SSE - Passive SSH Key Compromise via Lattices - Ferrara Justin

Prérequis pour que l'attaque réussisse

- Posséder une partie d'un message signé avec sa signature invalide ???
- Posséder la clé publique correspondante
- L'algorithe étudié ici concerne les signature RSA PKCS#1 V1.5
- Les signatures sont réalisées en utilisant le théorème des restes chinois
- L'erreur dans la signature provient des opérations faite dans le monde des tuples (théorème des restes chinois)

> Pour faire cette attaque sur TLS, il faut faire une écoute passive sur TLS 1.2 et une attaque active sur TLS 1.3.

SSH

Authentification et échange de clés

Authentification du serveur

- Les serveurs SSH sont identifiés par leurs clés publiques.
- L'échange commence par une négociation de chiffrement suivie d'un échange de clés Diffie-Hellman.
- Le serveur s'authentifie en signant le session identifier avec sa clé privée, vérifié ensuite par le client.
- Le session identifier contient le D-H, l'ID du client et l'ID du serveur.

Authentification du client

- Se déroule après l'établissement du canal chiffré.
- **Méthodes courantes :**
  - **Mot de passe** : envoyé en clair dans le canal chiffré.
  - **Clé publique** : le client signe un identifiant de session avec sa clé privée.

Algorithmes cryptographiques

- Échange de clés : Diffie-Hellman, Elliptic Curve Diffie-Hellman (ECDH), RSA.
- Signatures : DSA, RSA, ECDSA, Ed25519.
- OpenSSH 8.8 (septembre 2021) désactive par défaut `ssh-rsa` (SHA-1) mais supporte `rsa-sha2-256` et `rsa-sha2-512`.

Sécurité et attaques possibles

Compromission des clés de signatures

- Ne permet pas de déchiffrer les connexions de manière passives mais peut permettre une attaque active (Man-in-the-middle).
- Un attaquant peut usurper l'identité du serveur et établir une connexion chiffrée avec le client pour obtenir son mot de passe par exemple.

Exploitation des méthodes d'authentification

- **Mot de passe** : un attaquant MITM peut intercepter et relayer les identifiants.
- **Clé publique** : plus sécurisé car ne permet pas un Man-in-the-middle. Permet tout de même de connaitre les commandes que le client envoie au serveur.

SSH Agent Forwarding ???

- Permet de transmettre l'authentification via un agent SSH distant.
- Peut être exploité par un attaquant pour se connecter à d'autres serveurs avec les clés de l'agent.
- OpenSSH 8.9 (février 2022) introduit des restrictions pour limiter ce risque, mais leur adoption reste limitée.

IPsec

IPsec est un ensemble de protocoles (RFC 2408, 2409, 7296) visant à garantir la confidentialité, l'intégrité des données et l'authentification des sources des paquets IP. Il est beaucoup utilisé par les VPN et repose sur le protocole Internet Key Exchange (IKE) pour négocier les algorithmes utilisés, définir comment dériver les clés, etc.

IKE : Versions et Fonctionnement

IKE existe en deux versions : **IKEv1** et **IKEv2**.

- **Établissement d'une Security Association (SA)** : choix des suites de chiffrement et échange initial de clés Diffie-Hellman.
- **Authentification mutuelle** entre l'initiateur et le répondeur via différentes méthodes (signatures, clés pré-partagées, etc.).

IKEv1

IKEv1 prend en charge trois modes d'authentification :

- Signatures numériques
- Chiffrement de clé publique
- Clé pré-partagée (PSK)
  Il y a deux modes de communication :

1. **Main Mode** : Plus sécurisé, toutes les communications sont chiffrées après l'échange initial.
2. **Aggressive Mode** : Échange réduit mais moins sécurisé (les signatures sont envoyées en clair).

Vulnérabilités dans IKEv1 ???

- **Attaque passive sur Aggressive Mode** : Un attaquant peut capturer une signature envoyée en clair en écoutant passivement la communication.
- **Format non standard de signature RSA** : RFC 2409 impose une variation du format PKCS#1 v1.5, supprimant l'OID du hachage car il est inclus dans la SA.
- **Compromission des clés de signature** : Un attaquant obtenant la clé privée de signature pourrait usurper cette identité. Pour pouvoir faire une attaque Man-in-the-middle, il faut compromettre la clé privée des deux parties à cause du D-H qui est signé.

IKEv2

IKEv2 n'est pas compatible avec IKEv1 :

- Toutes les signatures sont chiffrées et transmises dans l'AUTH payload après l'établissement de la SA.
- Extensible Authentication Protocol (EAP) permet d'obtenir de manière active une signature de l'host distant sans s'authentifier.
- L'authentification par clés cryptographique peut ne concerner qu'un participant en fonction des modes utilisés.

Vulnérabilités dans IKEv2

- **Compromission des clés de signature** : Un attaquant obtenant la clé privée de signature pourrait usurper cette identité. Pour pouvoir faire une attaque Man-in-the-middle, il faut compromettre la clé privée des deux parties.
- **Attaques sur l'authentification par clé pré-partagée (PSK)** : Si la PSK est faible (mot de passe faible), un attaquant peut mener une attaque par dictionnaire hors ligne et ensuite, avec la clé privé de signature de l'autre partie, faire un Man-in-the-middle complet.
- **Attaque MITM avec EAP** : Certains modes EAP (comme **EAP-MS-CHAPv2**) permettent une attaque Man-in-the-middle complète, en attaquant le hash du mot de passe utilisé par une attaque hors ligne.

PKCS#1 V1.5 padding pour signature RSA

Le padding PKCS#1 V1.5 utilisé par les signatures est donné par :

$$00||01||FF...FF||ASN.1||Hash(m)$$

Où :

- `ASN.1` est un identifiant pour définir la fonction de hachage utilisée
- `Hash(m)` est le hash du message `m`

Expression du padding

Pour le padding d'un message `m` inconnu, le padding ISO/IEC 9796-2 peut être exprimé de la manière suivante :

$$a + b * x + c * y$$

Avec `a`, `b` et `c` connu et `x` et `y` inconnu. Si une erreur de calcul survient `mod q` et mais que `mod p` est effectué correctement, nous pouvons en déduire l'équation suivante :

$$s'^e = a + b * x + c * x \mod p$$

Pour le padding PKCS#1 V1.5 d'un message, l'équation originale peut être simplifiée de la manière suivante :

$$a + x$$

$$x < 2^{hash\_len}$$

Avec `a` la partie connue du padding et `x` representant la partie inconnue du message, bornée par la taille de sortie de la fonction de hachage. En posant cette équation, il est possible de retrouver la clé privée de signature si `hash_len * 4 >= N`. ???

RSA Signatures avec padding PKCS#1 V1.5

La signature `s` RSA d'un message `m` sans padding est donnée par :

$$s = f(m)^d \mod N$$

La vérification de cette signature est donnée par :

$$f(m') = s^e \mod N$$

$$f(m) = f(m')$$

Avec `f(m)` une fonction de hachage définie par le padding.

> À noter que dans ce cas la signature doit être faite en utilisant le théorème des restes chinois.

PACD (Partial Approximate Common Divisors)

PACD est une généralisation du PGDC :

$$N_i = p \cdot q_i$$

Avec `p` un facteur inconnu de longueur `log p`.

Pour pouvoir résoudre le problème PACD, il faut que les diviseurs communs approximatifs soient suffisamment proches pour que l'algorithme LLL puisse les trouver. Nous pouvons donc essayer de poser les équations suivantes :

$$N_0 = p_0 \cdot q_0$$

$$N_1 = p_1 \cdot q_1 + r_1$$

Avec `r_1` une erreur de calcul et `r` l'espace de la fonction de hachage.

$$|r_1| < 2^{log_2(r)}$$

Comme nous n'avons pas `N1`, nous pouvons essayer de le trouver en utilisant les équations qui définissent la padding PKCS#1 V1.5. ???

$$s'^e = m' = a + h \mod p$$

$$s'^e \neq a + h \mod q$$

$$s'^e = k * p + a + h \mod N$$

$$(s'^e - a \mod N) = k * p + h \mod N$$

$$h \leq 2^{r-1}$$

Nous pouvons donc déduire les équations suivantes :

$$N_0 = p \cdot q_0$$

$$N_1 = (s'^e - a \mod N) = k \cdot p + h$$

Avec `h` une valeur inconnue et `a` la partie connue du padding PKCS#1 V1.5.

Lattice

Nous pouvons ensuite remplir la matrice suivante :

$$B = \begin{bmatrix} -2^{2\log r} & 2^{\log r} N_1 & 0 \\ 0 & -2^{\log r} & N_1 \\ 0 & 0 & N_0 \end{bmatrix}$$

Avec `r` l'espace de la fonction de hachage.

LLL

Pour réduire la matrice précédente, nous pouvons utiliser l'algorithme LLL pour essayer de trouver des vecteurs avec une base plus courte et plus proche de l'orthogonalité que la base d'entrée:

- appliquer la réduction de Gram-Schmidt pour décomposer chaque vecteur
- modifier un peu les vecteurs en les ajustant, en remplaçant certains vecteurs par des combinaisons linéaires des autres vecteurs
- vérifie la condition de Lovász pour chaque vecteur

À la fin de l'algorithme, on obtient une base plus courte et plus proche de l'orthogonalité que la base d'entrée. Si la réduction a fonctionné, nous pouvons retrouver un vecteur `v` qui peut être interprété comme les coefficients du polynôme suivant :

$$\overrightarrow{\mathrm{v}}$$

$$g(2^{\log r} x)$$

Si les coefficients sont assez petits, on peut poser les équations suivantes :

$$|g(y)| < p^k$$

$$|y| \leq 2^{\log r}$$

$$g(y) = 0$$

$$y = r_1$$

Avec `r_1` une petite racine mod p.

Comme nous connaissons `r_1`, nous pouvons donc déduire `p` :

$$p = \gcd(N_0, N_1 - r_1)$$

> Note : LLL trouvera une solutions seulement si `hash_len * 4 >= N`.

Mesures de protections

- **Validation des signatures** : Il vérifier que la signature effectuée ne contient pas d'erreur avant de l'envoyer et dans le cas contraire, en refaire une nouvelle.
- **RSA dans SSH** : Éviter d'utiliser des versions vulnérables ou faibles du padding PKCS#1 V1.5 avec RSA (SHA-1).
- **Design du protocol** :
  - Chiffrer la communication le plus tôt possible dès que les clés cryptographiques sont disponibles pour protéger les metadata
  - Définir une authentification par session et les lier ensemble
  - Séparer l'authentifiaction des clés cryptographiques

Bibliography

Keegan Ryan, Kaiwen He, George Arnold Sullivan, Nadia Heninger. *Passive SSH Key Compromise via Lattices*. Cryptology ePrint Archive, Paper 2023/1711, 2023. DOI: 10.1145/3576915.3616629, URL: https://eprint.iacr.org/2023/1711.