Wombat: one more Bleichenbacher attack toolkit

Olivier Levillain Aina Toky Rasoamanana

Télécom SudParis

SSTIC 2020

Quelque rappels sur RSA et sur l'attaque de Bleichenbacher

Rappels sur RSA

Algorithme de chiffrement asymétrique (et de signature)

- clé publique $N = p \cdot q$, e
- clé privée d
- chiffrement brut : $C = M^e[N]$
- déchiffrement brut : $C^d = M^{ed} = M[N]$

Rappels sur RSA

Algorithme de chiffrement asymétrique (et de signature)

- clé publique $N = p \cdot q$, e
- clé privée d
- chiffrement brut : $C = M^e[N]$
- déchiffrement brut : $C^d = M^{ed} = M[N]$

Besoin d'un schéma de bourrage (*padding*)

- faiblesses de ces opérations brutes (p. ex. : la malléabilité)
- formatage des messages avant de les chiffrer/signer avec PKCS#1

Rappels sur RSA

Algorithme de chiffrement asymétrique (et de signature)

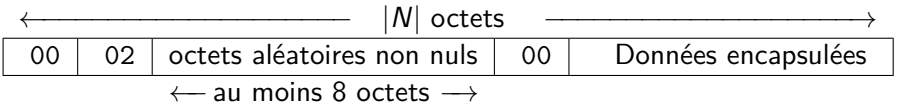
- clé publique $N = p \cdot q$, e
- clé privée d
- chiffrement brut : $C = M^e[N]$
- déchiffrement brut : $C^d = M^{ed} = M[N]$

Besoin d'un schéma de bourrage (*padding*)

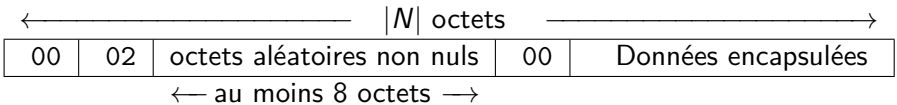
- faiblesses de ces opérations brutes (p. ex. : la malléabilité)
- formatage des messages avant de les chiffrer/signer avec PKCS#1

Étudions le *padding* de type 2 (PKCS#1 v1.5), utilisé pour le chiffrement.

Le *padding* de type 2

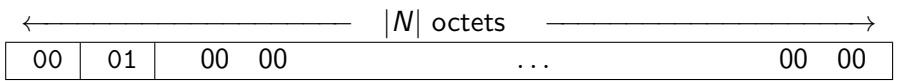


Le *padding* de type 2



Soit M_{pad} un message correctement formaté

- M_{pad} commence par les octets 00 02
- si on interprète M_{pad} comme un entier, $2B \leq M_{pad} < 3B$, où B est représenté ainsi :



Principe de l'attaque de Bleichenbacher (CRYPTO 1998)

Supposons qu'il existe un oracle

- un acteur qui accepte de déchiffrer nos messages
- il renvoie vrai si le *padding* est correct, faux sinon
- (le message déchiffré n'est pas révélé)

Principe de l'attaque de Bleichenbacher (CRYPTO 1998)

Supposons qu'il existe un oracle

- un acteur qui accepte de déchiffrer nos messages
- il renvoie vrai si le *padding* est correct, faux sinon
- (le message déchiffré n'est pas révélé)

L'attaquant souhaitant retrouver $m = c^d$ peut alors

- envoyer des messages altérées de la forme $c \cdot s^e$ (avec s connu)
- laisser l'oracle calculer $(c \cdot s^e)^d = c^d \cdot s^{ed} = ms$
- déduire que $2B \leq ms < 3B$ lorsque l'oracle retourne vrai
- répéter la manipulation et retrouver m à partir des équations obtenues

Les oracles en pratique

L'attaquant souhaite découvrir tous les messages qui commencent par `00 02`

Cependant, les implémentations de RSA peuvent faire d'autres vérifications, et avoir un comportement moins idéal

- le bourrage doit contenir au moins 8 octets
- le bourrage doit se terminer par un octet nul
- le message décapsulé doit avoir une longueur donnée

Dans ce cas, rater des bons messages (qui commencent par `00 02`) revient à *perdre* des inégalités intéressantes pour l'attaque

Bleichenbacher, une réalité en 2020 ?

Publications récentes autour des applications de Bleichenbacher à TLS

- 2014 – Meyer et al. : oracles dans l'implémentation JSSE
- 2016 – Aviram et al. : DROWN (SSLv2 sert d'oracle + bugs dans OpenSSL)
- 2018 – Bock et al. : ROBOT (Return of Bleichenbacher's Oracle Threats)
- 2019 – Ronen et al. : CAT (Cache-like ATtacks)

Wombat : one more Bleichenbacher toolkit

Principe de fonctionnement de Wombat

Pour tester une implémentation, il faut écrire un *stub* (connecteur), qui permet de

- récupérer la clé publique RSA de la cible
- produire un message chiffré à retrouver (*challenge*)
- soumettre des messages à déchiffrer

Principe de fonctionnement de Wombat

Pour tester une implémentation, il faut écrire un *stub* (connecteur), qui permet de

- récupérer la clé publique RSA de la cible
- produire un message chiffré à retrouver (*challenge*)
- soumettre des messages à déchiffrer

L'attaquant peut alors soumettre différents chiffrés correspondant à des messages clairs connus

- messages bien formés
- messages ne commençant pas par 00 02
- messages avec un *padding* trop court
- ...

Principe de fonctionnement de Wombat

Pour tester une implémentation, il faut écrire un *stub* (connecteur), qui permet de

- récupérer la clé publique RSA de la cible
- produire un message chiffré à retrouver (*challenge*)
- soumettre des messages à déchiffrer

L'attaquant peut alors soumettre différents chiffrés correspondant à des messages clairs connus

- messages bien formés
- messages ne commençant pas par 00 02
- messages avec un *padding* trop court
- ...

Si les observations permettent d'identifier à coup sûr des *bons* messages, il y a un oracle !

Exemple avec un serveur de type *root me*

Le serveur S fonctionne de la manière suivante :

- lorsqu'un client se connecte, il envoie sa clé publique, et un message chiffré avec PKCS#1 v1.5
- le client peut alors soumettre au serveur des messages à déchiffré

Écriture d'un *stub* avec Wombat

```
def __init__(self, host, port):
    """Creation de la socket et recuperation des infos"""
    WStub.__init__(self, None)
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket.connect((host, port))
```

Écriture d'un *stub* avec Wombat

```
def __init__(self, host, port):
    """Creation de la socket et recuperation des infos"""
    WStub.__init__(self, None)
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket.connect((host, port))

    public_infos = self.socket.recv(16384).split(b"\n")
    n = int(public_infos[0], 16)
    e = int(public_infos[1], 16)
    self.set_public_key(n, e)
    self.challenge = codecs.decode(public_infos[2], "hex")
```

Écriture d'un *stub* avec Wombat

```
def __init__(self, host, port):
    """Creation de la socket et recuperation des infos"""
    WStub.__init__(self, None)
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket.connect((host, port))

    public_infos = self.socket.recv(16384).split(b"\n")
    n = int(public_infos[0], 16)
    e = int(public_infos[1], 16)
    self.set_public_key(n, e)
    self.challenge = codecs.decode(public_infos[2], "hex")


def do_decrypt(self, c):
    """Soumission d'un chiffre dans la socket"""
    self.socket.send(codecs.encode(c, "hex"))
    return self.socket.recv(16384).strip()
```

Identification et exploitation de l'oracle

```
stub = Stub(("127.0.0.1", 12345))
oracle_type, true_signals, false_signals = identify_oracle(stub)
```

Identification et exploitation de l'oracle

```
stub = Stub(("127.0.0.1", 12345))
oracle_type, true_signals, false_signals = identify_oracle(stub)
```

```
o = OracleFromStub(stub, true_signals, false_signals)
attack = DecryptionAttack(o)
msg = attack.run(stub.get_challenge())
```

Identification et exploitation de l'oracle

```
stub = Stub(("127.0.0.1", 12345))
oracle_type, true_signals, false_signals = identify_oracle(stub)
```

```
o = OracleFromStub(stub, true_signals, false_signals)
attack = DecryptionAttack(o)
msg = attack.run(stub.get_challenge())
```

Démo

Applications (1/2)

Enseignement

- illustration de l'attaque de Bleichenbacher
- travaux pratiques possibles (attaque d'un serveur vulnérable)

Applications (1/2)

Enseignement

- illustration de l'attaque de Bleichenbacher
- travaux pratiques possibles (attaque d'un serveur vulnérable)

Résolution de défis de type `rootme`

- tutoriel dans doc/tuto-rootme.md

Applications (1/2)

Enseignement

- illustration de l'attaque de Bleichenbacher
- travaux pratiques possibles (attaque d'un serveur vulnérable)

Résolution de défis de type `rootme`

- tutoriel dans doc/tuto-rootme.md

TLS

- outil de test fonctionnel `wombat-tls`
- tutoriel pour comprendre comment appliquer Wombat à des serveurs TLS
- mise en évidence d'oracles sur des serveur du TopAlexa 1M

Applications (2/2)

XML Encryption

- projet étudiant en cours
- mise en évidence d'un oracle dans `xmlsec1`
- code en cours d'intégration

Applications (2/2)

XML Encryption

- projet étudiant en cours
- mise en évidence d'un oracle dans `xmlsec1`
- code en cours d'intégration

OpenPGP

- preuve de concept sordide
- présence probable d'un oracle dans gpg
- code à écrire

Conclusion

Conclusion

L'attaque de Bleichenbacher (*million-message attack*)

- une attaque connue depuis longtemps
- un exemple non trivial d'attaque cryptographique
- une réalité encore aujourd'hui ?

Wombat

- un outil libre pour reproduire des attaques existantes
- application actuelle : TLS, XML Encryption, OpenPGP (pas encore intégré)
- extension possible à d'autre protocoles via des *stubs*
- extension en cours à des attaques symétriques (CBC Padding Oracle)

Questions ?

Merci pour votre attention

`https://gitlab.com/pictyeye/wombat`