Chez les Barbus – Java & Sécurité

Romain Pelisse
Sustain Developer @Red Hat

François Le Droff
Achitecte @Adobe

"Chez les Barbus - Java & Sécurité", c'est le titre de la conférence donnée par François Le Droff et Romain Pelisse à l'occasion de Devoxx France 2015. Cet article propose d'en reprendre le contenu de manière plus didactique et plus adaptée à ce nouveau support.

Si la première partie a été publiée dans le GNU/Linux Magazine[1], voici maintenant sa seconde partie, qui discute des vertus de l'authentification à deux étapes, mais aussi des dangers liés à l'intégration continue sans politique de sécurité, et aussi au « Cloud ».

/// Résumé ///

Après avoir vu comment François a mis en place le SAML au sein de son application d'entreprise, nous allons maintenant étudier en détail la mise en place d'une authentification forte, et en l'occurrence à deux étapes, ainsi que les détails de son implémentation à l'aide de Oauth2.

/// Fin Résumé ///

Dans le premier article, François et Romain ont longuement discuté des problématiques de sécurité des applicatifs (Java/JEE ou autres), évoquant la méthode du « threat modeling » proposée par Microsoft et permettant d'évaluer sa sécurité applicative, et même d'identifier les zones à risque.

L'article a continué par une brève présentation du cas d'étude, une petite application interne faite par François, mais qui touche à beaucoup de données sensibles. Après avoir évoqué la relative inutilité des « firewalls », et discuté longuement des possibilités beaucoup plus pertinentes des « reverse proxy », la difficile question du chiffrement des données, de bout en bout, a elle aussi été abordée.

Avant de conclure, François et Romain ont aussi abordé la problématique de l'authentification et de sa mise en place au sein de l'applicatif utilisé pour le cas d'étude. C'est par ce point que nous commençons ici : voyons comment assurer une authentification forte – tout en justifiant aussi son absolue nécessité.

1 Authentification à deux étapes (2FA)

Romain : Du coup, j'imagine que, une fois arrivé là, tu as demandé à ton fournisseur d'identité (IDP) d'utiliser une authentification forte, une authentification à deux facteurs (souvent nommée 2FA pour « 2 Factors Authentication »).

François : Oui, en effet. Désormais, l'utilisateur devra non seulement me démontrer qu'il sait quelque chose mais aussi qu'il a quelque chose que mon IDP pourra valider : l'utilisateur fournira un secret (son mot de passe), mais aussi un jeton (« token »). Pour obtenir ce jeton, il devra utiliser un générateur installé sur un appareil référencé par mon IDP (une clef USB, une Yubi key ou une application sur son téléphone).

Romain : C'est parfait en terme de sécurité, mais en terme d'expérience utilisateur, c'est assez rébarbatif. À chaque nouvelle session, l'utilisateur devra saisir un mot de passe et en plus fournir un jeton.

François : De manière générale, c'est vrai que l'équilibre entre l'expérience utilisateur, le confort et la sécurité est difficile à trouver...

Romain : Et du coup, mettre en place ce genre de mécanisme, pourtant nécessaire, devient difficile à vendre.

François : Pour convaincre tes utilisateurs ou tes sponsors il suffit d'un peu de pédagogie, raconte quelques histoires.

Comme la comique mais authentique histoire de Paris Hilton : l'actrice/héritière était alors cliente de l'opérateur téléphonique T-Mobile. Comme beaucoup de fournisseurs de services en ligne, TMobile.com exigeait alors que ses utilisateurs répondent à une « question secrète » s'ils oublient leur mot de passe. Pour le compte de Hilton, la question secrète était très classique : "Quel est le nom de votre animal de compagnie préféré ?" En fournissant correctement la réponse, tout internaute pourrait donc modifier le mot de passe de Hilton et accéder librement à son compte. Or il suffisait d'une simple requête sur un moteur de recherche pour connaître le nom de ce chien de star…

Romain : Et ce qui devait arriver arriva**[2]**…

François : C'est déjà un argument très fort pour le 2FA, et si ça ne suffit pas, fais circuler l'article de Mat Honan**[3]**, journaliste nouvelle technologie chez Wired qui s'est fait vandaliser l'ensemble de ses comptes sociaux en ligne et effacer l'ensemble de ses archives digitales.

Romain: Oui, après avoir vu cette histoire, j'ai moi-même immédiatement basculé la plupart de mes comptes sur du 2FA, qu'il s'agisse de twitter, facebook, google, yahoo, github, paypal...

François : C'est effrayant de découvrir comment ce journaliste a pu voir toute sa vie numérique « détruite » aussi facilement, il témoigne ainsi dans un second article**[4]** : « Je n'ai pas activé l'authentification à deux facteurs sur Google ou Facebook. Je n'ai jamais mis en place de comptes de messagerie dédiés (et secrets) pour la gestion des mots de passe. Je prends ces mesures maintenant. Mais je sais aussi que, peu importe les mesures de sécurité que je prends, elles peuvent toutes être défectueuses par des facteurs indépendants de ma volonté.»

Romain : Note que ceci nous amène à un point important dont nos collègues et nos pairs dans notre industrie ne sont pas forcément conscients : en tant que développeurs, nous formons une cible privilégiée pour les pirates, et il est donc de notre responsabilité de protéger nos secrets.

François : En effet, nous avons nos profils sur linkedIn, où nous nous sommes qualifiés « full stack engineer » ou « expert sécurité ». Et pour peu que l'on travaille chez BNP (où 1 ms valent 100 millions d'euros), les accès aux secrets associés à notre emploi forment une cible de choix pour de nombreux pirates mal intentionnés.

Romain : Bref, si vous êtes convaincus, allez faire un tour sur le site de https://twofactorauth.org/, vous verrez quels services en ligne supporte le 2FA.

François : vous y constaterez qu'à l'heure où nous écrivons cet article, certains acteurs majeurs dans nos vies de développeurs (comme docker et jetbrains) ne supportent pas encore l'authentification forte. Vous noterez aussi, au passage, que le monde émergeant de l'« internet of things » (IoT) et les applicatifs santé font eux aussi globalement fi de la sécurité !

Romain : et de manière surprenante (et peu rassurante) il en est de même pour la plupart des sites de micro-paiement. Et en tant que « particulier », si vous n'avez peur de rien, vous pouvez aussi utiliser les services bancaires de citibank ou de la banque postale qui ne supportent pas le 2FA.

Les sites les plus sécurisés de ce point de vue, sont, en fait, les réseaux sociaux ! Ce qui signifie que si on peut accéder à vos données médicales, utiliser vos fonds pour jouer en bourse à votre place ou faire des micro-paiements en votre nom, il est beaucoup, beaucoup plus difficile de voler la photo de votre chat sur Facebook !

François : Ceci nous ramène directement à la problématique d'expérience utilisateur évoquée plus tôt, on notera que justement, ces sites de réseaux sociaux font tout pour offrir le meilleur confort d'utilisation à leurs utilisateurs.

Du coup, ils sont les premiers à chercher (et trouver) un bon équilibre entre l'expérience utilisateur et la sécurité ; et ceci en proposant néanmoins l'authentification à deux facteurs.

Romain : Preuve donc qu'on peut parfaitement y arriver. Et l'une des clés de cet équilibre est, par exemple, de permettre aux utilisateurs de définir leurs téléphones et navigateurs habituels comme étant « dignes de confiance », assurant ainsi que la session y soit maintenue, sans demander sans cesse de s'authentifier.

François : Figure-toi que c'est ce que nous avons mis en place pour notre application ! Voyons en détail comme nous avons choisi de l'implémenter.

2 SAML et OAuth

Romain : La dernière fois, tu nous avais montré comment tu avais mis en place l'authentification SAML. J'imagine qu'il va falloir repartir de là.

François : Tout à fait : une fois authentifié (avec SAML donc et en mode 2FA), nous provisionnons un client OAuth2 – uniquement valable pour notre utilisateur, et associé à son secret.

François : Va alors démarrer une petit « danse » OAuth2: une série de pas (de redirection) que vont effectuer « client » et « serveur » avec un minimum d'intervention humaine. *In fine*, notre client récupère un jeton (« token ») qui lui permet de s'authentifier, sans saisie de mot de passe et de secret.

Romain : OK, super, mais ce jeton, pour des raisons évidentes de sécurité, doit bien avoir une date de péremption, n'est-ce pas ? Et une fois atteinte, le jeton n'est plus valable.

François : Oui, bien sûr, mais c'est pour ça qu'on utilise aussi un jeton de rafraîchissement (« refresh token »). Ce jeton ne peut être utilisé qu'une seule fois et est remis avec le jeton d'authentification. Une fois ce dernier périmé, le client peut simplement retourner cet autre jeton pour renouveler son authentification (et récupérer un nouveau jeton de rafraîchissement au passage).

Romain : Mais, attends, du coup, si ton client – téléphone ou navigateur, est compromis, que se passe-t-il ?

François: Pas de problème, on peut alors révoquer, coté serveur, tous ces jetons, pour être sûr que les appareils ou clients compromis ne puissent pas être utilisés pour attaquer notre système. Mais avant de conclure sur cette partie, il est important de noter que SAML et OAuth2 n'étaient pas les seules solutions à notre disposition.

Romain: Oui, parce que même si OAuth2 colle parfaitement à tes besoins, il ne fournit pas de standard pour la signature de la requête.

François: C'est une bonne raison pour opter pour l'utilisation de « JWT token » qui supporte cette fonctionnalité.

Romain: Oui, mais au fait, pourquoi ne pas simplement utiliser OAuth1 ? C'est standard, non ? Et, ça comprend un standard de signature de requête…

François : Je crois oui, mais je t'avoue que quand j'ai regardé la spécification… Je n'ai rien compris !

Romain : OK, ce genre de chose arrive, mais du coup, tu n'as pas jeté un œil au code pour y voir plus clair ?

François : Si.

Romain : Et alors ?

François : Rien compris non plus.

Romain : OK, je pense que là, tout est dit sur ce sujet :) !

3 Extension du périmètre de la lutte - CI & Cloud

Romain: Jusqu'à maintenant, quand on pense sécurité des applicatifs, on a naturellement tendance à penser aux plates-formes de production, et à délaisser, voire à être laxiste avec les environnements de développement et les plates-formes d'intégration continue.

François : c'est mal.

Romain : Et si en plus tu déploies dans le « Cloud », ta chaîne de déploiement continue doit être irréprochable, ou tout un nouvel univers de failles et d'exploitation s'ouvre aux attaquants, dans un domaine, où les experts de la sécurité sont très loin d'avoir évangélisé et diffusé les bonnes pratiques associées. Voyons un peu toutes ces problématiques en détail.

a) Gestion des secrets

François: Commençons par regarder notre code, contient-il des secrets ?

Romain : C'est un excellent point de départ. Tu as bien raison de vérifier ceci.

François : Oui, on y a pensé, en effet, mais est-ce que c'est vraiment important ? À part pour la production bien-sûr.

Romain : Méfie-toi, car même si les données ne sont pas sensibles, les secrets peuvent quand même donner accès à de nombreux systèmes. Ainsi, nombreux sont les utilisateurs d'Amazon qui poussent leur

clef sur github. Immédiatement repérées, leurs clés peuvent été utilisées par des tiers pour générer des instances (et miner du bitcoin par exemple), et avec une facture de plusieurs milliers de dollars**[5]** !

François: Clairement il faut faire attention à ne pas exposer ses secrets, mais il faut aussi pouvoir gérer leur cycle de vie, assurer qu'ils soient proprement déployés mais aussi accessibles aux personnes autorisées.

Romain : Et tout ceci n'est pas franchement simple, surtout avec l'exigence d'aujourd'hui en terme d'automatisation des déploiements d'applicatifs ou de systèmes. Une solution François ?

François: Dans le cadre de notre cas d'étude, j'avais utilisé Chef pour l'automatisation de mon infrastructure. Chef est un Framework open source écrit en Ruby. Il permet de déployer facilement des serveurs et des applications à grande échelle sur des infrastructures d'entreprise. Il existe un bel écosystème de recettes et bibliothèques autour de cette solution, l'une d'elles s'appelle Chef-vault.

Romain : Chef-vault ? C'est intéressant, dis m'en un peu plus.

François: Chef-vault**[6]** est un outil de gestion des secrets open-source et communautaires. Il s'appuie sur le système de chiffrement du « framework » Chef mais le pousse plus loin: il permet d'éviter l'utilisation d'une clé partagée par tous les utilisateurs et les infrastructures**[7]**.

Romain : Et si j'ai bien compris, Chef-vault s'appuie sur une liste blanche (« white list ») ? Mais comment tu montes en charge avec un mécanisme pareil ?

François: Bien vu, Chef-vault est assez bien limité lorsqu'il s'agit d'auto-scaling ou d'auto-guérison. Noah Kantrowitz le résume bien dans son blog**[8]** : il y propose d'ailleurs quelques alternatives. Mais cette limitation de Chef n'en était pas une pour moi, puisque mon application interne avait une charge déterministe.

Romain : Rappelons enfin également que : que nous utilisions Chef et Chef-vault ou pas, il faudra bien sécuriser notre serveur de secrets.

François: comme tous les autres éléments de la machinerie qui construisent et déploient les applicatifs qui tournent sur nos serveurs.

Jenkins

François : Donc là, on est bon sur les secrets applicatifs, mais *quid* de la sécurité du serveur d'intégration continue ?

Romain : C'est aussi une source de beaucoup de failles – car peu de gens se soucient de ces serveurs qui pourtant offrent d'utiles, mais aussi très dangereuses fonctionnalités.

François : Par exemple, sur Jenkins, tu peux créer des tâches qui exécutent des scripts Shell, directement sur le système, ou sur un nœud esclave !

Romain : Et comme souvent, on ne prend pas soin de mettre en place une authentification sur le serveur, on ne sait même pas qui a créé le script, ni comment !

François : Il est donc crucial de s'assurer que le serveur suive les bonnes pratiques de sécurité, car il peut non seulement être détourné, mais aussi offrir un accès à beaucoup de points sensibles de notre application.

Romain : Comme son code source, ou même simplement les secrets que nous venons d'évoquer. Si on a accès au serveur d'intégration continue, on a accès à tout ça ! Il faut donc mettre en place, comme l'avons déjà évoqué**[1]** une authentification forte, un chiffrement des flux de données, et une journalisation appropriée des événements pour permettre un audit de sécurité.

François: Même constat pour nos serveurs d'artefacts.

Romain : Justement, parlons un peu d'eux.

Nexus, Satellite et les gestionnaires d'artefacts

François: Depuis nos serveurs d'artefacts, nous collectons des bibliothèques/gem ruby, des fichiers javascripts, des paquets/npm node.js, des archives/jar java, des paquets/rpms linux et d'autres archives propriétaires: la collection complète des binaires et fichiers qui seront déployés sur nos serveurs. Chacun de ces binaires, chacune de ces archives, chacun de ces builds peuvent être un vecteur de « malware » et autre cheval de Troie. Leurs intégrités doivent être contrôlées de façon continue.

Romain : Une solution est de mettre en place des serveurs mandataires (audités et sécurisés), des « proxy »

tels que Nexus ou Artifactory, pour les dépendances Java. Node ou Ruby, ou même Red Hat Satellite, pour les dépendances systèmes (paquets RPMs). Les équipes de production et de sécurité peuvent ensuite vérifier les binaires utilisés dans l'assemblage des logiciels – et ceci sans interférer avec les développements.

François : Comment peuvent-ils vérifier qu'un fichier binaire n'a pas été corrompu ?

Romain : En vérifiant que les signatures correspondent à celles publiées. C'est fastidieux à faire à la main, mais ce genre d'outil s'en charge automatiquement et rapporte les éventuelles infractions détectées.

Le grand méchant Cloud

Romain: Ton application finie et maintenant à la pointe de la sécurité, j'imagine que tu vas maintenant « voir si tu n'es pas un homme du XXIème siècle », comme dirait Audiard, et la déployer sur le « Cloud », non ?

François: Avec mes problématiques de sécurité, je ne sais pas si c'est une bonne idée.

Romain: Non, au contraire ! L'architecture d'un « Cloud » (comme Amazon, Azure, CleverCloud…) te force en effet à réfléchir à la sécurité dans ce contexte, mais t'apporte aussi beaucoup dans ce domaine ! Par exemple, combien de systèmes, pourtant bien sécurisés lors de la mise en place, ne sont par la suite plus maintenus, ni mis à jour. Combien d'applications « oubliées », sont ainsi exposées au monde extérieur et offrent des failles connues pour le pirate informatique ?

François: Chez nous ? Aucune bien sûr !

Romain : Bien sûr ! Quoi qu'il en soit, sur le « Cloud », les systèmes sont souvent automatiquement rafraîchis, « patchés », mis à jour. Et si on fait bien les choses, les machines virtuelles ne sont utilisées que quelques jours, voire quelques heures, avant d'être dé-commissionnées et reconstruites avec une version mise à jour du système – contenant ainsi tous les éventuels correctifs de sécurité publiés depuis la précédente version. Comme le fait Netflix !

François: C'est tout l'intérêt de l'approche décrite comme « pet versus cattle » (animaux domestiques versus troupeaux), qui peut se résumer ainsi : « Ne traitez plus vos serveurs à la main, cessez de passer votre temps à les cajoler, traitez-les plutôt comme du bétail : au moindre signe de fatigue ou de maladie, tuez-les et remplacez-les. » En effet, il faut une famille entière pour éduquer un animal domestique et juste quelques cowboys pour gérer un troupeau (cette métaphore ne plaira pas aux végétarien et autres vegans, désolé).

Romain: Tu peux aller encore plus loin. Je viens de mentionner Netflix ; chez Netflix ils ne laissent jamais tourner leurs instances, sur Amazon, pas plus de quelques jours. Si tu as déjà travaillé avec l'infrastructure de Amazon, tu sais que : bien que le service soit d'excellente qualité, il n'est pas fiable à 100%. De manière générale on n'est jamais à l'abri d'une défaillance de logiciel – même s'il s'agit de son propre logiciel.

François : Surtout que pour gérer les défaillances, il faut s'y prendre en amont, dans le code. Être sûr d'avoir géré toutes les pannes et avaries possibles.

Romain : Et c'est pour ça que Netflix a développé un outil nommé Chaos Monkey**[9]**, qui sert à tester que le système produit a bien géré tous les cas possibles, en … causant des avaries !

François : C'est super pour les phases de qualification !

Romain : Mais, dans le cas de Netflix, il le laisse même s'exécuter en production !

François : Impressionnant en effet… Quand j'y pense, en fait, il serait génial d'avoir exactement la même chose pour la sécurité. Une sorte de « hacky monkey » qui forcerait les développeurs à penser, dès le début, à la sécurité.

Jouer au pompier

Romain : Penser à la sécurité dès le début c'est important, mais ce n'est pas tout. En fait, aujourd'hui, quand on voit les attaques, de type déni de services ou autres, ou encore le vol de données, on réalise qu'il est tout aussi important de savoir comment réagir.

François : C'est vrai. La question n'est plus de savoir si on sera attaqué, mais comment le détecter, réagir et assurer que si l'attaque réussit, le système ou les données ne peuvent pas être compromis outre mesure.

Romain : Et c'est possible ! Regarde comment Github a superbement survécu à plusieurs attaques de type déni de service distribué (DDoS), tout en communiquant dessus, sans complexe.

François : À l'inverse, le cas de la base de données piratées de (Site de rencontre adultère), dont seuls les

mots de passe étaient chiffrés, (comme nous l'avons évoqué lors du premier article) est un parfait contre-exemple. Ils auraient mieux fait, avec le recul, de chiffrer les données de l'utilisateur, pas seulement son mot de passe.

Romain : On peut même aller plus loin dans ce domaine, par exemple en utilisant un module matériel de sécurité, un HSM (« Hardware Security Module »**[10]**).

François : En fait, en conclusion, et pour faire une métaphore facile à retenir, quand on pense à la sécurité, il faut penser comme un pompier. Les pompiers ne font pas que faire briller leur beau camion, ils se préparent aussi à affronter et arrêter des incendies – dans notre cas, il faut donc être prêt à faire face à une attaque.

Romain : nous donnerons toutes leurs chances aux pompiers, en équipant notre système de détecteurs de fumée (comme des systèmes de détection d'intrusion) mais aussi de portes coupe-feu (avec des outils comme SELinux ou le Gestionnaire de Sécurité de la JVM (« security manager »), pour s'assurer que l'incendie ne pourra pas se propager...

François : Comment marchent ces derniers ?

Romain : Les deux outils suivent le même principe. Ils vérifient que les activités du système, dans le cas de SELinux, et de la machine virtuelle Java, dans le cas du « security manager », soient conforme aux attentes. C'est-à-dire conformes à une politique qui définit quelles utilisations des ressources peuvent être faites par tel ou tel processus ou programme. Ainsi, si ton serveur HTTP – Nginx ou Apache – se met soudainement à vouloir écrire dans le fichier /etc/password, SELinux va lui interdire.

François : Et pareil avec Java, j'imagine ? Si mon applicatif « web » veut soudainement ouvrir une connexion directe vers mon serveur LDAP ou ma base de données, le gestionnaire de sécurité de la JVM le bloque ?

Romain : Exactement. Du coup, le pirate a certes compromis ton système, mais sa marge de manœuvre est très réduite.

François: Oui, mais si l'exploit se base sur une "activité" normale de l'applicatif. Par exemple si l'on modifie le contenu d'un fichier de données légitime de l'application, là, on n'est plus protégé.

Romain: Non, en effet, et c'est là encore qu'il est essentiel de penser à la sécurité en développant. Dans ce cas précis, il est nécessaire de développer un mécanisme de sécurité supplémentaire.

François: Par exemple, faire un contrôle de cohérence (« checksum ») sur le fichier ?

Romain: Bonne idée, en effet ! Je te félicite, après être devenu un vrai « DevOp », tu viens de devenir un « DevSec » !

Conclusion

Nous avons démontré dans cet article l'absolue nécessité de l'utilisation systématique d'une authentification forte et sa relative facilité de mise en place. Ne vous en privez pas ! Votre compagnie, et peut-être vous-même, pourriez être amenés à le regretter.

Nous avons vu ensuite comment le périmètre de la lutte en sécurité s'est largement élargi avec l'arrivée de l'intégration continue et du « cloud ». Il ne suffit plus aujourd'hui d'assurer la sécurité du seul applicatif en production, mais aussi de s'assurer que toute la chaîne de production liée à ce dernier n'est pas compromise, tout autant que sa plate-forme d'exécution...

Bref, comme nous l'avons évoqué tout au long de ces deux articles, il est essentiel de penser à la sécurité, de la conception à la mise en production de l'applicatif, mais aussi au-delà car, rappelez-vous, un bon pompier s'entraîne aussi à affronter des incendies...

Références

[1] Chez les Barbus – Java & Sécurité pt 1 *GNU/Linux Magazine* n°196 (Septembre 2016), p.58: **http://www.gnulinuxmag.com/creez-votre-premiere-intelligence-artificielle/**

[2] How Paris (Hilton) Got Hacked? **http://archive.oreilly.com/pub/a/mac/2005/01/01/paris.html**

[3] Mat Honan Hacked (1) - **https://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/**

[4] Mat Honan Hacked (2) - **https://www.wired.com/2012/08/mat-honan-data-recovery/**

[5] My $500 Cloud Security Screwup - **https://securosis.com/blog/my-500-cloud-security-screwup**

[6] Chef Vault - **https://github.com/chef/chef-vault/**

[7] What can Chef Vault do for you - **https://blog.chef.io/2016/01/21/chef-vault-what-is-it-and-what-can-it-do-for-you/**

[8] Chef et la gestion des secrets - **https://coderanger.net/chef-secrets**

[9] Le « Chaos Monkey » de Netflix - **https://medium.com/netflix-techblog**

[10] Hardware Security Module (HSM) - **https://en.wikipedia.org/wiki/Hardware_security_module**