Analyse de failles directes et indirectes à OpenSSL

Mathieu Latimier et Claire Smets

17 février 2014

Table des matières

1 Introduction

Sur ce document, nous nous focaliserons sur les failles et les correctifs à appliquer sur les primitives cryptographiques – selon les systèmes utilisés. Dans l'ordre : la génération de l'aléa, la génération des clés, l'échange des clés, les signatures et vérifications, l'authentification, la génération des masques et les protocoles. J'émets également des doutes sur le code utilisé par OpenSSL, notamment sur les appels de PRNG comme /dev/urandom.

La partie "Forces et faiblesses des différents PRNG" a été rédigé conjointement avec Pascal Edouard et Julien Legras.

Enfin, une partie est consacrée sur la possibilité d'une back-door par la NSA dans les outils de chiffrement (dont OpenSSL) et une corruption au sein des normes et des standards (dont le NIST).

Vous trouverez l'ensemble de nos sources en Annexe, si elles ne sont pas déjà citées en cours de document. Les sources proviennent de l'ensemble de l'équipe.

2 Failles générales d'OpenSSL

2.1 Common Vulnerability & Exposure

Pour connaître l'ensemble des vulnérabilités et des expositions d'OpenSSL il est intéressant d'étudier les CVE (Common Vulnerability and Exposure).

Voici le lien de toutes les CVE concernant OpenSSL :

Source : www.cvedetails.com/vendor/217/ Openssl.html

Vous pouvez également parcourir la base de données des vulnérabilités du NIST. Voici la recherche pour "OpenSSL" :

Source : web.nvd.nist.gov/view/vuln/search-results?query=openssl&search_type=all&cves=on

On retrouve bien toutes les CVE, cependant l'affichage est moins esthétique que le précédent lien. Il peut toutefois être un bon complément pour la recherche d'informations sur une vulnérabilité particulière

Pour jeter un coup d'oeil sur le statut général des CVE OpenSSL :
supportcenter.checkpoint.com/supportcenter/portal?eventSubmit_doGoviewsolutiondetails&solutionid=sk92447

2.2 Common Weakness Enumeration

Pour ce qui est des failles logicielles (qui ne concerne OpenSSL qu'au niveau de l'application où il est installé), il y a le site CWE (Common Weakness Enumeration). Sur l'onglet de recherche entrez "OpenSSL" :

Source : cwe.mitre.org/find/index.html

2.3 Exemples de failles

Voici quelques failles récentes d'OpenSSL :

Le 07 janvier 2014, un déférencement de pointeur NULL via ssl3_take_mac() est la cause de DoS (Deny of Service) :

Les produits concernés sont Debian, Fedora, FreeBSD, Copssh, **OpenSSL**, openSUSE, pfSense, RHEL, Slackware.

La bibliothèque OpenSSL implémente SSL/TLS, pour les clients et les serveurs.

La fonction ssl3_take_mac() du fichier ssl/s3_both.c est utilisée par les clients pour calculer le Finished MAC. Cependant, la fonction ssl3_take_mac() ne vérifie pas si un pointeur est NULL, avant de l'utiliser.

Un serveur TLS illicite peut donc envoyer un handshake invalide vers le client OpenSSL, pour déréférencer un pointeur NULL, afin de mener un déni de service.

Le 06 janvier 2014, plusieurs vulnérabilités ont étés détectées sur OpenSSL. Elles permettraient à un attaquant distant de provoquer des DoS.

Versions affectées : antérieurs à OpenSSL 1.0.0l et 1.0.0f

Source :
www.cert.ssi.gouv.fr/site/CERTA-2014-AVI-003/CERTA-2014-AVI-003.html

Source :
vigilance.fr/vulnerabilite/OpenSSL-dereferencement-de-pointeur-NULL-via-ssl3-take-mac-14029

Le 02 janvier 2014, une faille est détecté sur le protocole DTLS d'OpenSSL.
Les produits concernés sont : Debian, Fedora, FreeBSD, Copssh, MBS, **OpenSSL**, openSUSE, pfSense, Puppet, RHEL, Slackware.

Le protocole DTLS (Datagram Transport Layer Security), basé sur TLS, fournit une couche cryptographique au-dessus du protocole UDP.

Cependant, la fonction dtls1_hm_fragment_new() du fichier ssl/d1_both.c ne gère pas correctement l'état de retransmission.

Un attaquant, placé en Man-in-the-middle, peut donc forcer l'utilisation d'un contexte DTLS invalide dans OpenSSL, afin d'obtenir des informations sensibles.

Source :
vigilance.fr/vulnerabilite/OpenSSL-obtention-d-information-via-DTLS-14007

Le 12 décembre 2013, TLS 1.2 est vulnérable à une attaque de type DoS.

Produits concernés : Debian, Fedora, FreeBSD, Copssh, **OpenSSL**, openSUSE, pfSense, RHEL, Slackware.

La bibliothèque OpenSSL supporte les versions 1.0 à 1.2 de TLS.

La fonction ssl_get_algorithm2() du fichier ssl/s3_lib.c obtient la version de la session en cours. Cependant, cette fonction utilise une structure qui n'est pas à jour parfois. Une erreur interne se produit alors.

Un attaquant peut donc employer TLS 1.2 avec une application liée à OpenSSL, afin de mener un déni de service.

Source :
vigilance.fr/vulnerabilite/OpenSSL-deni-de-service-via-TLS-1-2-13978

Le 5 février 2013, deux failles de type DoS sur OpenSSL ont étés publiés.
La première est possible en combinant CBD et AES-NI.
Les produits concernés sont : HP-UX, Tivoli Workload Scheduler, **OpenSSL**, openSUSE, Slackware.

Depuis 2008, certains processeurs de la famille x86 implémentent les instructions assembleur AES-NI. Elles permettent de demander au processeur d'effectuer des calculs AES en une seule instruction.

Lorsqu'une application liée à OpenSSL s'exécute sur un processeur avec AES-NI, un attaquant peut stopper les sessions TLS en mode CBC.

Les détails techniques ne sont pas connus. L'erreur pourrait être située dans la fonction aesni_cbc_hmac_sha1_cipher() du fichier crypto/evp/e_aes_cbc_hmac_sha1.c.

Source :
vigilance.fr/vulnerabilite/OpenSSL-deni-de-service-via-CBC-et-AES-NI-12377

La seconde provient de OCSP.

Produits concernés : Debian, BIG-IP Appliance, Fedora, FreeBSD, HP-UX, AIX, Tivoli Workload Scheduler, Juniper J-Series, JUNOS, MBS, MES, McAfee Email and Web Security, **OpenSSL**, openSUSE, Solaris, pfSense, RHEL, JBoss Enterprise, Slackware, ESX, ESXi, vCenter, VMware vSphere Hypervisor.

L'extension OCSP (Online Certificate Status Protocol) vérifie la validité des certificats.

La fonction OCSP_basic_verify() du fichier crypto/ocsp/ocsp_vfy.c décode la réponse OCSP reçue. Cependant, si la clé est vide, un pointeur NULL est déréférencé.

Un attaquant peut donc mettre en place un serveur OCSP illicite, afin de stopper les applications OpenSSL qui s'y connectent.

Source :
vigilance.fr/vulnerabilite/OpenSSL-deni-de-service-via-OCSP-12378

Nous remarquons que tout les mois plusieurs failles sur OpenSSL apparaissent, et majoritairement des DoS.

Nous pourrons donc nous servir des précédentes failles pour notre audit, afin de détecter d'éventuelles failles du même type.

3 Failles au niveau du générateur de l'aléa

3.1 Debian 4.0 et OpenSSL 0.9.8

Il n'y a à priori aucune faille sur les générateurs d'aléas dans OpenSSL.
Cependant, l'utilisation de générateurs d'aléas provenant de systèmes d'exploitations, peuvent engendrer de fortes pertes d'entropie, voir la rendre totalement nulle.

Le 13 Mai 2008, Luciano Bello découvert une faille critique du paquet d'OpenSSL sur les systèmes Debian. Un mainteneur Debian souhaitant corriger quelques bugs aurait malencontreusement supprimé une grosse source d'entropie lors de la génération des clés.
Il ne restait plus que le PID comme source d'entropie !
Comme celui-ci ne pouvait dépasser 32.768 (qui le PID maximal atteignable), l'espace des clés a été restreint à 264.148 clés distinctes.

Source :
linuxfr.org/news/d%C3% A9couverte-dune-faille-de-s%C3%A9curit%C3%A9-critique-dans-openssl-de-deb

Analysons plus en détail cette faille. Elle se situe au niveau de la fonction **md_rand.c**. La ligne **MD_Update(&m, buf, j);** a été commentée. La conséquence est le blocage de la graine (seed) que l'on passe ensuite au PRNG.

Cette ligne a été commentée par erreur en voulant corriger un avertissement soulevé par le compilateur Valgrind sur une valeur non initialisé.

Le 14 Mai 2008, Steinar H. Gunderson démontre simplement comment en connaissant le secret k d'une signature, on peut retrouver la clé privée d'un certificat immédiatement.

Ce secret k étant généré avec un PRNG prévisible, on peut stocker deux signatures utilisant le même k, où le prédire directement.

Une signature DSA consiste en deux nombres r et s tels que :
r = (g^k [p]) [q]
s = (k^{-1} * (H(m) + x * r)) [q]

La clé publique = (p, q, g).
Le message en clair = m, et H(m) est le fingerprint de m connu.

Attaque n° 1 : En connaissant k
s * k [q] = (H(m) + x*r) [q]
<==> (s * k – H(m)) [q] = x*r [q]
<==> ((s*k – H(m))*r^{-1})[q] = x
<==> (s*k – H(m))*r^{-1} = x

Attaque n° 2 : Deux messages possèdent le même k
s1 = (k-1 (H(m1) + xr)) [q]
s2 = (k-1 (H(m2) + xr)) [q]

<==> s1 - s2 = (k-1 (H(m1) - H(m2)) [q]
<==> (s1 - s2)(H(m1) - H(m2))-1 = k-1 [q]
<==> On connaît k ==> Attaque 1

Source :
plog.sesse.net/blog/tech/2008-05-14-17-21_some_maths.html
Pour savoir si une clé SSL, SSH, DNSSEC ou OpenVPN est affectée, un détecteur de données de clés faibles est fourni par l'équipe Security de Debian :
security.debian.org/project/extra/dowkd/dowkd.pl.gz

Il y a également un logiciel plus simple d'utilisation toujours développé par l'équipe Debian Security - openssl-vulnkey - que vous pouvez télécharger à l'adresse ci-dessous :
packages.debian.org/fr/sid/openssl-blacklist

Pour voir l'avertissement de sécurité de l'équipe Debian :

Source :
www.debian.org/security/2008/dsa-1571

3.2 LinuxMintDebianEdition et Android

On pensait alors que cette faille ne surviendrait plus, que c'était une erreur farfelue (qui a tout de même durée plus de deux ans sur l'un des systèmes les plus en vogue - L'erreur datant de Septembre 2006).

Et récemment, en Août 2013 précisément, un patch de sécurité pour les systèmes Android utilisant la version LinuxMintDebianEdition/OpenSSL, dévoile une réparation du générateur de nombres pseudo-aléatoire (PRNG) qui ne donnait pas suffisamment d'entropie.

Le patch indique que le PRNG de cette version d'OpenSSL utilise dorénavant une combinaison de données plus ou moins prévisibles associées à l'entropie générées par /dev/urandom.
Mais sachant que le PRNG d'OpenSSL utilise lui-même /dev/urandom, on a du mal à comprendre pourquoi en rajouter davantage.

Eric Wong et Martin Boßlet apporte la solution sur leur site, l'erreur provient d'un bug "à la Debian", une simple ligne diffère de la version officielle d'OpenSSL (utilisant SecureRandom) à celle de OpenSSL : :Random ce situant dans la fonction ssleay_rand_bytes.

La cause est là même que celle de Debian, un patch de sécurité atteint la source d'entropie du PRNG. Alors que tout semblait être rentré dans l'ordre, un résidu de cette erreur reste dans cette version. Les développeurs d'OpenSSL assure que ça n'a pas d'impact (ou alors très peu) sur la sécurité globale. Mais à cause de la mémoire non initialisé des systèmes Android, la source d'entropie ne nous permet pas de générer des nombres non-prédictibles.

La conséquence n'est pas aussi lourde que celle de Debian, tout d'abord parce que le système Android est rarement utilisé pour du chiffrement de données sensible, et une attaque par prédiction bien que plus rapide qu'une attaque par brute-force, reste infaisable. Mais l'erreur est quand même là.

Sources :
- emboss.github.io/blog/2013/08/21/openssl-prng-is-not-really-fork-safe/
- www.nds.rub.de/media/nds/veroeffentlichungen/2013/03/25/paper_2.pdf
- android-developers.blogspot.de/2013/08/some-securerandom-thoughts.html

3.3 NetBSD 6.0 et OpenSSH

Autre faille du même genre sur les systèmes NetBSD 6.0 (Mais sur OpenSSH cette fois-ci) et datant de Mars 2013 :

C'est le syndrome OpenSSH de Debian qui frappe une nouvelle fois.
Du fait d'une parenthèse mal placée dans le code du fichier /src/sys/kern/subr_cprng.c, il s'avère que le générateur pseudo-aléatoire de NetBSD 6.0 est bien moins solide que ce qui était attendu.
C'est une manière polie de dire que sa sortie n'est pas assez aléatoire et qu'il faut d'urgence changer les clés SSH qui ont été générées avec ce noyau !

L'alerte de sécurité : ftp.netbsd.org/pub/NetBSD/security/advisories/NetBSD-SA2013-003.txt.asc

Un article de h-online : www.h-online.com/open/news/item/Weak-keys-in-NetBSD-1829336.html

Le problème est corrigé dans NetBSD Current et le fix sera disponible dans la furture version NetBSD 6.1. Il est probable que les dégâts seront bien moins étendus que lors de l'affaire OpenSSH dans Debian car les systèmes NetBSD 6.0 ne sont sans doute pas très fréquents.

Toutes les clefs générées en utilisant /dev/urandom sont vulnérables.

Analysons le diff :
cvsweb.netbsd.org/bsdweb.cgi/src/sys/kern/subr_cprng.c.diff?r1=1.14&r2=1.15&only_with_tag=MAIN&f=h).

Pour la partie concernant la ligne 183 : rnd_extract_data (key + r, sizeof(key - r), RND_EXTRACT_ANY);

Le deuxième paramètre devrait être sizeof(key) - r.

Il semble que dans la version 1.15, cette appel a été déplacé (et corrigé rapidement) vers la nouvelle fonction cprng_entropy_try.
Source :
linuxfr.org/users/patrick_g/journaux/faille-de-securite-critique-dans-le-generateur-pseudo-aleatoire-de-netbsd-6-0.

3.4 Analyse de la faille Debian avec OpenSSL-Vulnkey

Finalement, nous avons décidé de tester le nombre de certificats vulnérables causés par le bug OpenSSL de Debian (qui reste le plus populaire), et connaissant la blacklist des clés privés.

Les résultats nous montrent que sur 500.000 certificats récupérés, au moins[1] 769 sont vulnérables.

1. Le logiciel ne prend pas en compte les clés $\leq$ à 512 bits et celles $\geq$ à 4096 bits, et ne prend en compte que les certificats RSA

Vous pouvez trouver nos scripts parcourant un fichier contenant un certificat sur chaque ligne, ou un dossier contenant des certificats sous forme de fichiers PEM et nos résultats, dans le dossier consacré à l'audit des clés cryptographiques.

Le format de nos résultats est :

COMPROMISED : *<haché_du_certificat> <nom _fichier_corrompu (sous forme d'adresse IP)*

Évidemment, nous ne mettons pas ces résultats sur le net puisqu'il indique très clairement les adresses IP contenant le certificat friable, et sa clé privée (que l'on peut facilement retrouvé parmi la courte blacklist).

Pour information, parmi les entreprises vulnérables nous trouvons les géants IBM et CISCO.

4 Forces et faiblesses des différents PRNG

Nos machines utilise ce qu'on appel des PRNG (Pseudo Random Number Generator), ce sont des algorithmes qui génèrent une séquence de nombre s'apparentant à de l'aléatoire. En réalité rien est aléatoire car tout est déterminé par des valeurs initiales (État du PRNG) et des contextes d'utilisation.

Un bon PRNG se doit d'avoir une très forte entropie (proche de un), afin d'éviter de délivrer de l'information.
Comme l'entropie est fourni majoritairement (si ce n'est totalement) par l'OS, il est donc nécessaire de détailler les PRNG les plus utilisés (Surtout par les systèmes Linux et BSD - qui sont les ceux qui génèrent le plus de certificats SSL).

Nous nous basons sur la RFC 4086 : Randomness requirements for security pour le choix des PRNG selon les différents systèmes.
Source : www.ietf.org/rfc/rfc4086.txt

4.1 /dev/urandom et /dev/random sous Linux

Sous Linux, un pool est initialisé avec 512 octets, auquel on ajoute le temps émis par un évènement et son état parmis :
- Les interruptions clavier - heure et code d'interruption
- Les interruptions de disques - heure de lecture ou écriture
- Les mouvements de souris - heure et position

Quand des octets aléatoires sont demandés, la pool est haché avec SHA-1 (20 octets). S'il est demandé plus que 20 octets, le haché est mélangé dans la pool pour rehacher la pool ensuite etc. À chaque fois que l'on prend des octets dans la pool, l'entropie estimée est décrémentée.

Pour assurer un niveau minimum d'entropie au démarrage, la pool est écrite dans un fichier à l'extinction de la machine.

/dev/urandom fonctionne selon le même principe sauf qu'il n'attend pas qu'il y ait assez d'entropie pour donner de l'aléatoire. Il convient pour une génération de clefs de session.

Pour générer des clefs cryptographiques de longue durée, il est recommandé d'utiliser /dev/random pour assurer un niveau minimum d'entropie.

En effet, sur un serveur sans souris ni clavier, définir l'entropie avec /dev/urandom est très risqué. On recommande donc l'utilisation de /dev/random lors de l'audit OpenSSL sur les versions Linux.

/dev/random utilise une pool d'entropie de 4096 bits (512 octets) génère de l'aléa et s'arrête lorsqu'il n'y a plus assez d'entropie et attend que la pool se remplisse à nouveau.

Si vous souhaitez connaître l'entropie disponible, la commande est :
cat /proc/sys/kernel/random/entropy_avail

Désormais, la taille de la pool est hardcodée dans le noyau Linux (/drivers/char/random.c :275)

Linux offre également la possibilité de récupérer de l'aléa depuis un RNG matériel avec la fonction get_random_bytes_arch

Source : wiki.archlinux.org/index.php/Random_Number_Generation

Un patch est également disponible afin de générer de l'aléa avec un débit de 100kB/s. L'entropie est récupérée par le CPU timing jitter.

Source : lkml.iu.edu//hypermail/linux/kernel/1302.1/00479.html

En conclusion, /dev/random doit être utilisé pour une haute qualité d'entropie (i.e. haute sécurité de chiffrement, one-time pad).
Tandis que /dev/urandom doit être utilisé pour des applications non sensibles à des attaques cryptographiques (i.e. jeu en temps réel), car elle génère plus d'entropie que /dev/random sur un temps donné, mais s'arrêtera même si il n'a pas récolté suffisamment d'entropie.

4.2 /dev/random sous FreeBSD et /dev/arandom sous OpenBSD

Il faut faire attention au faux ami, le /dev/random du FreeBSD n'est pas le même que celui de Linux. En fait, il est semblable au /dev/urandom de Linux, et est donc tout autant proscrit lors de notre audit.

Même principe avec le /dev/arandom de OpenBSD, qui a également une entropie faible pour du chiffrement cryptographique sûr. Il se base en fait sur un algorithme modifié du RC4 nommé ARC4 (Alleged RC4) pour générer des données aléatoires.
Pour rappel, RC4 était un projet commercial de la RSA Security, et un hacker anonyme a publié un code identique, devenu légitime, identifié par ARC4.
De nos jours, il est fortement conseillé de ne plus utiliser RC4 car le flux de données aléatoire n'est pas vraiment aléatoire et il existe des attaques qui prédisent la sortie de l'algorithme (Attaque de Fluhrer, Mantin et Shamir).
Sur plusieurs de nos sources (plus anciennes), il est recommandé d'utiliser /dev/arandom pour sa rapidité (71 Mb/s) et sa bonne source d'entropie. Ce n'est plus vraiment le cas aujourd'hui.

Source : www.cs.rit.edu/ axl6334/crypto/Report.pdf

4.3 CryptGenRandom sous Windows

Du coté de Microsoft, il recommande aux utilisateurs de Windows d'utiliser CryptGenRandom, qui est un appel système de génération d'un nombre pseudo-aléatoire. La génération est réalisée par une librairie cryptographique (Cryptographic service provider library). Celui ci gère un pointeur vers un buffer en lui fournissant de l'entropie afin de générer un nombre pseudo aléatoire en retour avec en plus, le nombre d'octet d'aléatoire désiré.

```
BOOL WINAPI CryptGenRandom(
        _In_    HCRYPTPROV hProv,
        _In_    DWORD dwLen,
        _Inout_ BYTE *pbBuffer
);
```

Le service provider sauvegarde une variable d'état d'un sel pour chaque utilisateur. Lorsque CryptGenRandom est appelé, celui ci est combiné avec un nombre aléatoire généré par la librairie en plus de différentes données systèmes et utilisateurs telles que :

- l'ID du processus

- l'ID du thread
- l'horloge système
- l'heure système
- l'état de la mémoire
- l'espace de disque disponible du cluster
- le haché du block d'environnement mémoire de l'utilisateur

Le tout est envoyé à la fonction de hachage SHA-1 et le nombre en sortie est utiliser comme sel pour une clef RC4.

Cette clef est enfin utilisé pour produire des données pseudo aléatoire et mettre à jour la variable d'état du sel de l'utilisateur.

4.4 Autres systèmes

Nous avons également d'autres RNG comme srandom, prandom, wrandom, ici sur MirOS BSD :
www.mirbsd.org/htman/i386/man4/arandom.htm

/dev/srandom est simple et lent, il n'est pas recommandé de l'utilisé.

Certains systèmes ne disposant pas de /dev/*random, il est alors possible d'utiliser l'EGD (Entropy Gathering Daemon).

Source :
egd.sourceforge.net/
Il faut pour cela utiliser les fonctions OpenSSL RAND_egd, RAND_egd_bytes et RAND_query_egd_bytes.

L'EGD est également utilisé par GPG, et peut être utilisé comme seed.

5 Failles au niveau de la génération et l'échange des clés

Lorsque nous avons étudiés le code de Diffie-Hellman dans OpenSSL, nous nous sommes penchés sur un choix plutôt étrange .
La valeur du générateur est toujours fixé à 2 ou à 5.

Le générateur de Diffie-Hellman n'étant pas une racine primitive dans $\mathbb{Z}/\mathbb{Z}_p$, les conséquences sont :
- L'espace des clés possibles est fortement réduit (Si g=2 => espace divisé par deux)
- Deux clés privées distinctes pourront avoir une clé publique commune
- La méthode de cryptanalyse Baby-step Giant-step peut s'en trouver facilité.

Évidemment, ce choix n'est pas une faille en soit, il n'est juste pas optimal et résulte d'un bon compromis entre vitesse et sécurité.

Pour une sécurité optimale, il est conseillé de choisir un générateur qui soit une racine primitive, pour être certain que personne ne puisse signer, déchiffrer des messages à votre place !

Voici l'algorithme de génération de g (d'après la RFC 2631 - 1999 / dérivé de [FIPS-186]) :

- 1- Soit j = (p - 1)/q.
- 2- Choisir h $\in \mathbb{N}$, tel que 1 < h < p - 1
- 3- Calculer g = h^j mod p
- 4- Si g = 1 recommencer l'étape 2

Mais depuis 2006, on peut lire comme recommandation dans la RFC 4419 (Pour une utilisation SSH) :
" It is recommended to use 2 as generator, because it improves efficiency in multiplication performance. It is usable even when it is not a primitive root, as it still covers half of the space of possible residues. "

OpenSSL peut être compilé en mode FIPS (Federal Information Processing Standard) avec "./config fipscanisterbuild".

Cependant, un attaquant, situé entre le client et le serveur, et connaissant la clé secrète du serveur, peut déchiffrer une session SSL/TLS.

L'algorithme EDH/DHE (Diffie-Hellman Ephémère) permet de calculer une nouvelle clé connue uniquement du client et du serveur, donc l'attaquant intermédiaire ne peut plus déchiffrer la session. Cependant, en mode FIPS, OpenSSL ne rejette pas les paramètres P/Q faibles pour EDH/DHE.
Lorsque OpenSSL est compilé en mode FIPS, un attaquant en Man-in-the-middle peut donc forcer la génération d'un secret Diffie Hellman prédictible.
Source :
vigilance.fr/vulnerabilite/OpenSSL-Man-in-the-middle-FIPS-Diffie-Hellman-10585
Date : Avril 2011

6 Failles au niveau de l'authentification, la signature et la vérification

L'Université du Michigan a réussi l'exploit de récupérer la clé privée d'un certificat RSA en un peu plus de 100h. L'attaque fonctionne par injection de fautes sur la méthode d'authentification. La technique est donc très poussée, mais le résultat en vaut le détour.

Voici un petit tutoriel sur l'injection de faute :
rdist.root.org/2008/03/10/advances-in-rsa-fault-attacks/
L'injection de faute doit se faire sur quelques bits pour ne pas disfonctionner le système tout entier. Les signatures erronées produites révèleront de l'information sur la clé privée. Avec le bon matériel et 100h d'attente, la clé peut être reforgé.

La cause venait de l'algorithme d'exponentiation modulaire (Fixed-Window Exponentiation), qui a l'inconvénient d'utiliser plus de 1000 multiplications. La multiplication étant très sensible en cas de dégradation du microprocesseur.

"The fixed-window exponentiation algorithm in the OpenSSL library does not validate the correctness of the signature produced before sending it to the client, a vulnerability that we exploit in our attack"

Sources :
web.eecs.umich.edu/ valeria/research/publications/DATE10RSA.pdf
www.theregister.co.uk/2010/03/04/severe_openssl_vulnerability/

Malheureusement pour pouvoir exploiter cette faille il faut pouvoir contrôler la machine (en ayant un accès au BIOS par exemple).
En 2008, une vulnérabilité sur la malformation des signatures survient sur OpenSSL (re-analysé en Novembre 2012) :

Sources :
www.openssl.org/news/secadv_20090107.txt
web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-5077

Dans les recommandations générales, il est clairement indiqué que ce sont les clients qui ne doivent plus utiliser une ancienne version d'OpenSSL ou alors ne pas utiliser de certificats DSA/ECDSA

Reste à savoir si c'est toujours d'actualité (e.g. si la faille est toujours exploitable), et si les serveurs respectent bien la recommandation.
En 2009, même cas trouvé dans un autre protocole (NTP) avec la même fonction EVP_VerifyFinal :
www.cvedetails.com/cve/CVE-2009-0021/

Autres sources :
cwe.mitre.org/data/definitions/599.html

7 Failles au niveau de la génération des masques

RSA-OAEP peut être soumis à une attaque nommée "Mangers Attack" selon son implantation. OpenSSL semble être vulnérable à une attaque de ce type, à base de "prédictions".
La vulnérabilité semble être très récente puisqu'elle fonctionne sous OpenSSL 1.0.0.

La Technische Universität Darmstadt (Allemagne) apporte des contre-mesures, et note qu'il existe toutefois des cas où l'algorithme est plus résistant.

Source :
www.cdc.informatik.tu-darmstadt.de/reports/reports/mangers_attack_revisited.pdf

Il semble également y avoir un problème avec l'OAEP_padding sur le chiffrement RSA. Bill Nickless recommande l'utilisation de PKCS_padding.

Source :
sourceforge.net/p/trousers/bugs/126/

8 NSA, NIST et RSA - une back-door dans nos systèmes cryptographiques ?

.

8.1 L'affaire Snowden et les documents top secrets de la NSA

Coup d'éclat en 2012, Edward Snowden, ancien-membre de la NSA et de la CIA, dévoile l'existence des backdoors ainsi qu'un lot d'informations conséquent sur la forte affluence de la NSA sur le NIST et la RSA.

Source :
www.reuters.com/article/2013/09/23/us-usa-security-snowden-rsa-idUSBRE98M06Q20130923

Il préleva ainsi plus de 1.700.000 documents de la NSA (d'après un officier de la NSA - 15 décembre 2013), dont 31.000 ultra-confidentiels.

Il délivra quelques documents à plusieurs journaux populaires tels que "The Guardian" et "The New York Times".

Parmis les documents top secrets rendus publiques, un en particulier nous intéresse, il concerne le contrôle de la NSA sur les systèmes de chiffrement actuels, nom de code BULLRUN, voici quelques points importants :

- Insert vulnerabilities into commercial encryption systems, IT systems, networks, endpoint communications devices used by targets
- Influence policies, standards and specification for commercial public key technologies

Sources concernant le projet BULLRUN : s3.documentcloud.org/documents/784159/sigintenabling-clean-1.pdf fr.wikipedia.org/w

Bruce Schneier, un des plus grands cryptologues actuel, et fervent détracteur de la NSA, publie plusieurs articles concernant ce contrôle d'informations sur la question "La NSA a t-elle réellement placé une backdoor au sein d'un nouveau système de chiffrement ?"

Sources :

www.wired.com/politics/security/commentary/securitymatters/2007/11/securitymatters_1115

Il évoque également le projet BULLRUN, montre comment la NSA peut placer ses backdoors, comment elle les choisient, et propose plusieurs stratégies de défense pour les vaincre :

- Les vendeurs doivent rendre au minimum le code du chiffrement publique (spécifications concernant les protocoles inclus). Le reste peut être conservé secret. Afin d'en détecter les vulnérabilités.
- La communauté des cryptologues doit pouvoir offrir une version compatible et indépendante du système de chiffrement, en open-source ou en vente auprès des entreprises privées (pour financer les universités par exemple).
- Aucun secret ! Tout doit être entièrement transparent auprès des clients.
- L'ensemble des PRNG doivent être rendus conformes avant publication et acceptation.
- Aucune fuite d'informations n'est permise, surtout au niveau des protocoles de chiffrement. Ceci afin d'éviter la prédiction de clés privées.

Source :
/www.schneier.com/blog/archives/2013/10/defending_again_1.html

En Septembre 2013, Matthew Green publie un article sur ce vaste problème entre la NSA et la sécurité cryptographique, qui a été salué par plusieurs cryptologues dont B. Schneier.

Il précise cependant que ça ne reste que des spéculations, mais qu'elles sont nécessaire afin de doubler d'effort dans la sécurité de nos communications.

Source : blog.cryptographyengineering.com/2013/09/on-nsa.html

8.2 La NSA verse 10M$ à la RSA Company

Un rapport de Snowden, indique que la NSA a déversé plus de 10.000.000$ à la compagnie RSA pour qu'elle utilise ce dernier, et discutable, algorithme comme générateur.
On comprend donc mieux les suscipsions autour d'un accord entre le NIST et la NSA pour la publication d'une recommandation de cet algorithme

Source :
www.techienews.co.uk/973955/report-nsa-paid-rsa-10m-use-dual-ec-drbg-preferred-random-number-generator/

8.3 Le NIST et l'algo Dual EC DRBG

Les recommandations du NIST en matière de PRNG (qu'ils appelent plutôt DRNG - Determinist Random Number Generation), débute en 2006, la publication du dernier document sur les DRNG date de Janvier 2012 avec la SP800-90A : csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

Ce document présente quatres algorithmes de PRNG qui sont :
- Le Hash_DRBG basé sur des fonctions de hachage
- Le HMAC_DRBG basé également sur des fonctions de hachage
- Le CTR_DRBG basé sur du chiffrement par bloc
- Le Dual Elliptic Curve Deterministic RBG (ou Dual EC DRBG) basé sur une théorie mathématique

Les trois premiers sont conventionnels, acceptés par toute la communauté des cryptologues, et s'avère efficace car ils génèrent "suffisamment" d'entropie.

Le dernier est très différent des trois autres, dans le sens où il utilise une fonction de chiffrement à sens unique. Certains cryptologues ont démontrés que cet algorithme posséde des failles (d'autres indiquent clairement que c'est une back-door du NIST...).

En effet, on peut accepter l'utilisation d'une fonction à sens unique, à condition que le secret utilisé ne soit pas conservé ailleurs (en d'autres termes qu'il soit détruit).

Que faire si le NIST garde le secret des algorithmes permettant d'affaiblir considérablement le Dual EC DRGB, et rendre l'aléatoire prévisible pour qui s'en donne les moyens ?

En recoupant plusieurs sources, le doute augmente considérablement.

En 2006, Berry Schoenmakers et Andrey Sidorenko établissent une cryptanalyse du DUAL_EC_DRGB.

Source :
www.propublica.org/documents/item/786216-cryptanalysis-of-the-dual-elliptic-curve

En 2007, Dan Shumow et Niels Ferguson furent les premiers à dénoncer le NIST d'avoir placer une backdoor délibérement dans cet algorithme.

Source :
rump2007.cr.yp.to/15-shumow.pdf

Avant Septembre 2013, tout cela n'était que suspicion, mais depuis le NIST à publié un bulletion de nouvelles recommandations pour les DRNG, et indique (surtout grâce à un forcing de la communauté cryptologue) que le Dual EC DRBG ne doit plus être utilisé pour les raisons suivantes :

- La provenance des points par défaut de la courbe elliptique utilisée n'est pas clairement détaillée
- La génération de ces courbes n'est pas digne de confiance

D'où découle la recommandation suivante :
"**Recommending against the use of SP 800-90A Dual Elliptic Curve Deterministic Random Bit Generation :** *NIST strongly recommends that, pending the resolution of the security concerns and there - issuance of SP 800-90A, the Dual_EC_DRBG, as specified in the January 2012 version of SP 800-90A, no longer be used*"

Source :
csrc.nist.gov/publications/nistbul/itlbul2013_09_supplemental.pdf

8.4 ... Et OpenSSL ?

Maintenant, il faut rechercher l'utilisation de cet algorithme dans les classes d'OpenSSL. La nouvelle recommandation du NIST faisant foi.

Le directeur technique d'OpenSSL : Steve Marques a posté le 19 décembre 2013 : "Un bug inusuel a été détecté sur une situation inusuelle".

L'implantation du DUAL_EC_DRGB dans OpenSSL contient une faille, celle- ci causant un arrêt brutal ou un blocage de programme.
Le bug a toujours été là, et il vient seulement d'être détecté.
Heureusement, personne n'a pu utiliser cet algorithme (à vérifier tout de même !), celui-ci est resté dans les phases de tests, les passant tout de même avec succès.

Source :
lwn.net/Articles/578375/

Source principal :
Source :

nakedsecurity.sophos.com/2013/12/22/the-openssl-software-bug-that-saves-you-from-surveillance/

Utilité : 8/10 **Date** : Décembre 2013

9 Failles sur les protocoles SSL et TLS

9.1 SSLv2

9.2 SSLv3 et TLSv1.0

En Septembre 2011, une attaque en man in the middle très efficace a vu le jour contre les protocoles SSLv3 et TLSv1.0.

L'attaque est à clair choisi. Le but étant d'insérer des morceaux de texte clair grâce au navigateur dans la requête chiffré avec ces protocoles, ceci afin de récupérer les cookies de session.

La technique est basique, un individu enregistre plusieurs cookies de session auprès de divers sites officiels (banques, messageries, etc...). Puis, il clique malencontreusement sur du code Java malveillant (publicité, image, etc...). Et là, l'attaque se déroule automatiquement, l'ensemble des cookies est envoyé au serveur malveillant qui n'a plus qu'à déchiffrer les clés de session.

La cause viendrait du mode de chiffrement choisi : CBC.

SSL/TLS est un protocole qui chiffre un canal de communication.
De ce fait il ne chiffre pas un fichier unique, mais une série d'enregistrements.
Il y a deux façon d'utiliser le mode CBC dans ce cas précis.

- Prendre chacun de ces enregistrements indépendamment des autres. Générer un nouveau vecteur d'initialisation à chaque fois.
- Traiter ces enregistrements comme un seul objet en les concaténant. Le vecteur d'initialisation est donc choisi aléatoirement pour le premier enregistrement et pour les autres, il aura pour valeur le dernier bloc de l'enregistrement précédent.

SSLv3 et TLS 1.0 utilisent ce deuxième choix, cela soulève un lourd problème de sécurité.

En 2004, Moeller trouve une méthode pour exploiter ce mauvais choix afin de récupérer des morceaux de textes clairs.

Sources :
www.educatedguesswork.org/2011/09/security_impact_of_the_rizzodu.html
www.imperialviolet.org/2011/09/23/chromeandbeast.html
www.theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl/?page=2
arstechnica.com/business/2011/09/new-javascript-hacking-tool-can-intercept-paypal-other-secure-sessions/

Un étudiant de l'Université de Versailles à développer un logiciel nommé BEAST en javascript (les sources sont introuvables).

La vidéo de l'attaque est accessible sur Youtube au lien ci dessous :
Source :

www.youtube.com/watch?v=ujz4SXzWK9o
Alors certes il y a une faille immense, mais peu exploitable.
Les grandes entreprises sont au courant (normalement) qu'il ne faut pas utiliser le mode CBC pour du chiffrement SSL/TLS. Et, dans tout les cas, plusieurs navigateurs ne permettent pas ce type d'attaque (c'est le cas de Chrome par exemple).

9.3 Validation des certificats SSL sans navigateur

Six chercheurs des universités de Stanford et d'Austin au Texas, analyse une attaque en Man in the Middle autour des certificats SSL sans utilisation d'un navigateur.

Le titre est sans appel "Le code le plus dangereux du monde".

Source : www.cs.utexas.edu/ shmat/shmat_ccs12.pdf

SSL doit permettre d'être sécurisé en toutes circonstances, que le cache DNS soit empoisonné, que les attaquants contrôlent les points d'accès et les routeurs, etc...
Il assure théoriquement trois grands principes de la cryptologie : la confidentialité, l'intégrité et **l'authentification**.
Nous connaissons certaines failles au niveau du navigateur et de l'implantation SSL (voir ci-dessus).

Mais il existe également d'autres cas d'utilisation du protocole SSL. Par exemple :
- Administration à distance basé sur le cloud, stockage sécurisé sur le cloud en local.
- Transmissions de données sensibles (ex : e-commerce)
- Services en lignes comme les messageries électroniques
- Authentifications via applications mobiles comme Android et iOS

L'étude montre que la validation des certificats SSL ne fonctionne pas sur plusieurs applications et librairies critiques dont OpenSSL.

10 ANNEXE A - Listes des vulnérabilités OpenSSL

CVE : www.cvedetails.com/vendor/217/Openssl.html

CWE : web.nvd.nist.gov/view/vuln/search-results?query=openssl&search_type=all&cves=on

CVE Particulière :
web.nvd.nist.gov/view/vuln/detail;jsessionid=902BC1FA28A415474A03C712CB991154?vulnId=CVE-2013-7295&cid=1

- CVE récente
- OpenSSL combiné à Intel Sandy Bridge
- Les RNG sont mauvais ! !
- Criticité plutôt haute : 4.9/10

Utilité : 9/10 **Date** : Janvier 2014

SecuObs : www.secuobs.com/revue/aaaopenssl0.shtml

11 ANNEXE B - État de l'art : Génération de l'aléa

NIST - Toolkit :
csrc.nist.gov/groups/ST/toolkit/rng/index.html
csrc.nist.gov/groups/ST/toolkit/random_number.html

- Téléchargement de logiciels (PRNG)
- Documentation
- Batterie de tests
- Analyse statistique
- Description de RNG

Utilité : 8/10 **Date** : Décembre 2013

NIST - Publications :
csrc.nist.gov/publications/PubsFIPS.html
csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
csrc.nist.gov/publications/drafts/800-90/draft-sp800-90c.pdf

- Publications et Drafts du NIST sur sécurité informatique (dont FIPS)
- FIPS 140-1 140-2 140-3

Utilité : 8/10 **Date** : Août 2013

INRIA :
hal.archives-ouvertes.fr/docs/00/73/86/38/PDF/rr8060.pdf

- Transfert d'entropie au sein du RNG de Linux
- Collecte d'entropie. Comment ?
- Tests

Utilité : 9/10

Publication de l'université de Jérusalem :
eprint.iacr.org/2006/086.pdf

- Analyse du RNG de Linux
- Collecte et analyse d'entropie
- Recommandations

Utilité : 9/10 **Date** : 2006

INTEL LABS :
crypto.stanford.edu/RealWorldCrypto/slides/jesse.pdf

- Graine des RNG
- Problème similaire à notre 1e partie
- Procédure de génération des clés RSA mise en doute
- Liste de recommandations
- Source d'entropie

Utilité : 9/10 **Date** : 2013-2014

Source :
www.linuxfromscratch.org/hints/downloads/files/entropy.txt

- /dev/random VS /dev/urandom
- Entropie et sécurité cryptographique
- Obtenir plus d'entropie avec Glibc et RNG-tools
- Modification de l'entropie dans OpenSSL
- Tester la qualité de l'entropie

Utilité : 9/10 **Date** : Mai 2007

TOR :
lists.torproject.org/pipermail/tor-talk/2013-December/031483.html

- Bug sur les RNG OpenSSL (entre autres)

Utilité : 9/10 **Date** : Décembre 2013

Calomel :
calomel.org/entropy_random_number_generators.html

- Bon tutoriel sur l'entropie et la génération de nombre aléatoire (PRNG ou DRBG)
- PRNG's state, exemple : /dev/random
- Quel problème avec les PRNG ? Analyse de random, urandom, arandom, srandom
- Il propose sa solution de PRNG
- Il teste les PRNG avec ENT
- Deux qualités d'un PRNG = vitesse + qualité entropie
- MAIS surtout qualité d'entropie ! La crypto est trop sensible !
- Augmenter l'entropie sur Linux avec RNGD (Julien a trouvé mieux) On passe de 150 o/s ← 4096 o/s (Julien 100 Mb/s)
- NIST SP800-90, FIPS 140-2, and ANSI X9.82

Utilité : 8/10 **Date** : Août 2013

Henric :
www.henric.info/random/

- Bon récapitulatif
- Liens vers NIST
- Tests statistiques (NIST, DIEHARD, TESTU01, ...)
- c7random
- Ajout d'entropie avec VIA PadLock OpenSSL Patch

Utilité : 7/10 **Date** : 2007-2008

International Computer Institute :
ube.ege.edu.tr/ mutaf/random.pdf

- Généralités
- Statistiques d'aléatoire, entropie, prédiction
- Problèmes avec certains PRNG
- PRNG supposés forts
- Standards

Utilité : 5/10

The POSSE Project :
www.cis.upenn.edu/ dsl/POSSE/

- Equipe de chercheurs en sécurité
- Audit de plusieurs logiciels open source dont OpenSSL
- Articles

Utilité : 5/10 **Date** : 2003

Blog - Analyse de l'API Random d'OpenSSL :
jbp.io/2014/01/16/openssl-rand-api/

- Rand_bytes() et Rand_pseudo_bytes()
- Tests de non-sûreté
- Recommandations et patchs de sécurité

Utilité : 9/10 **Date** : Janvier 2014

Blog - La cryptographie d'OpenSSL est-elle cassée ? :
www.linuxadvocates.com/2013/09/is-openssls-cryptography-broken.html

- NSA can break internet Encryption (RSA ?)
- Courbes elliptiques, et choix d'aléa
- Constantes générées par un employée de la NSA
- A prendre avec des pincettes (trouver des sources fiables)

Utilité : 8/10 **Date** : Septembre 2013

Tutoriel - Utiliser l'API OpenSSL Random Number :
etutorials.org/Programming/secure+programming/Chapter+11.+Random+Numbers/11.9+Using+the+OpenSSL+Random+Number+A

- Problème
- Solution
- Discussion
- Listing des fonctions de génération d'aléatoires

Utilité : 7/10 **Date** : 2013

11.1 Forums - Discussions intéressantes autour de la randomisation

Source : marc.info/?l=openssl-dev&m=132733475012928&w=2

Utilité : 6/10 **Date** : Janvier 2012

Source : openssl.6102.n7.nabble.com/understanding-openssl-entropy-td42000.html

Utilité : 8/10 **Date** : Février 2012

Source : openssl.6102.n7.nabble.com/seed-RANDFILE-confusion-td19793.html

Utilité : 7/10 **Date** : Octobre 2012

Source : crypto.stackexchange.com/questions/9412/what-to-watch-for-with-openssl-generating-weak-keys-low-entropy

Utilité : 7/10 **Date** : Juillet 2013

Source : security.stackexchange.com/questions/3259/howto-seed-the-prng-in-openssl-properly

Utilité : 7/10 **Date** : Avril 2011

Source : stackoverflow.com/questions/18349321/random-number-generator-and-seed

Utilité : 6/10 **Date** : Septembre 2013

Source : security.stackexchange.com/questions/3259/howto-seed-the-prng-in-openssl-properly

Utilité : 7/10 **Date** : Juillet 2013

11.2 Autres sources

[WIKI] CryptGenRandom de Windows : en.wikipedia.org/wiki/CryptGenRandom

[WIKI] OpenSSL/Random Numbers : en.wikibooks.org/wiki/OpenSSL/Random_numbers

[BLOG] Bug OpenSSL/Debian : Accident ou Backdoor ? freedom-to-tinker.com/blog/kroll/software-transparency-debian-openssl-bug/

[AUDIT] Vulnérabilité toujours d'actualité - mais non exploitable : www.digital-chaos.net/2014/01/openssl-story-about-some-old-source.html

[FLAWS] OpenSSL Security Advisory - 2012 : lwn.net/Articles/475009/

[CWE] - Failles d'application (ex N-599, N-699) : cwe.mitre.org/data/definitions/599.html

12 ANNEXE C - État de l'art : Génération des clés

Sources :
tools.ietf.org/html/rfc2631
tools.ietf.org/html/rfc4419

Utilité : 8/10

Qui a raison entre OpenSSL 1.0.0 et 0.9.8 ?
La version 0.9.8 ne permet pas de rejeter les clés DH "éphémères" et "dégénérées".
La version 1.0.0 oui.

Source :
http://openssl.6102.n7.nabble.com/Degenerate-DH-key-vulnerability-in-0-9-8-td45380.html

Utilité : 4/10