Livrable 4.3.2: Implantation des composants d'audit et gouvernance

Janvier 2015

LORIA

Ahmed BOUCHAMI, Olivier PERRIN

Contents

1 Audit et gouvernance des ressources collaboratives dans OpenPaaS

Ce document présente l'implantation du service d'audit et de gouvernance.

Dans cette partie, nous proposons une approche de suivi en temps réel du comportement des utilisateurs dans la plate-forme. Nous proposons également des mesures de prévention face aux comportements suspicieux des utilisateurs vis-à-vis des ressources collaboratives déployées. Cette supervision se fait grâce à l'évaluation de la conduite (les actions) de chaque utilisateur au sein de sa communauté. Pour cela, nous référençons cette évaluation sous le nom *confiance numérique* ou par le terme *trust* qui est plus utilisé dans le domaine de la supervision numérique. L'évaluation du *trust* nous permet de modifier le comportement du service d'autorisation présenté précédemment. De ce fait, la décision de suspendre (ou pas) l'accès d'un utilisateur à l'ensemble des ressources d'une (ou éventuellement plusieurs) session(s) ne dépend plus uniquement des règles/politiques stockées dans le RSE, mais également par la dynamique des actions initiées par un utilisateur. La décision est basée sur la comparaison de la valeur du *trust* calculée par rapport à un seuil donné. Le calcul de la valeur du *trust* peut être calculé à la volée (calculé en temps réel), ou bien à la fin de chaque session.

L'évaluation du *trust* d'un utilisateur débouche sur l'évolution de la valeur de *trust*, là où elle était auparavant statique. Dans le cadre de la plate-forme collaborative *OpenPaaS*, nous suggérons que l'on prenne en compte le nombre de tentatives d'accès aux ressources pour lesquelles l'utilisateur ne dispose pas des droits d'accès requis. En d'autres termes, supposons qu'un utilisateur n'ait pas le droit d'accès à une ressource donnée (déployée dans *OpenPaaS*). S'il tente plusieurs fois d'y accéder, le système de contrôle d'accès va lui indiquer qu'il n'a pas les autorisations nécessaires, et le système de supervision va adapter sa valeur de trust suite à son comportement jugé par le système de supervision comme pouvant être suspicieux.

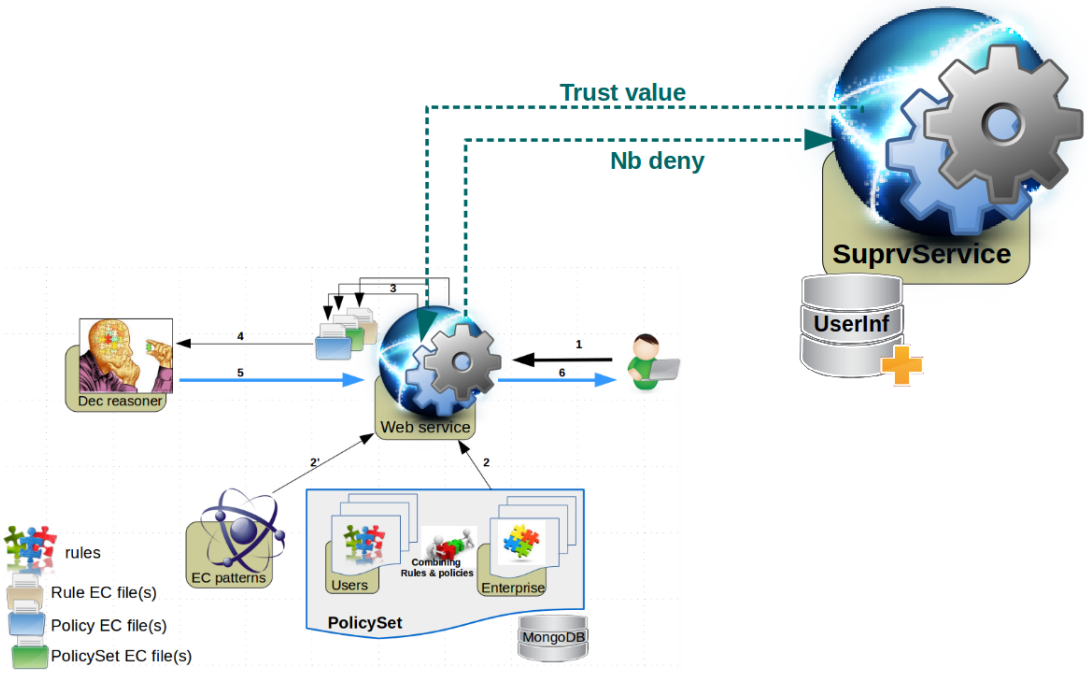

Figure 1: Architecture du système de contrôle d'accès avec le module de gouvernance

1.1 Implantation

Comme illustré dans la figure 1, le module de supervision qui est intégré dans le service de contrôle d'accès consulte un service de gouvernance, nommé *SuprvService*. Ce dernier reçoit en entrée les informations (logs) concernant l'interaction d'un utilisateur donné pour estimer les valeurs de confiance. Ensuite, il met à jour les informations concernant l'historique des expériences de l'utilisateur dans *OpenPaaS*. Ceci signifie que le service de gouvernance **SuprvService** prend en considération de l'**historique** dans l'estimation des valeurs de confiance (**trust**) des utilisateurs.

1.1.1 Algorithme

Notre algorithme d'estimation est basé sur la fonction *exponentialDecay*: $N(t) = N_0 e^{-\lambda}$.

Cette fonction permet d'évaluer l'évolution (positive ou négative) des résultats d'un phénomène à partir d'au moins deux échantillons extraits à des instants différents $\{t_0 ... t_n\}$. L'évolution est déduite à partir de la valeur du *DecayFactor* λ: si $\lambda > 0 \rightarrow Growth$, si $\lambda < 0 \rightarrow Decay$.

La valeur de λ nous permet de projeter le nombre de requêtes rejetées (*nbDeny*) sur la valeur de *trust* correspondante (cf. figure2). À chaque fin de session, le *DecayFactor* λ est recalculé en fonction de l'évolution (à la hausse ou la baisse) du *trust*.

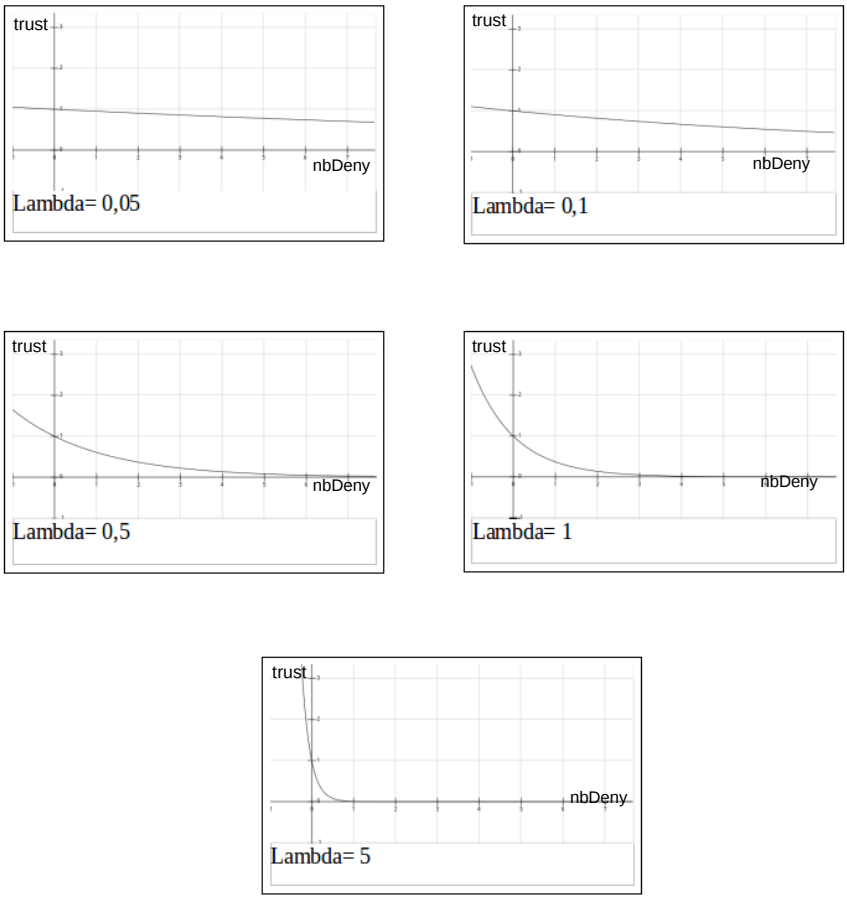

Figure 2: Évolution du trust par rapport au DecayFacor

Les principales étapes de l'algorithme d'estimation du *trust* sont les suivantes:

Phase 1 : calcul du trust de la session courante

1. Récupération de la dernière valeur du *DecayFactor* depuis la base de données;
2. Calcul de la valeur de *trust* pour la session qui vient de se terminer (voir figure 4), en passant en paramètres l'identifiant de l'utilisateur, le nombre de deny et le facteur de pénalisation de la communauté;
    - chaque communauté permet un nombre maximum de tentatives d'accès non abouties. Si ce nombre est atteint par un utilisateur, la session sera **interrompue** (figure 3), et la valeur du *trust* sera égale au facteur de pénalisation qui dépend de la sévérité de la communauté en question (*comSeverity*). Ce facteur de pénalisation est de l'ordre de ϵ, par conséquent, plus il sera petit, plus l'utilisateur en question aura du mal à faire évoluer son *trust*.
    - la méthode de calcul de trust est paramétrable par les différentes valeurs que pourrait prendre le *decayFactor*, à condition que $\lambda < 0$.

```
//killing session
    if (trust>=nbMaxDeny){
        HttpSession session = request.getSession(false);
        if (session != null) {
            session.invalidate();
        }
    }
```

Figure 3: Interruption d'une session

Phase 2 : estimation de l'évolution du trust

3. Récupération de la dernière valeur de trust depuis la base de données: *TrustHistory*: ($\rightarrow$ *TrustStore >> getTrustHistory(iduser)*)
4. Calcul de la moyenne des valeurs antérieures de trust. On donne plus d'importance à la dernière valeur dans l'historique lui passant un coefficient 2: ($\rightarrow$ *ExpDecay >>calculateAverageOfTrustHistory (trStore.getTrustHistory(iduser)*)
5. Calcul du *DecayFactor* propre à la session qui vient de se terminer, pour qu'il soit appliqué pour la session suivante: ($\rightarrow$ *ExpDecay >>lambdaComputing(pastTrValue, currentTrValue)*)
6. Récupération du *DecayFactor* le plus proche par rapport à l'évolution du DecayFactor: ($\rightarrow$ *ExpDecay >>calculateNewDecayFactor*)
    - la méthode *calculateNewDecayFactor* prend en paramètres le *decayFactor* de la dernière session **additionné** à celui de la session courante (past DecayFactors consideration);
    - cette méthode retourne le *decayFactor* le plus proche de celui calculé par rapport aux decayFactors de référence qui sont prédéfinis (pour notre prototype nous avons choisi *trEvaluationReferences* $= \{-5, -1, -0.5, -0.1, -0.05\}$, cf. figure 2). La méthode *calculateNewDecayFactor* est basée sur l'algorithme de *recherche dichotomique* pour répondre aux besoins d'optimisation dans le cas d'un *trEvaluationReferences* contenant un grand nombre d'éléments.
7. Mise à jour de la base de données MongoDB contenant l'historique du *trust* et du *decayFactor* de l'utilisateur après le calcul de la nouvelle valeur de trust.

1.1.2 Développement

Nous avons implanté l'algorithme d'estimation de *trust* défini ci-dessus en Java, et nous l'avons exposé en tant que service Web de type REST accessible grâce à GET (cf. figure 4).

```java
@GET
public Response trComputing(@QueryParam("subject") String subject, @QueryParam("nbDeny") int nbDeny, @QueryParam("comSeverity")
    double comSeverity) throws IOException {

        double trVal = evaluationTrust(subject, nbDeny, comSeverity);

       return Response.status(Response.Status.OK).entity("trust value = " + trVal).build();
}
```

Figure 4: Calcul du *trust*.

Nous utilisons une base de données MongoDB pour le stockage des informations concernant l'historique des utilisateurs (*trust, decayFactor*,...). Comme le montre la figure 4, la méthode qui se charge du calcul de la valeur de trust *trComputing* prend en paramètres l'identifiant de l'utilisateur (requester), le nombre de Deny et la sévérité de la communauté en question. Ce dernier facteur représente le degré de pénalisation d'un utilisateur ayant dépassé le nombre de tentatives d'accès maximum (abus) autorisé par la communauté en question. Ce facteur de sévérité est un nombre réel positif $\in [0, 1]$, choisi tel que plus le nombre se rapproche de zéro (est plus petit), plus l'utilisateur aura du mal à faire évoluer sa valeur de confiance au sein de la communauté en question. La méthode *trComputing* est accessible via une requête HTTP du type :
`http://.../AuditService/trustComputing?subject=Member&nbDeny=2&comSeverity=0.01`
La méthode *trComputing* est basée sur la méthode *evaluationTrust* (figure 5) qui prend en paramètres les mêmes paramètres que *trComputing*, à savoir l'identifiant utilisateur (iduser), le nombre de refus qu'il a eu (nbDdeny) et la sévérité de la communauté à laquelle il appartient (epsilon). Le processus que déclenche la méthode *evaluationTrust* est le même que celui décrit dans l'algorithme de la section 1.1.1.

```java
public static double evaluationTrust(long iduser, long idSession, int nbDeny, double eplison) throws UnknownHostException {
    double currentDecayFactor,
            lastDecayFactor = 0,
            currentTrValue = 0,
            pastTrValue = 0;

    double[] trEvaluationReferences = {-5, -1, -0.5, -0.1, -0.05};

    TrustStore trStore = new TrustStore();

    // Récupération de la dernière valeur du DecayFactor depuis la base de donnée
    lastDecayFactor = trStore.getDecayHist(iduser);

    ExpDecay expDec = new ExpDecay();

    //Calcul de la valeur de trust pour la session qui vient de se terminée
    if (nbDeny > 5) {
        currentTrValue = eplison;
        }
    else {
        currentTrValue = (float) expDec.calculateCurrentTrustValue(lastDecayFactor, nbDeny);
        }

    //Récupération de la dernière valeur de trust depuis la base de donnée: TrustHistory
    //Calcul de la moyenne des trust values passées
    pastTrValue = expDec.calculateAverageOfTrustHistory(trStore.getTrustHistory(iduser));

    //Calcul du DecayFactor propre à la session qui vient de se terminer (qui sera appliqué pour la session suivante)
    currentDecayFactor = (float) expDec.lambdaComputing(pastTrValue, currentTrValue);

    //Recupération de l'indice du Decay le plus proche par rapport à l'évolution du DecayFactor (past DecayFactors consideration)
    lastDecayFactor = expDec.calculateNewDecayFactor(lastDecayFactor + currentDecayFactor, trEvaluationReferences);

    //Ajout de la valeur de trust de la session à l'historique du trust de l'utilisateur
    // add decay value to the dataBase
    trStore.updateTrustHistory(iduser, currentTrValue);
    trStore.updateDecayHist(iduser, lastDecayFactor);
    return currentTrValue;
}
```

Figure 5: Calcul du *trust*.

L'interaction avec la base de données utilisateurs MongoDB est gérée par la classe :
@Path("trustStre") public class ***TrustStore*** *{...}*

```
/*
Example JSON request
http://adresseIP/AuditService/resources/trustStre/PostPath

msg:
{
"UserName" : "Ahmed",
"Role" : "Admin"
}
*/
@POST
@Path("PostPath")
public Response insertUserInfos(String msg) throws UnknownHostException {
    String errorReport = null;
    int status = 0;

    MongoClient mongoClient = new MongoClient();
    db = mongoClient.getDB(nameDataBase);

    JSONObject obj = new JSONObject(msg);
    String usrName = obj.getString("UserName");
    String Role = obj.getString("Role");

    if (!usrName.equals("") && !Role.equals("")) {
        errorReport = "Ok";
        status = 200;

        DBCollection CollectionUsers = db.getCollection("CollectionUsers");

        BasicDBObject userDoc1 = new BasicDBObject("UserName", usrName)
                .append("Role", Role);

        CollectionUsers.insert(userDoc1);

    } else {
        errorReport = "Bad Request";
        status = 400;
    }

    return Response.status(status).entity(errorReport).build();

}
```

Figure 6: Initialisation de la base de données MongoDB des utilisateurs (ajout d'utilisateur "Ahmed")

1.1.3 Dépendances

- javaee-api-7.0
- json
- mongo-java-driver
- activation-1.1
- javax.mail-1.5.0