zero Trust:** Verify and authenticate all connections
4. **Continuous Monitoring:** Real-time security event monitoring
5. **Regular Audits:** Periodic security configuration reviews

Last Updated: September 2025

Spring Kafka Version Coverage: 3.2.x

Apache Kafka Compatibility: 3.6.x

Spring Boot Version: 3.2.x

This comprehensive Spring Kafka Security guide provides production-ready patterns for implementing robust security in Kafka applications, from basic SSL setup to advanced ACL management, ensuring data protection and compliance in distributed systems.

[632] [633] [634]