9月12日汇报

当服务器接收到客户端的assertion时，由JWTClientAuthenticator类中的authenticateClient函数完成对assertion的认证

    public void authenticateClient(ClientAuthenticationFlowContext context) {

        JWTClientValidator validator = new JWTClientValidator(context, getId());

        if (!validator.clientAssertionParametersValidation()) return;

        try {

            validator.readJws();

            if (!validator.validateClient()) return;

            if (!validator.validateSignatureAlgorithm()) return;

            RealmModel realm = validator.getRealm();

            ClientModel client = validator.getClient();

            JWSInput jws = validator.getJws();

            JsonWebToken token = validator.getToken();

            String clientAssertion = validator.getClientAssertion();

            // Get client key and validate signature

            PublicKey clientPublicKey = getSignatureValidationKey(client, context, jws);

            if (clientPublicKey == null) {

                // Error response already set to context

                return;

            }

            boolean signatureValid;

            try {

                JsonWebToken jwt = context.getSession().tokens().decodeClientJWT(clientAssertion, client, (jose, validatedClient) -> {

                    DEFAULT\_VALIDATOR.accept(jose, validatedClient);

                    String signatureAlgorithm = jose.getHeader().getRawAlgorithm();

                    ClientSignatureVerifierProvider signatureProvider = context.getSession().getProvider(ClientSignatureVerifierProvider.class, signatureAlgorithm);

                    if (signatureProvider == null) {

                        throw new RuntimeException("Algorithm not supported");

                    }

                    if (!signatureProvider.isAsymmetricAlgorithm()) {

                        throw new RuntimeException("Algorithm is not asymmetric");

                    }

                }, JsonWebToken.class);

                signatureValid = jwt != null;

            } catch (RuntimeException e) {

                Throwable cause = e.getCause() != null ? e.getCause() : e;

                throw new RuntimeException("Signature on JWT token failed validation", cause);

            }

            if (!signatureValid) {

                throw new RuntimeException("Signature on JWT token failed validation");

            }

            validator.validateTokenAudience(context, realm, token);

            validator.validateToken();

            validator.validateTokenReuse();

            context.success();

        } catch (Exception e) {

            ServicesLogger.LOGGER.errorValidatingAssertion(e);

            Response challengeResponse = ClientAuthUtil.errorResponse(Response.Status.BAD\_REQUEST.getStatusCode(), OAuthErrorException.INVALID\_CLIENT, "Client authentication with signed JWT failed: " + e.getMessage());

            context.failure(AuthenticationFlowError.INVALID\_CLIENT\_CREDENTIALS, challengeResponse);

        }

    }

做了如下几步目前我们比较关注的工作

先读取并解析JWT令牌

            validator.readJws();

验证发来的client\_id和发来的JWT中的client\_id是否相同

            if (!validator.validateClient()) return;

        String clientIdParam = params.getFirst(OAuth2Constants.CLIENT\_ID);

        if (clientIdParam != null && !clientIdParam.equals(clientId)) {

        //  clientIdParam请求中的client\_id   clientId:assertion中绑定的client\_id

            throw new RuntimeException("client\_id parameter not matching with client from JWT token");

        }

    public boolean validateClient() {

        if (token == null) throw new IllegalStateException("Incorrect usage. Variable 'token' is null. Need to read JWS first before validateClient");

        String clientId = token.getSubject();

        if (clientId == null) {

            throw new RuntimeException("Can't identify client. Subject missing on JWT token");

        }

        if (!clientId.equals(token.getIssuer())) {

            throw new RuntimeException("Issuer mismatch. The issuer should match the subject");

        }

        String clientIdParam = params.getFirst(OAuth2Constants.CLIENT\_ID);

        if (clientIdParam != null && !clientIdParam.equals(clientId)) {

            throw new RuntimeException("client\_id parameter not matching with client from JWT token");

        }

        context.getEvent().client(clientId);

        client = realm.getClientByClientId(clientId);

        if (client == null) {

            context.failure(AuthenticationFlowError.CLIENT\_NOT\_FOUND, null);

            return false;

        } else {

            context.setClient(client);

        }

        if (!client.isEnabled()) {

            context.failure(AuthenticationFlowError.CLIENT\_DISABLED, null);

            return false;

        }

        if (!clientAuthenticatorProviderId.equals(client.getClientAuthenticatorType())) {

            context.failure(AuthenticationFlowError.INVALID\_CLIENT\_CREDENTIALS, null);

            return false;

        }

        return true;

    }

先会生成一个预期的audience，然后验证发来的assertion的audience中是否至少包含一个预期的audience

            validator.validateTokenAudience(context, realm, token);

validateTokenAudience

    public void validateTokenAudience(ClientAuthenticationFlowContext context, RealmModel realm, JsonWebToken token) {

        List<String> expectedAudiences = getExpectedAudiences(context, realm);

        if (!token.hasAnyAudience(expectedAudiences)) {

            throw new RuntimeException("Token audience doesn't match domain. Expected audiences are any of " + expectedAudiences

                    + " but audience from token is '" + Arrays.asList(token.getAudience()) + "'");

        }

        if (!isAllowMultipleAudiencesForJwtClientAuthentication(context) && token.getAudience().length > 1) {

            throw new RuntimeException("Multiple audiences not allowed in the JWT token for client authentication");

        }

    }

JWTClientValidator类中validateTokenAudience方法

生成预期audience列表

        List<String> expectedAudiences = getExpectedAudiences(context, realm);

可以通过配置决定是否只接受一个audience，如果只接受一个audience则无法进行audience injecte攻击。如果能接受列表，则可以进行audience inject攻击

        if (!isAllowMultipleAudiencesForJwtClientAuthentication(context) && token.getAudience().length > 1) {

            throw new RuntimeException("Multiple audiences not allowed in the JWT token for client authentication");

        }

    private boolean isAllowMultipleAudiencesForJwtClientAuthentication(ClientAuthenticationFlowContext context) {

        OIDCLoginProtocol loginProtocol = (OIDCLoginProtocol) context.getSession().getProvider(LoginProtocol.class, OIDCLoginProtocol.LOGIN\_PROTOCOL);

        OIDCProviderConfig config = loginProtocol.getConfig();

        return config.isAllowMultipleAudiencesForJwtClientAuthentication();

    }

用于配置的类

    public OIDCProviderConfig(Config.Scope config) {

        this.additionalReqParamsMaxNumber = config.getInt(OIDCLoginProtocolFactory.CONFIG\_OIDC\_REQ\_PARAMS\_MAX\_NUMBER, DEFAULT\_ADDITIONAL\_REQ\_PARAMS\_MAX\_NUMBER);

        this.additionalReqParamsMaxSize = config.getInt(OIDCLoginProtocolFactory.CONFIG\_OIDC\_REQ\_PARAMS\_MAX\_SIZE, DEFAULT\_ADDITIONAL\_REQ\_PARAMS\_MAX\_SIZE);

        this.additionalReqParamsMaxOverallSize = config.getInt(OIDCLoginProtocolFactory.CONFIG\_OIDC\_REQ\_PARAMS\_MAX\_OVERALL\_SIZE, DEFAULT\_ADDITIONAL\_REQ\_PARAMS\_MAX\_OVERALL\_SIZE);

        this.additionalReqParamsFailFast = config.getBoolean(OIDCLoginProtocolFactory.CONFIG\_OIDC\_REQ\_PARAMS\_FAIL\_FAST, DEFAULT\_ADDITIONAL\_REQ\_PARAMS\_FAIL\_FAST);

        this.allowMultipleAudiencesForJwtClientAuthentication = config.getBoolean(OIDCLoginProtocolFactory.CONFIG\_OIDC\_ALLOW\_MULTIPLE\_AUDIENCES\_FOR\_JWT\_CLIENT\_AUTHENTICATION, DEFAULT\_ALLOW\_MULTIPLE\_AUDIENCES\_FOR\_JWT\_CLIENT\_AUTHENTICATION);

    }

该方法之中检查了assertion是否处于有效期内 默认有效期为60s

            validator.validateToken();

        if (!token.isActive(ALLOWED\_CLOCK\_SKEW)) {

//检查assertion是否过期，有15s偏差时间

            throw new RuntimeException("Token is not active");

        }

    public int getTokenEndpointAuthSigningMaxExp() {

    //设置assertion的有效期，默认60s，也可以从配置中获得配置值

        final String value = getAttribute(OIDCConfigAttributes.TOKEN\_ENDPOINT\_AUTH\_SIGNING\_MAX\_EXP);

        try {

            final int maxExp = Integer.parseInt(value);

            if (maxExp > 0) {

                return maxExp;

            }

        } catch (NumberFormatException e) {

            // ignore and return default value

        }

        return 60; // default to 60s

    }

获取期望audience列表

    private List<String> getExpectedAudiences(ClientAuthenticationFlowContext context, RealmModel realm) {

        String issuerUrl = Urls.realmIssuer(context.getUriInfo().getBaseUri(), realm.getName());

        String tokenUrl = OIDCLoginProtocolService.tokenUrl(context.getUriInfo().getBaseUriBuilder()).build(realm.getName()).toString();

        String tokenIntrospectUrl = OIDCLoginProtocolService.tokenIntrospectionUrl(context.getUriInfo().getBaseUriBuilder()).build(realm.getName()).toString();

        String parEndpointUrl = ParEndpoint.parUrl(context.getUriInfo().getBaseUriBuilder()).build(realm.getName()).toString();

        List<String> expectedAudiences = new ArrayList<>(Arrays.asList(issuerUrl, tokenUrl, tokenIntrospectUrl, parEndpointUrl));

        String backchannelAuthenticationUrl = CibaGrantType.authorizationUrl(context.getUriInfo().getBaseUriBuilder()).build(realm.getName()).toString();

        expectedAudiences.add(backchannelAuthenticationUrl);

        return expectedAudiences;

    }