Heartbleed

Introduction à la sécurité des systèmes d'informations

29 décembre 2016

Guillaume Mallet, Tom Russello, Vincent Tavernier

Introduction

Nous avons choisi de travailler sur la faille nommée *Heartbleed* qui impacte *OpenSSL* entre les versions 1.0.1 et 1.0.1f.

OpenSSL est une librairie utilisée dans de nombreux sites pour chiffrer et authentifier les échanges, notamment par l'utilisation du protocole `SSL/TLS`.

Malheureusement, suite à une mise à jour, un bug permet à des attaquants de lire la mémoire du serveur vulnérable, qui peut contenir divers éléments tels que mots de passe, logins, clés, cookies de session...

Les détails techniques (lancement de l'image Docker, exploitation avec Metasploit) sont dans le fichier README.md contenu dans l'archive. Ce rapport ne présente que l'analyse de la vulnérabilité.

Des détails sur la faille

Quelles sont les victimes de cette faille

Les victimes sont les utilisateurs dont l'authenticité et la confidentialité des données n'est plus garantie lors de leurs échanges avec un serveur vulnérable. Les mots de passe pour l'authentification HTTP Basic et les cookies des utilisateurs peuvent être lus depuis la mémoire du serveur par une tierce partie. De plus, si la clé privée associée au certificat a été compromise, des attaques de type Man in the Middle deviennent possibles, l'attaquant pouvant s'authentifier en tant que serveur légitime pour l'échange SSL/TLS. Il peut ainsi écouter tous les échanges entre l'utilisateur et le véritable serveur.

Fonctionnement de la faille

TLS implémente une fonctionnalité nommée *Heartbeat* qui permet de savoir si un pair est toujours en vie en envoyant un payload. Si le pair est encore en vie, il doit renvoyer ce même payload grâce au code page 2.

L'instruction `n2s` ligne 3 permet de mettre la taille du message (2 octets, donc jusqu'à 65535) dans la variable `payload`. Ensuite, ligne 17, on alloue de la mémoire à `buffer` dépendant de `payload` et donc de la taille du message pour pouvoir ligne 21 copier `payload` octets du message dans la réponse `bp`. Le problème est que le message est contrôlé par l'attaquant et qu'il peut donc très bien envoyer un message de 1 octet en mentant sur sa taille (par exemple le maximum 65535). Ainsi `memcpy` va renvoyer de la mémoire locale récemment libérée pouvant contenir des informations confidentielles.

Quel type de machine est affecté

Tout serveur utilisant une version d'OpenSSL vulnérable pour des échanges SSL/TLS.

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
if (s->msg_callback)
        s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                &s->s3->rrec.data[0], s->s3->rrec.length,
                s, s->msg_callback_arg);
if (hbtype == TLS1_HB_REQUEST)
        {
        unsigned char *buffer, *bp;
        int r;
        /* Allocate memory for the response, size is 1 bytes
         * message type, plus 2 bytes payload length, plus
         * payload, plus padding
         */
        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
        bp = buffer;
        /* Enter response type, length and copy payload */
        *bp++ = TLS1_HB_RESPONSE;
        s2n(payload, bp);
        memcpy(bp, pl, payload);
        bp += payload;
```

Listing 1 – Code où se trouve la faille

Le problème a été résolu grâce au code suivant où l'on vérifie la taille du message donné avec la taille réelle.

```
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```

Listing 2 – Correctif du problème

Résultats de l'exploitation de la faille

Grâce à l'exploitation de la faille, nous arrivons à trouver la clé privée mais qui n'est pas exactement la même que celle donnée à `nginx` comme vous pouvez le voir sur la figure 3. Ceci est dû au fait que les entiers premiers p et q sont inversés donnant une autre version de la même clé comme nous pouvons le voir dans la figure 2.

Il nous est aussi arrivé de trouver (pas systématiquement) le login et mot de passe lors du `dump` de la mémoire du serveur.

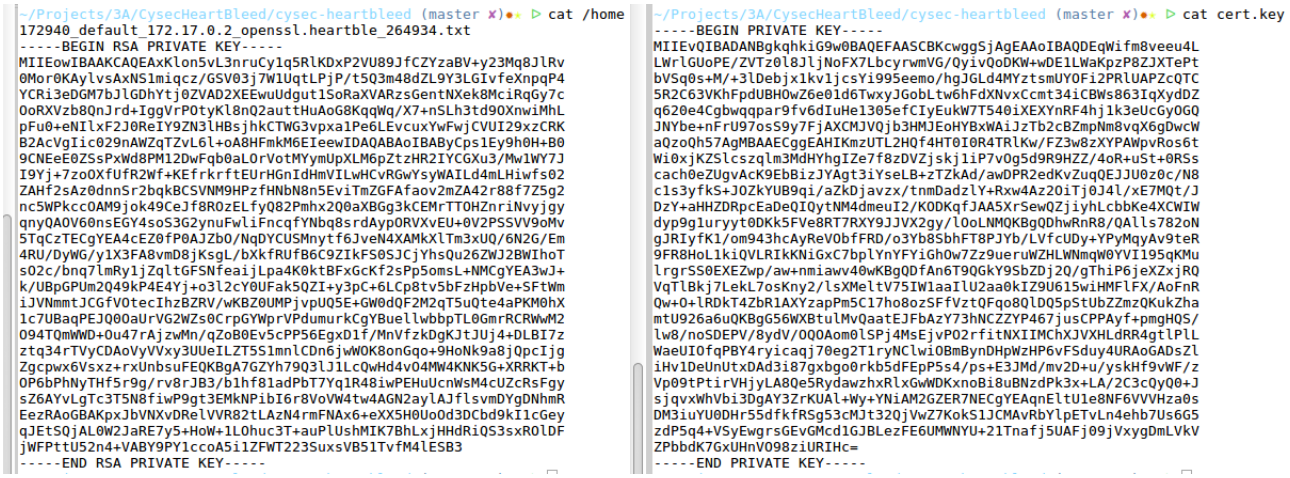

Figure 1 – Clé privée trouvée grâce à heartbleed à gauche et clé donnée en paramètre à `nginx` à droite

Nous avons les caractéristiques des clés sur la figure page 4 avec à gauche celles de la clé renvoyée par l'*exploit* et à droite celles de la clé donnée en paramètre de `nginx`. On peut donc voir que les paramètres sont les mêmes et que donc la clé qui a été dump est la bonne : avec cette clé, plus aucune donnée n'est protégée.

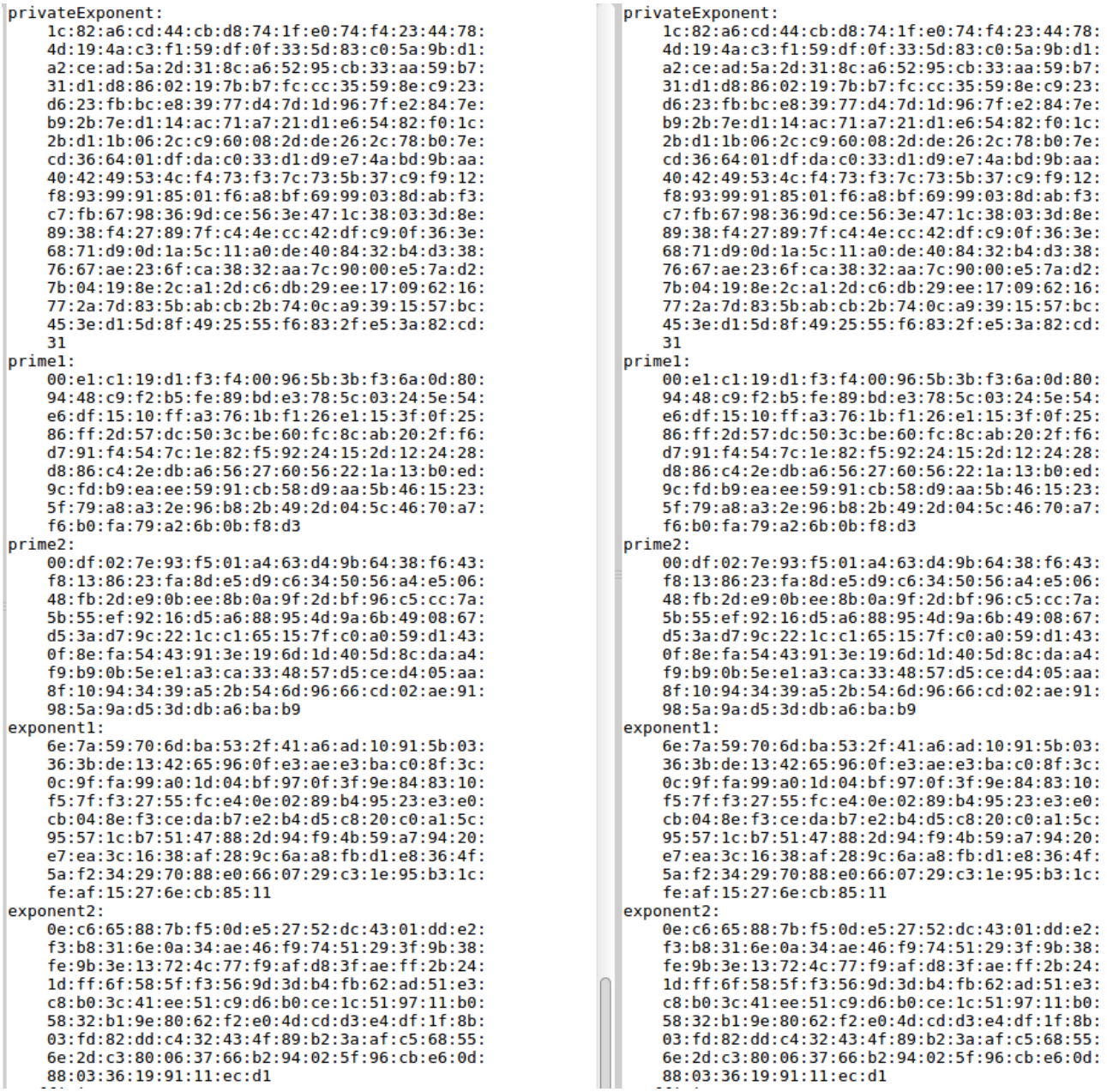

Figure 2 – Caractéristiques des clés privées

Architecture typique du système information permettant l'exploit

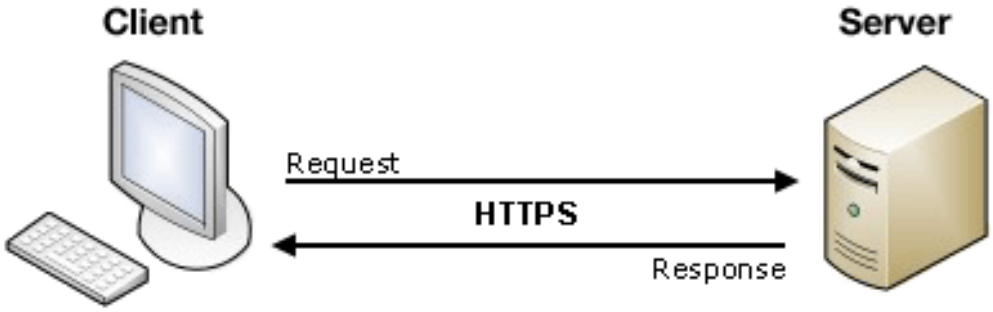

Figure 3 – Modèle client-serveur

Pour exploiter cette faille, il n'y a pas besoin d'avoir une architecture très complexe : un simple client peut être l'attaquant et peut dumper la mémoire du serveur contenant les informations confidentielles. C'est en partie pour cela que la faille est dangereuse. En effet, n'importe qui pouvait en quelques secondes recueillir des mots de passe anonymement sans s'introduire dans le réseau de quelqu'un ou du serveur et sans pouvoir se rendre compte de l'attaque.

Mesures de sécurité

Il est donc important de mettre à jour *OpenSSL* pour bénéficier des correctifs de la faille. Il existe aussi des scripts qui permettent de voir si son site est vulnérable à ce type de faille. En revanche, il n'existe pas de moyen de "limiter les dégâts" si la version utilsée d'*OpenSSL* est atteinte par le bug.

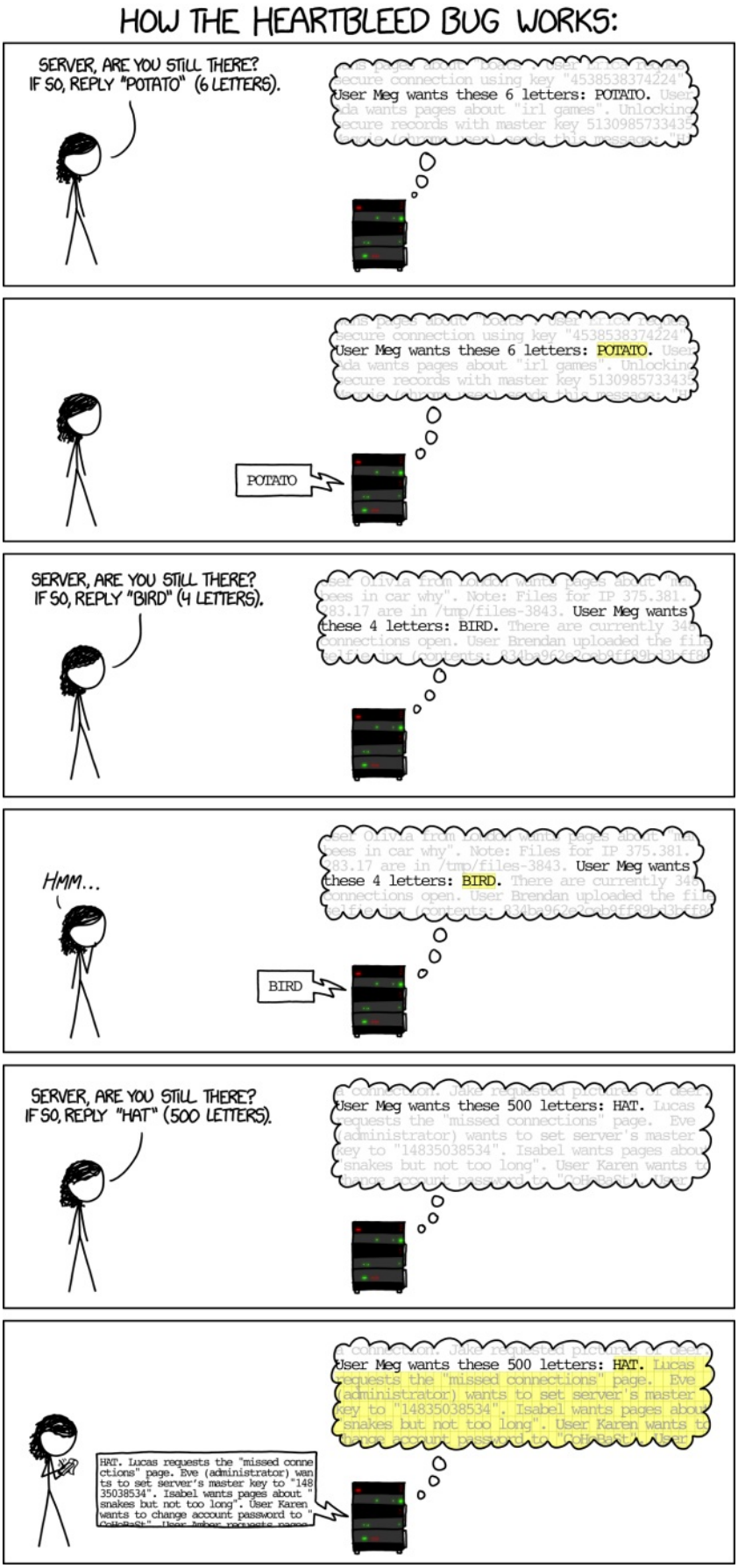

Figure 4 – Explication de la faille Hearbleed – XKCD