localhost

SecureInfo | Cheat Sheet OpenSSL

7-9 minutes

Pour avoir eu besoin de chercher la date d'expiration d'un certificat, je me suis mis en tête de chercher une liste de commande pratique avec openssl. Cette cheat-sheet s'est appuyée sur les ressources suivantes : sslshoper.com/article-most-common-openssl-commands.html et stackoverflow.com.

0x01. DEFINITIONS

x509 :
Attribution certificat lié à une clé publique à un nom distinctif

PKCS#12 :
Public Key Cryptography Standards : Standard de cryptographie et format de fichier qui stoque les clé X.509

ASN.1 :
Abstract Syntax Notation One : Structure de donnée indépendant de son encodage

CSR :
Certificate Signing Request : demande de certificat de signature

POODLE :
Padding Oracle On Downgraded Legacy Encryption : vulnérabilité permettant via la technique de l'homme du milieu (man in the middle) l'attaque de padding oracle (attaque permettant de "deviner" le dernier octet du dernier bloc)

DROWN :

Decrypting RSA with Obsolete Weakened Encryption : attaque visant à récupérer la clé d'une session TLS et donc de déchiffrer une communication interceptée

HEARTBLEED :

Vulnérabilité permettant le vol de donnée depuis l'état en mémoire de OpenSSL (information disclosure)

statem :

Module d'implémentation des états de flux et d'acquittement (handshake)

RCE :

Remote Code Execution : vulnérabilité permettant d'exécuter une commande arbitraire à distance

0x02. BENCHMARKS

Benchmarks local :

Benchmarks distant :

```
openssl s_time -connect www.server.com:443 -cipher
ECDHE-RSA-AES256-GCM-SHA384
```

Liste les algo de chiffrement SSL :

Vérifier que openssl supporte bien un chiffrement :

```
openssl ciphers -v|grep 'ECDHE-RSA-AES256-GCM-SHA384'
```

Vérifie qu'un nombre est premier (attention convertit le nombre en hexa, 10 = A, 255 = FF, ...), calcul le temps de factorisation (avec time)

```
time openssl prime
98765432198765432198765432198765432198 71
```

0x03. EMPREINTES

Calcul d'empreinte ("digest" ou "hash") :

```
# STDIN
echo 'toto'|openssl dgst -sha1

# Fichier
openssl dgst -sha256 fichier.txt

# Récupérer l'empreinte uniquement
openssl dgst -sha512 fichier.txt | awk -F'= ' '{ print
$2 }'
```

Pour calculer le hash en base64, il faut convertir le hash hexadécimal en son équivalent binaire, puis faire un base64 du résultat :

```
buff=$(openssl dgst -sha512 fichier.txt | awk -F'= ' '{
print $2 }'|sed -r 's/(..)/\\x\1/g') ; printf
"$buff"|base64 -w0
```

Vérifier l'empreinte MD5 d'une clé publique pour vérifier qu'elle correspond ...

```
# ... avec le certificat
openssl x509 -noout -modulus -in certificat.crt |
openssl dgst -md5

# ... avec la clé publique
openssl rsa -noout -modulus -in private.key | openssl
dgst -md5

# ... avec le certificat de requ&ecirc;te de signature
openssl req -noout -modulus -in request.csr | openssl
dgst -md5
```

0x04. CHIFFREMENT SYMETRIQUE

Chiffrement de mot de passe avec crypt() :

```
echo 'password'|openssl passwd  -stdin
```

Liste des algorithmes de chiffrement :

```
openssl -h 2>&1|grep -A50 '^Cipher'|grep -v
'^Cipher'|tr " " "\n"|grep .|sort
```

Chiffrement symétrique :

```
# STDIN
echo 'mes données à chiffrer' | openssl enc -aes256 -in
```

```
fichier.txt -out fichier.enc -k 'M()t2p/-\sse'

# Fichier avec demande de mot de passe
openssl enc -aes256 -in fichier.txt -out fichier.enc
```

0x05. LIRE UN CERTIFICAT

Lire le certificat au format texte :

```
openssl x509 -text  -noout -in www.monsite.com.pem
```

Récupérer la date d'expiration :

```
openssl x509 -enddate -noout -in www.monsite.com.pem
```

Décrit l'erreur de sortie :

Lire infos depuis fichier PEM (avec conversion au format DER) :

```
openssl x509 -in www.monsite.com.pem -outform
DER|openssl asn1parse -inform DER -i
```

0x06. GENERATION

Générer une clé privée et un CSR

```
openssl req -out csr.csr -new -newkey rsa:2048 -nodes
-keyout privatekey.key
```

Générer un certificat autosigné

```
openssl req -x509 -sha256 -nodes -days 365 -newkey
rsa:2048 -keyout private.key -out certificat.crt
```

Générer un CSR avec une clé déjà existante

```
openssl req -out csr.csr -key private.key -new
```

Générer un CSR avec un certificat déjà existant

```
openssl req -x509toreq -in certificate.crt -out csr.csr
-signkey private.key
```

0x07. CONVERSIONS

Conversion PKCS12 (X509) -> PEM :

```
# Conversion de la clef publique
openssl pkcs12 -in certificat.pfx -out
```

```
certificat.nokey.pem -nokeys

# Conversion du couple clef publique/privee
openssl pkcs12 -in certificat.pfx -out
certificat.withkey.pem

# Extraction de la clef privee depuis le PEM incluant
la clef
openssl rsa -in certificat.withkey.pem -out
certificat.key

# Fusion de la clé publique et de la clé privée
cat certificat.nokey.pem certificat.key >
certificat.pem
```

Conversion PKCS12 (X509) -> Certificat racine au format PEM + Certificat clients au format PEM

```
# Extraction des certificats clients (cl : clients)
openssl pkcs12 -in certificat.pfx -out
certificat_clients.pem -clcerts

# Extraction des certificats root (ca : authority)
openssl pkcs12 -in certificat.pfx -out
certificat_root.pem -cacerts
```

Conversion PEM (X509) + CERT -> PKCS12

```
# Extraction des certificats clients (cl : clients)
openssl pkcs12 -inkey site_key.pem -in site.cert
-export -out site.pfx
```

0x08. SIGNER UN FICHIER

Un certificat est nécessaire :

```
openssl req -x509  -sha256 -nodes -days 365 -newkey
rsa:2048 -keyout private.key -out certificat.crt
```

La signature s'effectue avec la clé privée :

```
openssl dgst -sha256 -sign private.key mon_fichier >
ma_signature.sig
```

La vérification de la signature se fait avec la clé publique :

```
openssl x509 -in certificat.crt -pubkey > public.key
openssl dgst -sha256 -verify public.key -signature
ma_signature.sig mon_fichier
```

0x09. VULNERABILITES

Vérifier la vulnérabilité POODLE (CVE-2014-3566)

```
target="www.monsite.com"
echo | timeout 3 openssl s_client -connect $target:443
```

```
>/dev/null 2>&1
if [ $? -ne 0 ]; then
  status="CONNECT_ERROR"
else
  status="VULNERABLE"
  echo | openssl s_client -connect $target:443 -ssl3
2>&1 |
   grep -qoi "sslv3 alert handshake
failure|SSL3_GET_RECORD:wrong version number" &&
   status="NOT_VULNERABLE"
fi
echo "$target:$status"
```

Vérifier la vulnérabilité DROWN (CVE-2016-0800)

```
echo|openssl s_client -connect www.monsite.com:443
2>&1|grep "SSLv2"
```

Vérifier si la version est vulnérable à HEARTBLEED (CVE-2014-0160) (versions : 1.0.1 à 1.0.1f inclus)

```
status="NOT_VULNERABLE" ; openssl version|egrep -sq
'1.0.1[a-f]?' || status="VULNERABLE" ; echo
"$(hostname):$status"
```

Vérifier si la version est vulnérable à Handshake Renegotiation (CVE-2017-3733) (versions : 1.1.0 à 1.1.0d inclus)

```
status="NOT_VULNERABLE" ; openssl version|egrep -sq
'1.1.0[a-d]?' || status="VULNERABLE" ; echo
"$(hostname):$status"
```

Vérifier si la version est vulnérable à ChaCha-Poly (DoS) (CVE-2016-7054) (versions : 1.1.0 à 1.1.0b inclus)

```
status="NOT_VULNERABLE" ; openssl version|egrep -sq
'1.1.0[a-b]?' || status="VULNERABLE" ; echo
"$(hostname):$status"
```

Vérifier si la version de openssl contient le module statem vulnérable (critical RCE) (CVE-2016-6309) (version : 1.1.0a uniquement)

```
status="NOT_VULNERABLE" ; openssl version|egrep -sq
'1.1.0a' || status="VULNERABLE" ; echo
"$(hostname):$status"
```

=> Écrit par : Nicolas, le 27 février 2018