SSL/TLS, 3 ans plus tard

Olivier Levillain

ANSSI

4 juin 2015

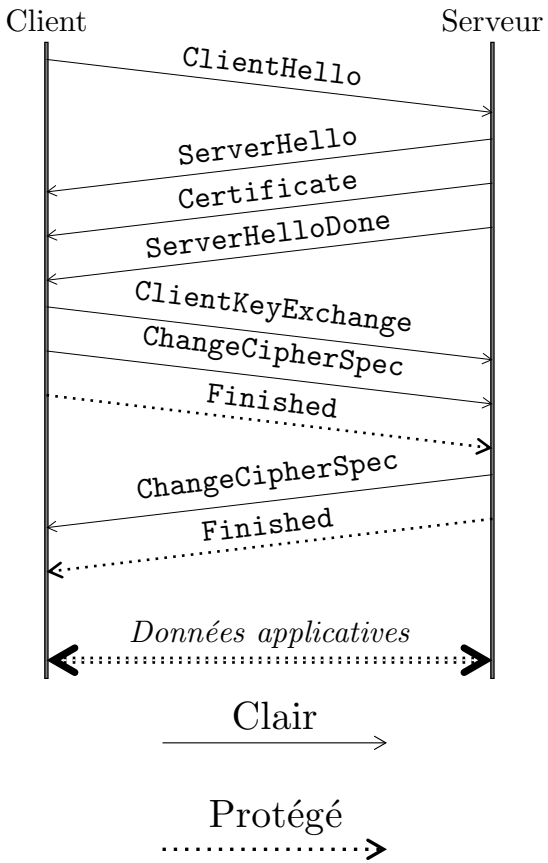
SSL/TLS : une brique essentielle d'Internet

- `https://` inventé par Netscape en 1995
  - début du commerce en ligne
- Utilisation massive aujourd'hui
  - HTTPS, bien au-delà du commerce en ligne
  - Sécurisation d'autres protocoles (SMTP, IMAP, LDAP, etc.)
  - VPN SSL
  - EAP TLS

SSL/TLS : une brique essentielle d'Internet

- `https://` inventé par Netscape en 1995
  - début du commerce en ligne
- Utilisation massive aujourd'hui
  - HTTPS, bien au-delà du commerce en ligne
  - Sécurisation d'autres protocoles (SMTP, IMAP, LDAP, etc.)
  - VPN SSL
  - EAP TLS
- SSL (*Secure Sockets Layer*) ou TLS (*Transport Layer Security*) ?

Rappels sur SSL/TLS (planche développement durable)



Objectif de la phase de négociation

- choisir les algorithmes (`Au`, `Kx`, `Enc`, `Mac`)
- authentifier le serveur
- établir un secret partagé, le *master secret*

TLS : la réalité du terrain

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `ECDH-ECDSA-AES128-SHA` ?

TLS : la réalité du terrain

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `ECDH-ECDSA-AES128-SHA` ?

A `AES128-SHA`

TLS : la réalité du terrain

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `ECDH-ECDSA-AES128-SHA` ?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

TLS : la réalité du terrain

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `ECDH-ECDSA-AES128-SHA` ?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C une alerte

TLS : la réalité du terrain

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `ECDH-ECDSA-AES128-SHA` ?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C une alerte

D la réponse D (`RC4_MD5`)

TLS : la réalité du terrain

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `ECDH-ECDSA-AES128-SHA` ?

A `AES128-SHA` (`0x002f`)

B `ECDH-ECDSA-AES128-SHA` (`0xc005`)

C une alerte

D la réponse D (`RC4_MD5`) (`0x0005`)

Le pire, c'est qu'on peut l'expliquer :

- une suite cryptographique est un entier sur 16 bits
- pendant longtemps, les seules valeurs utilisées étaient `00 XX`
- du coup, pourquoi considérer l'octet de poids fort ?

Pourquoi *{False,Snap} Start* a échoué ?

Un feuilleton qui a duré plusieurs années

- En 2010, Google propose des extensions, *False Start* et *Snap Start*
- Constat : une intolérance trop importante dans la nature
- Abandon en 2012 de ces propositions

Pourquoi *{False,Snap} Start* a échoué ?

Un feuilleton qui a duré plusieurs années

- En 2010, Google propose des extensions, *False Start* et *Snap Start*
- Constat : une intolérance trop importante dans la nature
- Abandon en 2012 de ces propositions

- Un an plus tard, le problème resurgit dans un autre contexte
- On apprend sur la liste `tls@ietf.org` qu'en fait, le problème vient d'un `ClientHello` trop gros...

Ce paquet est une chimère

Analysons les premiers octets d'un `ClientHello` de 258 octets

16	03	01	01	02

Ce paquet est une chimère

Analysons les premiers octets d'un `ClientHello` de 258 octets

16	03	01	01	02

Type	Version	Longueur
HS	*TLS 1.0*	*258*

TLS

Ce paquet est une chimère

Analysons les premiers octets d'un `ClientHello` de 258 octets

16	03	01	01	02

Type	Version	Longueur
HS	*TLS 1.0*	*258*

TLS

Longueur	*Pad.*	Type
5635	...	*CH*

SSLv2

Un `ClientHello` TLS dont la longueur comprise entre 256 et 511 peut être confondu avec un `ClientHello` SSLv2 !

Ce paquet est une chimère

Analysons les premiers octets d'un `ClientHello` de 258 octets

16	03	01	01	02

Type	Version	Longueur
HS	*TLS 1.0*	*258*

TLS

Longueur	*Pad.*	Type
5635	...	*CH*

SSLv2

Un `ClientHello` TLS dont la longueur comprise entre 256 et 511 peut être confondu avec un `ClientHello` SSLv2 !

Google a proposé une extension pour ajouter du bourrage...

Quoi de neuf depuis 2012 ?

TLS 1.3 et perspectives

Quelques vulnérabilités SSL/TLS depuis 2012

Quelques vulnérabilités SSL/TLS depuis 2012

- 2011 : BEAST (IV implicite dans le mode CBC)
- 2012 : CRIME (utilisation de la compression comme canal auxiliaire)
- 2013 : TIME et BREACH (améliorations/adaptations de CRIME)
- 2013 : *Lucky 13* (exploitation d'un oracle de *padding* CBC)
- 2013 : RC4 (exploitation de biais statistiques)

- 2014 : POODLE (exploitation d'un oracle de *padding* CBC avec SSLv3)

Quelques vulnérabilités SSL/TLS depuis 2012

- 2011 : BEAST (IV implicite dans le mode CBC)
- 2012 : CRIME (utilisation de la compression comme canal auxiliaire)
- 2013 : TIME et BREACH (améliorations/adaptations de CRIME)
- 2013 : *Lucky 13* (exploitation d'un oracle de *padding* CBC)
- 2013 : RC4 (exploitation de biais statistiques)
- 2014 : *Triple Handshake* (attaque sur la renégociation et la reprise de session)
- 2014 : POODLE (exploitation d'un oracle de *padding* CBC avec SSLv3)

Quelques vulnérabilités SSL/TLS depuis 2012

- 2011 : BEAST (IV implicite dans le mode CBC)
- 2012 : CRIME (utilisation de la compression comme canal auxiliaire)
- 2013 : TIME et BREACH (améliorations/adaptations de CRIME)
- 2013 : *Lucky 13* (exploitation d'un oracle de *padding* CBC)
- 2013 : RC4 (exploitation de biais statistiques)
- 2014 : `goto fail` Apple (...)
- 2014 : `goto fail` GnuTLS (`True`, `False`, `FILE_NOT_FOUND`)
- 2014 : *Heartbleed* (débordement de tampon en lecture)
- 2014 : *Triple Handshake* (attaque sur la renégociation et la reprise de session)

- 2014 : *Universal signature forgery* dans NSS (pendant *ShellShock*)
- 2014 : Exécution de code arbitraire dans *SChannel* (côté serveur)
- 2014 : POODLE (exploitation d'un oracle de *padding* CBC avec SSLv3)

Quelques vulnérabilités SSL/TLS depuis 2012

- 2011 : BEAST (IV implicite dans le mode CBC)
- 2012 : CRIME (utilisation de la compression comme canal auxiliaire)
- 2013 : TIME et BREACH (améliorations/adaptations de CRIME)
- 2013 : *Lucky 13* (exploitation d'un oracle de *padding* CBC)
- 2013 : RC4 (exploitation de biais statistiques)
- 2014 : `goto fail` Apple (...)
- 2014 : `goto fail` GnuTLS (`True`, `False`, `FILE_NOT_FOUND`)
- 2014 : *Heartbleed* (débordement de tampon en lecture)
- 2014 : *Triple Handshake* (attaque sur la renégociation et la reprise de session)
- 2014 : *EarlyCCS* (erreur dans l'automate d'état d'OpenSSL)
- 2014 : *Universal signature forgery* dans NSS (pendant *ShellShock*)
- 2014 : Exécution de code arbitraire dans *SChannel* (côté serveur)
- 2014 : POODLE (exploitation d'un oracle de *padding* CBC avec SSLv3)
- 2015 : SMACK/FREAK (automates d'état déficients + configurations antiques)
- 2015 : *LogJam* (configurations antiques + mauvaise négociation des groupes DH)

Quelques vulnérabilités SSL/TLS depuis 2012

- 2014 : *Triple Handshake* (attaque sur la renégociation et la reprise de session)
- 2014 : *EarlyCCS* (erreur dans l'automate d'état d'OpenSSL)

- 2015 : SMACK/FREAK (automates d'état déficients + configurations antiques)
- 2015 : *LogJam* (configurations antiques + mauvaise négociation des groupes DH)

De la dérivation du *master secret* (1/2)

Le *master secret* (et donc les clés de session) est dérivé à partir

- du résultat de l'échange de clé
- des aléas échangés en clair

L'intégrité de l'échange n'est garantie qu'avec les messages `Finished` !

De la dérivation du *master secret* (1/2)

Le *master secret* (et donc les clés de session) est dérivé à partir

- du résultat de l'échange de clé
- des aléas échangés en clair

L'intégrité de l'échange n'est garantie qu'avec les messages `Finished` !

Plusieurs attaques reposent sur cette dérivation à la couverture trop limitée

- présentation théorique des *cross-protocol attacks* (1999)
- propositions (presque) pratiques de ces attaques (2012)
- *Triple Handshake* (2014)
- SMACK/FREAK (2015)

De la dérivation du *master secret* (2/2)

Proposition de correction

- *session-hash* : la dérivation doit hacher tous les messages échangés
- document en cours de finalisation à l'IETF pour TLS 1.{0,1,2}
- intégration dans TLS 1.3

De la dérivation du *master secret* (2/2)

Proposition de correction

- *session-hash* : la dérivation doit hacher tous les messages échangés
- document en cours de finalisation à l'IETF pour TLS 1.{0,1,2}
- intégration dans TLS 1.3

Cela ne suffit pas, comme *LogJam* l'a montré

- le choix du groupe Diffie-Hellman est fait par la serveur à partir d'informations non intègres
- le groupe choisi est *signé* par le serveur avec les aléas
- un attaquant pouvant calculer le logarithme discret peut utiliser ce message et contrôler le reste de l'échange

De la dérivation du *master secret* (2/2)

Proposition de correction

- *session-hash* : la dérivation doit hacher tous les messages échangés
- document en cours de finalisation à l'IETF pour TLS 1.{0,1,2}
- intégration dans TLS 1.3

Cela ne suffit pas, comme *LogJam* l'a montré

- le choix du groupe Diffie-Hellman est fait par la serveur à partir d'informations non intègres
- le groupe choisi est *signé* par le serveur avec les aléas
- un attaquant pouvant calculer le logarithme discret peut utiliser ce message et contrôler le reste de l'échange

La solution : signer les messages précédents, et pas uniquement les aléas

Réflexion sur les attaques sur le mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller

Réflexion sur les attaques sur le mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller

- Attention aux oracles de *padding* dans le mode CBC de TLS

Réflexion sur les attaques sur le mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller

- Attention aux oracles de *padding* dans le mode CBC de TLS
- Le mode CBC avec IV implicite peut poser problème

Réflexion sur les attaques sur le mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller

- Attention aux oracles de *padding* dans le mode CBC de TLS
- Le mode CBC avec IV implicite peut poser problème
- Avec SSLv3, le *padding* n'est pas complètement spécifié, ce qui peut accentuer les oracles

Réflexion sur les attaques sur le mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller (2004-05-20)

- Attention aux oracles de *padding* dans le mode CBC de TLS
- Le mode CBC avec IV implicite peut poser problème
- Avec SSLv3, le *padding* n'est pas complètement spécifié, ce qui peut accentuer les oracles
- ... c'est-à-dire *Lucky 13*, BEAST et POODLE

Réflexion sur les attaques sur le mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller (2004-05-20)

- Attention aux oracles de *padding* dans le mode CBC de TLS
- Le mode CBC avec IV implicite peut poser problème
- Avec SSLv3, le *padding* n'est pas complètement spécifié, ce qui peut accentuer les oracles
- ... c'est-à-dire *Lucky 13*, BEAST et POODLE

Rassurez-vous, on est arrivé au bout du fichier...

True, False, FILE_NOT_FOUND

`http://thedailywtf.com/articles/What_Is_Truth_0x3f_`

Soit `f` une fonction C prenant une chaîne de certification et renvoyant

- `1` si la chaîne est reconnue comme valide
- `0` si la chaîne ne remonte pas à une autorité de confiance
- `-1` en cas d'erreur de *parsing* (non documenté)

True, False, FILE_NOT_FOUND

`http://thedailywtf.com/articles/What_Is_Truth_0x3f_`

Soit `f` une fonction C prenant une chaîne de certification et renvoyant

- `1` si la chaîne est reconnue comme valide
- `0` si la chaîne ne remonte pas à une autorité de confiance
- `-1` en cas d'erreur de *parsing* (non documenté)

Appel de `f` dans le code

- `if (f(certificates)) { ... }`

De quoi s'agit-il ?

True, False, FILE_NOT_FOUND

`http://thedailywtf.com/articles/What_Is_Truth_0x3f_`

Soit `f` une fonction C prenant une chaîne de certification et renvoyant

- `1` si la chaîne est reconnue comme valide
- `0` si la chaîne ne remonte pas à une autorité de confiance
- `-1` en cas d'erreur de *parsing* (non documenté)

Appel de `f` dans le code

- `if (f(certificates)) { ... }`

De quoi s'agit-il ?

- CVE-2014-0092 : `goto fail` GnuTLS

True, False, FILE_NOT_FOUND

`http://thedailywtf.com/articles/What_Is_Truth_0x3f_`

Soit `f` une fonction C prenant une chaîne de certification et renvoyant

- `1` si la chaîne est reconnue comme valide
- `0` si la chaîne ne remonte pas à une autorité de confiance
- `-1` en cas d'erreur de *parsing* (non documenté)

Appel de `f` dans le code

- `if (f(certificates)) { ... }`

De quoi s'agit-il ?

- CVE-2014-0092 : `goto fail` GnuTLS
- CVE-**2008**-5077 : *bug* identique dans OpenSSL

TLS dans tous ses états

Début 2014, plusieurs vulnérabilités des machines à état TLS

- EarlyCCS
- Exécution de code arbitraire dans *SChannel*
- SMACK/FREAK

TLS dans tous ses états

Début 2014, plusieurs vulnérabilités des machines à état TLS

- EarlyCCS
- Exécution de code arbitraire dans *SChannel*
- SMACK/FREAK

Qui est fautif lorsque toutes les piles TLS sont vulnérables et acceptent des messages hors contexte ?

- les développeurs de nombreuses piles indépendantes ?
- la spécification, peut-être un peu trop complexe ?

Quoi de neuf depuis 2012 ?

TLS 1.3 et perspectives

Enseignements tirés

Il faut améliorer la qualité des implémentations

- tests *négatifs* à développer
  - une photo de chat ne devrait pas être une chaîne valide
  - *Frankencerts* (IEEE SSP 2014)
- tests de l'automate d'état
  - FlexTLS, qui a déjà fait ses preuves
  - quelques autres projets démarrés récemment
- `<troll>` on peut aussi réécrire TLS en OCaml `</troll>`

Enseignements tirés

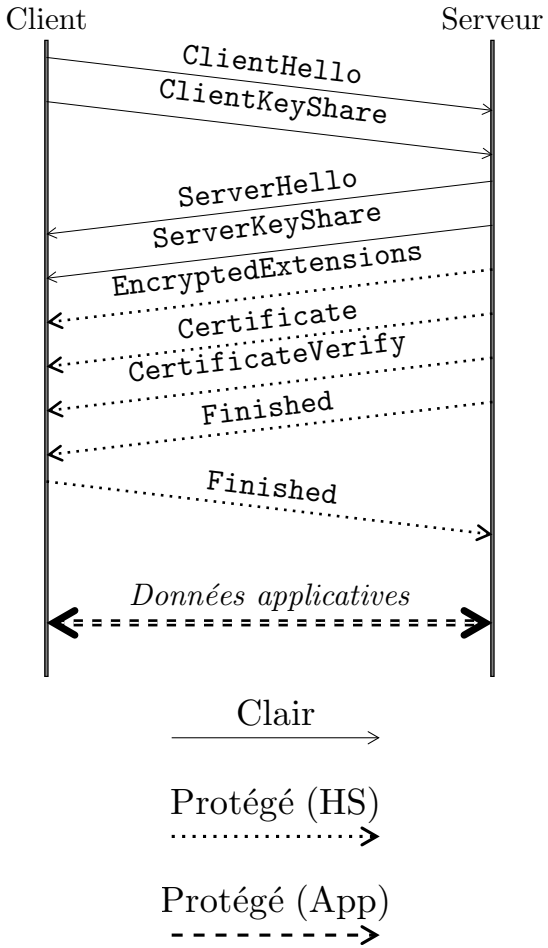
Il faut améliorer la qualité des implémentations

- tests *négatifs* à développer
  - une photo de chat ne devrait pas être une chaîne valide
  - *Frankencerts* (IEEE SSP 2014)
- tests de l'automate d'état
  - FlexTLS, qui a déjà fait ses preuves
  - quelques autres projets démarrés récemment
- `<troll>` on peut aussi réécrire TLS en OCaml `</troll>`

TLS est complexe

- ASN.1 et l'encodage DER fourmillent de pièges
- les algorithmes et modes cryptographiques antiques forcent à choisir entre sécurité et généricité
- la machine à état explose à cause du nombre de possibiltiés

TLS 1.3 : un échange de clés repensé



Avantages de ce nouveau flot de messages

- l'échange de clé RSA disparaît (*forward secrecy* pour tous)
- les groupes DHE et ECDHE sont prédéfinis
- un mode 1-RTT efficace
- réutilisation du message `CertificateVerify` pour le serveur
- plus de renégociation à l'ancienne

TLS 1.3 : Du nouveau dans la dérivation des clés

Une dérivation entièrement repensée

- utilisation du *session-hash*
- introduction de secrets intermédiaires
  - pour protéger une partie de la négociation
  - pour découpler le secret de reprise de session du secret courant

TLS 1.3 : Du nouveau dans la dérivation des clés

Une dérivation entièrement repensée

- utilisation du *session-hash*
- introduction de secrets intermédiaires
  - pour protéger une partie de la négociation
  - pour découpler le secret de reprise de session du secret courant
- tout cela risque de changer avec le mode 0-RTT

Conclusion

- Beaucoup d'attaques depuis 2011 sur TLS
- TLS 1.3 devrait apporter quelques bonnes réponses
  - *Forward Secrecy*
  - chiffrement authentifié propre des *records*
  - nettoyage de la phase de négociation
- Quelques inquiétudes demeurent
  - gestion de la transition et de TLS 1.0 à 1.2
  - discussions en cours sur 0-RTT
  - TLS 1.3 pourrait être plus complexe que ce qui a été présenté
  - TLS manque de tests de non-régression !

Questions ?

Merci de votre attention.