Sécurité ISI - Devoir 2
CVE-2017-12617

Yohan GOUZERH, Yassir IDHBI, Ke LI

Contents

1 Introduction

1.1 Introduction

La faille de sécurité CVE-2017-12617 qu'on va détailler concerne le serveur web qui permet la conteneurisation des servlets et des applications web écrites en Java EE et en JSP. La faille permet d'uploader un fichier JSP au serveur via une requête HTTP qu'on peut par la suite y accéder pour exécuter le code écrit dessus.

1.2 Notions techniques

1.2.1 Principes de bases de HTTP

HTTP ou Hypertext Transfer Protocol est un protocole de la couche applicative. Inventé par Tim Bernes-Lee (Prix Turing 2016), il est à la base du World Wide Web, et est utilisé pour l'échange de page HTML en mode client-serveur.

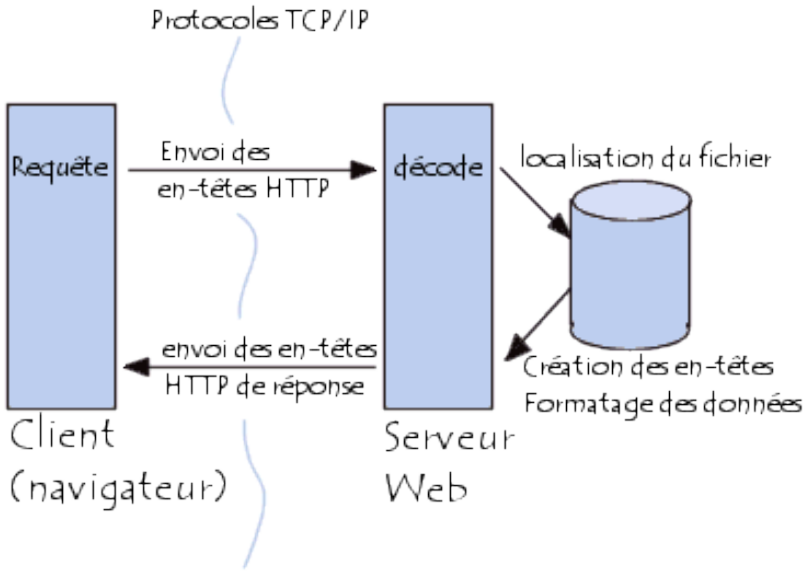

Figure 1: Envoi d'une requête HTTP à un serveur Web

Le protocole est assez simple. Le client envoie une requête HTTP au client avec un certain en-tête, et le serveur lui répond avec d'une partie la réponse, et de l'autre un en-tête de réponse.

Exemple d'en-tête de requête, cherchant à récupérer le fichier /index.html sur le serveur www.example.com en utilisant la version 1.1 de HTTP :

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Exemple de réponse, renvoyant la page web correspondante, ainsi que différentes méta-données, dont le code de retour http[1] (200 = Succès de la requête):

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
```

[1]La liste des codes de retour http peut être trouvée ici : `https://developer.mozilla.org/fr/docs/Web/HTTP/Status`

```
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

1.2.2 Les méthodes HTTP

Il est possible pour le client d'intéragir avec le serveur de différents moyens, en spécifiant dans l'en-tête client différentes méthodes[2].

Exemples de différentes méthodes :

GET : Méthode utilisée pour récupérer une ressource.

POST : Méthode utilisée pour envoyer des données au serveur, que celui-ci traitera.

HEAD : Comme GET, mais sert à récupérer juste l'en-tête, sans la réponse.

PUT : Méthode utilisée pour envoyer une ressource (fichier) à une certaine URI[3].

Exemple de requête PUT[4]:

```
PUT /new.html HTTP/1.1
Host: example.com
Content-type: text/html
Content-length: 16

<p>New File</p>
```

Cette requête demande à créé le fichier /new.html contenant le texte html `<p>New File</p>`

La réponse donnée par le serveur est par exemple la suivante :

```
HTTP/1.1 201 Created
Content-Location: /new.html
```

Le serveur doit renvoyer le code `201` si la ressource a bien été créée, et `200` ou `204` si la ressource existait déjà et qu'elle a été modifiée.

1.2.3 Apache Tomcat

Apache Tomcat est un conteneur de servlets[5] permettant par exemple de gérer le cycle de vie des servlets, gère le routing, et l'accès aux différentes servlets, permettant ainsi de faire tourner un serveur HTTP complet en Java.

[2] La liste des différentes méthodes peut-être trouvée ici : `https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thode`

[3] **URI :** Uniform Resource Identifier. Une URI est un identifiant de ressource sous la forme d'une chaîne de caractère. C'est un concept plus global que le terme URL (pour le web seulement), et regroupe par exemple les ISBN ou les QR Codes (GSI URI)

[4] Tiré de : `https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thode/PUT`

[5] Un servlet est une classe Java permettant de recevoir une requête http et de renvoyer une réponse http

1.3 Enjeux

La vulnérabilité vise à exploiter un bug lors d'une requête HTTP PUT qui permet de contourner la vérification des extensions de fichier et d'uploader le fichier JSP que nous souhaitons par le biais des paramètres de la requête PUT en question, par exemple:
Si nous avons un serveur Apache Tomcat vulnérable qui accepte les requêtes PUT et lancé en:
`http://localhost:8080/`
Nous pourrions faire un upload en envoyant une requête PUT avec le fichier que nous désirons uploader en paramètre et en contournant l'extension juste en ajoutant un caractère "/' après l'extension comme ceci:
`PUT http://localhost:8080/exploit.jsp/`

2 Analyse de la faille

La faille est présente lorsque le paramètre `readonly` dans la configuration du serveur est mise à `false`, afin de permettre les méthodes PUT et DELETE.

```
<init-param>
    <param-name>readonly</param-name>
    <param-value>false</param-value>
</init-param>
```

En principe, un serveur Apache Tomcat qui accepte les requêtes PUT ne doit permettre seulement l'upload de fichiers ne représentent pas de risques pour le serveur (html, jpg, png ...).

Avec cette faille, il est possible d'upload des fichiers malveillants comme des JSP, et ainsi faire exécuter du code arbitraire par le serveur.

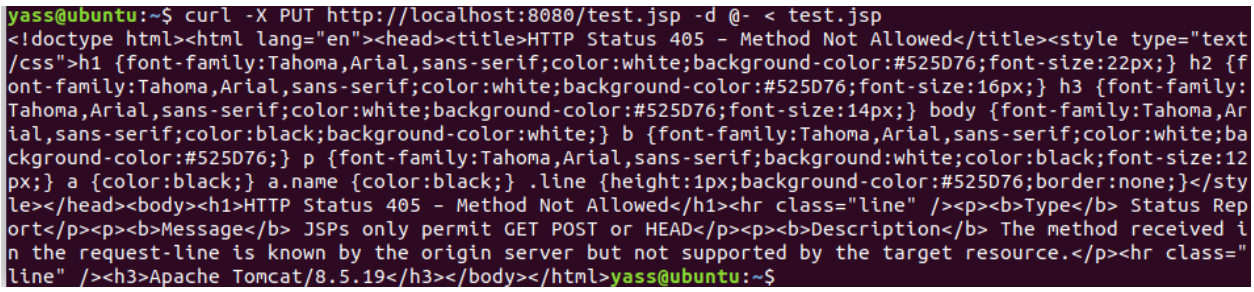

Figure 2: Envoi d'une requête PUT d'un fichier JSP

Sauf que les mesures de sécurité faites par les développeurs n'étaient pas très efficaces et il s'est avéré qu'il était assez facile de contourner les vérifications des fichiers en paramètre d'une requête PUT juste en ajoutant un caractère '/' après l'extension du fichier.

3 Exploitation de la faille grâce à notre PoC

Notre PoC consiste en une image Docker contenant un serveur Tomcat qui accepte les requêtes HTTP PUT et DELETE (puisque ce n'est pas accepté par défaut). Pour que le serveur accepte la requête PUT il faut modifier le fichier `web.xml` comme ceci:

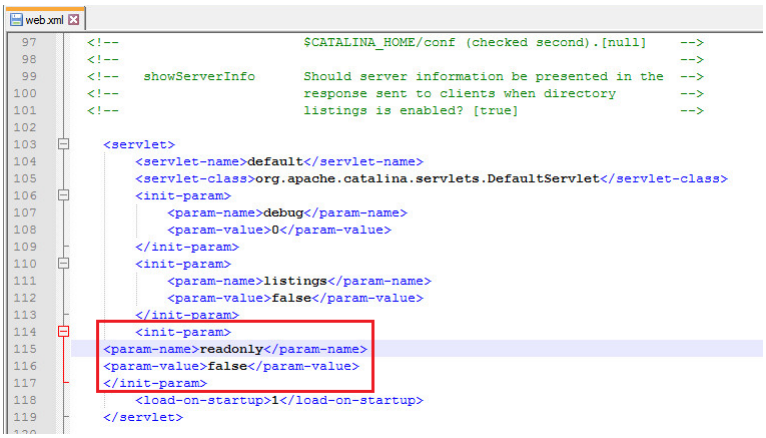

Figure 3: Paramètres à modifier

Notre PoC va vous permettre d'uploader une JSP particulière sur un serveur Tomcat présentant cette vulnérabilité. Cette JSP contient un shell, permettant de prendre la main sur le serveur sans authentification, et ce depuis votre navigateur.

4 Analyse post-mortem

4.0.1 Pourquoi cette faille a-t-elle eu lieu ?

Les webapp class loaders Par défaut, il n'est pas possible d'exécuter des classes java dynamiquement. De plus, les seuls classes pouvant être exécutée doivent être placés sur Tomcat dans le dossier WEB_INF ou META_INF.

Une fonctionnalité introduite par Fabrizio Giustina en décembre 2006 permet justement de définir des classes loaders[6], c'est à dire des classes permettant de charger du code dynamiquement dans la JVM.De plus, il est possible de les définir en dehors du WEB_INF.

Ce mécanisme permet de simplifier grandement le développement, en évitant d'avoir à chaque débogage à redéployer les jars dans le dossier WEB_INF par exemple.

Vérification initiale
Pour éviter les failles de sécurité, il est vérifié que les fichiers ne sortent pas du répertoires configuré par le développeur.

La validation est la suivante pour pouvoir exécuter un fichier : le chemin du fichier est normalisé, c'est à dire que sont transformé les . et .. en chemin complets. Si l'appel à l'API new File("cheminFichierNormalisé") renvoie un fichier, alors ce fichier est exécuté.

Bogue dans la vérification
Il est possible ici de bypasser cette validation, permettant ainsi d'exécuter une JSP situé dans un dossier de d'upload d'un utilisateur.

[6]Voir `https://bz.apache.org/bugzilla/show_bug.cgi?id=41260` pour plus d'informations sur ce mécanisme de class loaders personnalisés

Une problème de compréhension de l'API File de Java est à l'origine de cette faille. L'API réalise une normalisation automatique modifiant le chemin d'un fichier, comportement non pris en compte par les développeurs.

Lors de la recherche d'un fichier, un appel à (new File(name)).getPath() sur un fichier finissant par un "/" va supprimer ce "/". Par exemple, (new File("/file.jsp/")).getPath() retournera new File("/file.jsp")).getPath(), et récupérera le fichier correspondant, qui sera ainsi exécuté.

4.0.2 Comment cette faille a été corrigée ?

Tout d'abord, les développeurs se sont interrogés sur la possible suppression du readonly. Dans la documentation, il est même mentionné de ne jamais le mettre à true.

Simplement, ce paramètre est nécessaire pour les serveurs utilisant WebDAV, et il n'est pas possible de le supprimer et de le hardcoder à false.

Peter Stöckli 2017-09-22 08:02:14 UTC Comment 7

Isn't the mere existence of the readonly parameter also part of the problem?

https://tomcat.apache.org/tomcat-7.0-doc/default-servlet.html
It is currently documented as "Is this context "read only", so HTTP commands like PUT and DELETE are rejected? [true]"

But it holds more "surprises". IMHO this parameter should NEVER be set to false. Maybe it can be removed or the documentation of this parameter can be improved?

Remy Maucherat 2017-09-22 09:20:33 UTC Comment 8

(In reply to Peter Stöckli from comment #7)
> Isn't the mere existence of the readonly parameter also part of the problem?
>
> https://tomcat.apache.org/tomcat-7.0-doc/default-servlet.html
> It is currently documented as "Is this context "read only", so HTTP commands
> like PUT and DELETE are rejected? [true]"
>
> But it holds more "surprises". IMHO this parameter should NEVER be set to
> false. Maybe it can be removed or the documentation of this parameter can be
> improved?

Have you ever heard of WebDAV ? Obviously if we were writing Tomcat today, we would never bother implementing it. Also obviously, nobody running a public server should enable it, secured or not. But it's not going to be removed either.

Mark Thomas 2017-09-22 09:23:09 UTC Comment 9

Indeed. Lots of folks run Tomcat with WebDAV on internal sites. Hard-coding readonly to true is simply not an option.

Regarding better documentation, patches welcome.

Peter Stöckli 2017-09-22 11:20:52 UTC Comment 10

Created attachment 35361 [details]
proposal to improve doc of the readonly flag

First of all: your work is greatly appreciated!
And I didn't know that Tomcat is also widely used as WebDAV server. So it makes sense to keep that option.

Attached is a patch that could help improve the documentation of the readonly flag.

Figure 4: Discussion sur la raison d'être du readonly

Notes sur WebDAV :

WebDAV, ou Web Distributed Authoring and Versioning, est un protocole d'échanges de fichiers permettant de réaliser facilement un Cloud privé. C'est un protocole qui fut beaucoup utilisé en entreprise, du fait de sa simplicité d'utilisation pour l'utilisateur (accès aux

fichiers grâce à un simple lecteur réseau), et le fait qu'il soit facile à configurer (utilisation du port 80 ou 443, permettant). Il présente cependant des limitations[7] : caractères spéciaux dans les noms des fichiers interdits, fichiers devant être plus petits que 50MB, failles de sécurité (Voir `https://www.networkworld.com/article/2202909/network-security/-webdav-is-bad---says-security-researcher.html` pour plus de détails),... Il est aujourd'hui remplacé par d'autres solutions, comme SMB (`https://fr.wikipedia.org/wiki/Server_Message_Block`) par exemple.

La documentation sur le paramètre readonly a cependant été étoffée pour permettre à l'utilisateur de prendre la pleine mesure des conséquences d'une mise à false du paramètre.

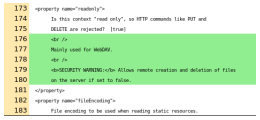

Figure 5: Modification de la documentation

La validation du chemin de fichiers a été étendue, et les vérification se font maintenant sur les chemins absolus, mais aussi sur les chemins canoniques.

4.0.3 Résultats de la solution

Ce correctif semble fonctionner, aucune CVE relative à du Remote Code Execution n'a été trouvée sur Tomcat dans les produits corrigés, depuis cette CVE de décembre 2017.

5 Les conséquences de cette faille

5.1 Agents concernés

Les agents vulnérables de cette failles sont:

- Apache Tomcat, versions 9.0.0.M1 à 9.0.0
- Apache Tomcat, versions 8.5.0 à 8.5.22
- Apache Tomcat, versions 8.0.0.RC1 à 8.0.46
- Apache Tomcat, versions 7.0.0 à 7.0.81

5.2 Contre mesure

Le patch pour la faille a été introduit dans les versions Apache Tomcat 9.0.1, 8.5.23, 8.0.47, 7.0.82 et donc il suffit de mettre à jour son serveur pour corriger la faille.

Il est de plus conseillé aux administrateurs ne pouvant réaliser un upgrade dans l'immédiat, de désactiver le paramètre readonly, ou d'effectuer des vérifications du format des fichiers envoyés dans la partie applicative.

5.3 Menaces possibles

Lors d'une attaque de ce type, la personne malveillante pourrait uploader plusieurs fichiers JSP et les exécuter par la suite ce qui peut entraîner toute sorte de menace sur le serveur. Le code sera par la suite interprété par Apache Tomcat et peut exécuter plusieurs instructions nocives au serveur qui héberge l'application web en question.

[7] Voir `https://docs.kentico.com/k10/managing-website-content/configuring-the-environment-for-content-editors/configuring-webdav/webdav-requirements-and-limitations` pour plus d'informations

L'exploitation la plus dangereuse pour cette faille (ce que nous avons réalisé dans notre PoC) c'est la possibilité pour l'attaquant sera capable de créer un shell en JSP et l'uploader au serveur afin de pouvoir lancer n'importe quelle commande à distance en tant qu'utilisateur apache.

La prise main sur un shell qui va lui permettre de lancer n'importe quel commande, et si jamais le serveur Apache Tomcat est lancé par l'utilisateur 'root' ou si l'attaquant réussit une élévation de privilèges, l'attaquant aura l'accès privilégié à la machine, ce qui va lui permettre de tenter plusieurs manipulations malicieuses dans la machine distante et sur le réseau interne de l'entreprise.

Sans même être root, cela pourrait entraîner différentes menaces importantes pour l'entreprise, comme de la fuite d'information, du déni de service (mise hors service du serveur), corruption des données, manipulation de l'utilisateur par création de pages malveillantes,...

6 Extrait de la Politique de Sécurité du Système d'Information

6.1 Scénario d'exploitation possible

Un des scénario d'utilisations les plus dangereux pour l'entreprise et qui a le plus chances d'arriver est le suivant : la divulgation d'informations confidentielles.

Prenons le scénario suivant : l'entreprise pharmaceutique **PharmaForGood** décide d'utiliser un système de partage de fichiers, afin de permettre à ses employés de collaborer efficacement sur ses nouveaux produits, et notamment sur une nouvelles solutions qui pourront sauver des vies à moindre frais.

Elle décide pour cela de mettre en place un service permettant à chaque employé de mettre en ligne ses fichiers pour que les autres puissent les voir et les modifier.

Elle choisit pour cela d'un serveur WebDAV sous Tomcat 8.5.0, solution facile à utiliser pour ses utilisateurs, et qui s'interface bien à son environnement Windows.
La réception et l'envoi des données se fait ici à l'aide de servlets.

Pour être disponible lors des déplacements, le serveur web est accessible depuis l'extérieur et possède une adresse ip publique et un nom de domaine associé.

Afin de sécuriser l'accès au données, une solution d'authentification à deux facteurs couramment utilisée avec WebDAV est mise en place : WIKID[8]
Du côté de notre attaquant, la société **Evil Corp**, multinationale au business agressive, se rend compte grâce à son système de crawling régulier, qu'un nouveau service est en ligne chez **PharmaForGood** à l'adresse http://stockage.pharmaforgood.com/index.jsp. Après une rapide analyse du service[9], elle se rend compte que c'est un serveur Tomcat qui fournit la page web. Elle tente une série d'exploits metasploit automatisés contre le serveur, et se rend compte que notre l'exploit 43008 fonctionne [10]. Elle découvre que cet exploit fait référence à la CVE-2017-12617.

Après un rapide ,elle décide d'utiliser notre exploit pour executer un shell à l'adresse http://stockage.pharmaforgood.com lui permettant d'avoir accès aux différents dossiers présents sur le serveur depuis une interface web difficilement découvrable.

Elle découvre ainsi le dossier webdav/projectSecret, contenant une liste de documents words, normalement accessibles seulement par double authentification.

[8] `https://www.wikidsystems.com/`
[9] L'analyse peut être aussi bien avec du scan de port ou tout simplement l'extension Wappalyzer sur Firefox `https://www.wappalyzer.com/`
[10] Voir `https://www.exploit-db.com/exploits/43008`

Trois semaines plus tard, l'entreprise **Evil Corp** mais sur le marché un vaccin révolutionnaire contre le paludisme, à un tarif difficilement atteignable pour les classes les plus pauvres.

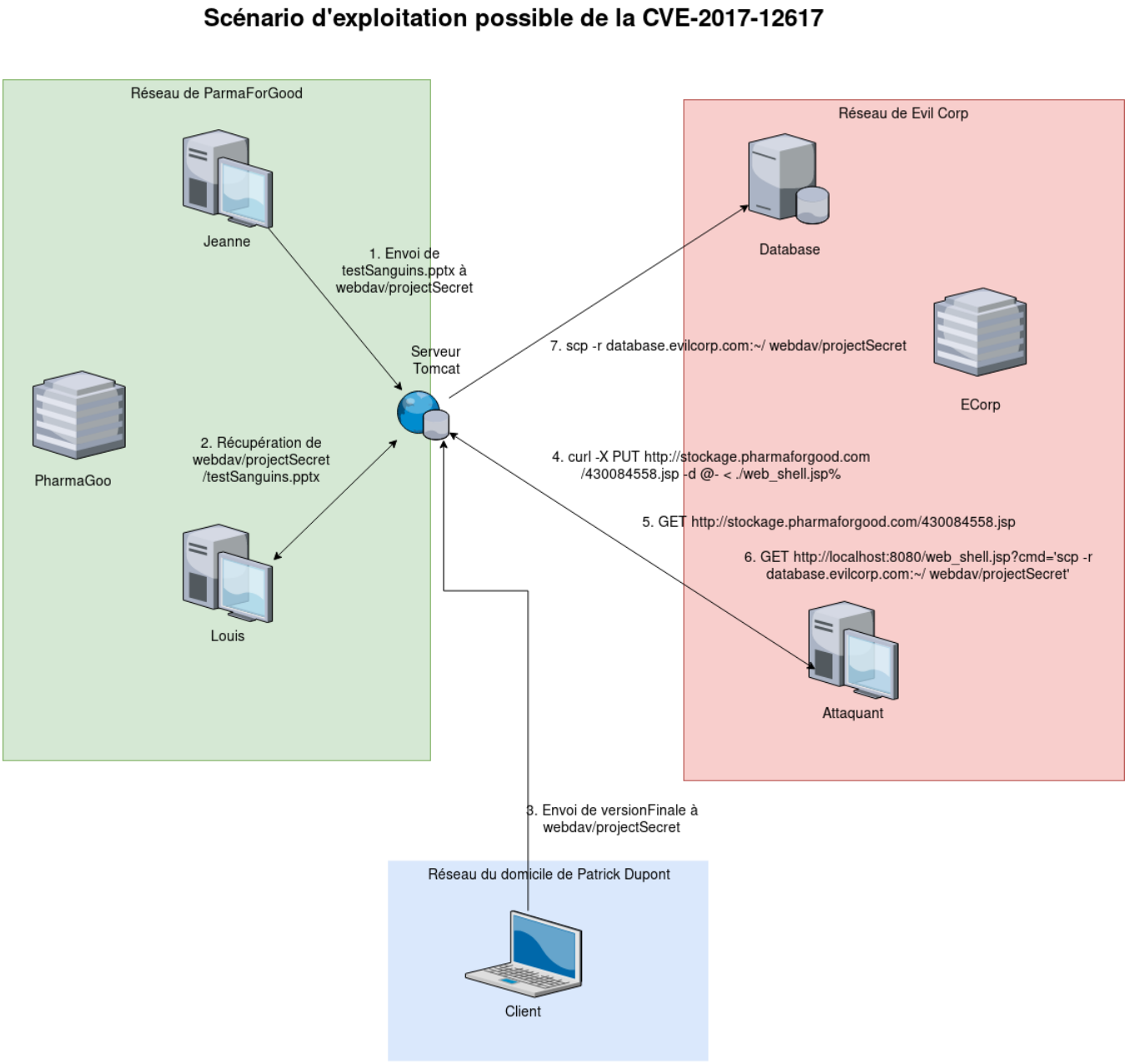

Figure 6: Architecture du scénario d'attaque

Déroulement de l'attaque :

1. Envoi d'un fichier sur le cloud depuis le réseau local de l'entreprise.
2. Récupération d'un fichier sur le cloud depuis le réseau local de l'entreprise.
3. Envoi d'un fichier sur le cloud depuis l'extérieur de l'entreprise.
4. Utilisation de la faille : envoi du web shell sur le serveur.
5. Récupération du web shell sur le navigateur de l'attaquant.

6. Envoi de la commande de copie des fichiers au serveur

7. Execution de la commande de copie du serveur, qui va copier les données du projet secret sur un serveur distant ne demandant pas d'authentification.

(Dans le cas d'une réelle attaque, les machines seront des machines jetables situées dans un autre réseau, et il faudra veiller à cacher ses traces).

Ici nous nous rendons compte, avec ce cas extrême, de la facilité avec une laquelle un attaquant extérieur, qui n'a pas besoin de connaissance avancés en sécurité peut récupérer des informations très confidentielles sur une entreprise implantant la solution Tomcat + WebDav, solution encore beaucoup utilisée en entreprise aujourd'hui. De plus la solution d'authentification très robuste mise en place n'a ici servi strictement à rien.

6.2 Criticité de la faille

Dans un contexte d'entreprise, la faille permettra à l'attaquant de lancer du code à distance dans la machine vulnérable.

7 Extrait de la Politique de Sécurité du Système d'Information

7.1 Actions préventives

7.1.1 Architecture du SI

- L'architecture du SI ainsi que la brique logicielle utilisée devront être mis à jour régulièrement afin de réduire les failles logiciel qui peuvent être exploitées.
- Les administrateurs devront éviter de lancer le serveur Apache Tomcat en mode 'root'.
- Les serveurs Web devront être isolés dans des conteneurs afin de limiter l'accès aux fichiers des serveurs de l'entreprise en cas d'attaque.

7.1.2 Développeurs des applications web

Les développeurs devraient respecter quelques bonnes pratiques au niveau de la sécurité de l'application notamment:

- Éviter les requêtes PUT et DELETE, et donc laisser le paramètre readonly à true ce qui est fortement recommandé par l'équipe de sécurité chez Apache Tomcat.
- Désactiver les messages d'erreurs Tomcat puisque l'attaquant pourrait savoir la version exacte du serveur utilisée et puis chercher la faille adéquate pour qu'il puisse l'exploiter par la suite.
- S'il n'est pas possible de désactiver le readonly, une bonne pratique est de bloquer au niveau applicatif le chargement de fichiers executables.

7.1.3 Serveurs

- Les serveurs devront être répliqués afin de faciliter la migration de l'un à l'autre afin de faire une maintenance pour les machines victimes à la suite d'une attaque.
- Les identifiants des différents services utilisés par l'application (Base de donnée, identifiants cloud ...) ne devront pas être stockés en clair.
- Le serveur doit avoir son propre utilisateur associé, afin de restreindre les actions sur le système possible.

Ressources humaines Signaler tout comportement inhabituelle du service aux administrateurs systèmes afin qu'ils puissent vérifier toute intrusion dans le SI.

7.2 Actions curatives

- Nous devons impérativement repérer les serveurs victimes de l'attaque, les déconnecter d'internet dans un premier temps afin de limiter les intrusions et les remplacer par des machines de réplications s'il en existe.
- Mettre dans un premier temps le paramètre "readonly" à true et bloquer les requêtes PUT sur le serveur temporairement.

7.3 Actions post-attaque

- Réinitialiser les machines victimes afin de supprimer toute traces malveillantes de l'attaquant.
- Mettre à jour son serveur Apache Tomcat au cas où la faille est déjà corrigée.
- Si la faille est encore présente même après une mise à jour, l'entreprise peut passer à des requêtes de type POST, permettant une validation plus poussée au niveau applicative.
- Informer le service juridique qui se chargera d'envoyer le formulaire de déclaration d'incident à l'ANSSI.

8 Glossaire

API: Application Programming Interface, une interface qui permet la communication entre deux systèmes différents.

Analyse post-mortem : Analyse de la situation après qu'un évènement important à eu lieu et fut résolu.

ANSSI : Agence nationale de la sécurité des systèmes d'information

JSP: JavaServer Pages, permet de générer des pages web tout en incluant du code Java.

JVM: La machine virtuelle Java qui exécute les programmes Java compilés.

PoC: Proof of Concept, un prototype créé pour démontrer la faisabilité de quelque chose.

Root : Nom donné à l'utilisateur qui possède toutes les permissions sur un système UNIX.

Servlet: Un programme Java qui se lance dans un serveur Web afin de recevoir et répondre aux requêtes HTTP.

SI : Système d'information.

Web shell : Script téléversé sur un serveur vulnérable et permettant le contrôle de la machine à distance.

9 Références

ils de la CVE en anglais : `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-12617`

ls de la CVE en français : `https://www.cyberveille-sante.gouv.fr/cyberveille/229-apache-tomcat-correction-dune-vulnerabil`

Bugtracker : `https://bz.apache.org/bugzilla/show_bug.cgi?id=61542`

Web Shell : `https://blog.netspi.com/hacking-with-jsp-shells/`