Détection automatique des lignes de caches exploitables

Théophile Wallez

5 Septembre 2017

Préliminaires

Préliminaires
- Le cache
- Les librairies partagées
- Exploitation d'une fuite d'information par le cache
- But du stage

Analyse d'une librairie partagée
- Détection des adresses utilisées
- Détection des adresses qui fuitent de l'information
- Détection des ensembles d'adresses qui fuient de l'information exploitable

Conclusion

Qu'est-ce que la SRAM aka. le cache

- Mémoire plus rapide que la DRAM mais plus chère et consomatrice d'énergie
- Environ 1000x plus petite que la DRAM
- Sert à mettre en cache les données souvent utilisées
- Sur les processeurs Intel, il n'est pas trop faux d'admettre que le cache est partagé entre les différents cœurs

Manipulation du cache

- Mettre une adresse dans le cache : y accéder
- Tester la présence d'une adresse dans le cache : mesurer le temps qu'on met pour y accéder avec l'instruction `rdtsc`
- Enlever une adresse du cache : utiliser l'instruction `clflush`

Une grave conséquence : l'attaque Flush+Reload

On peut alors construire un oracle qui dit si une adresse a été accédée entre un temps t_1 et t_2

- Au temps t_1, enlever l'adresse attaquée du cache (flush)
- Au temps t_2, mesurer son temps d'accès (reload)

Une grave conséquence (bis) : l'attaque Flush+Flush

On peut faire mieux :

- Au temps t_1, enlever l'adresse attaquée du cache (flush)
- Au temps t_2, mesurer le temps de l'instruction `clflush` (flush)

C'est plus rapide, on peut le pipeliner, c'est plus discret

Comment l'OS gère les librairies partagées

- L'OS mappe le code de la librairie partagée en mémoire virtuelle avec `mmap`
- Quand différents programmes mappent le même fichier en mémoire, l'OS en garde une seule copie en mémoire physique
- Grave conséquence : la mise en cache des adresses d'une librairie partagée est partagée entre les différents processus.

Que se passe-t-il si la librairie partagée est une librairie de cryptographie ?

Exponentiation rapide en temps constant

Pseudo-code d'une fonction de OpenSSL 1.0.2, jusqu'au 12 Mars 2014

Algorithme 1 : Montgomery Ladder

Data : $d = d_n \dots d_{1[2]} \in \mathbb{N}$, $x \in G$
Result : $R_0 = x^d$
$R_0 \leftarrow 0$
$R_1 \leftarrow x$
/* Invariant 1 : R_1 = mult(x, R_0) */
/* Invariant 2 : $R_0 = x^{d_n \dots d_{i[2]}}$ at the end of loop i */
for $i \leftarrow n$ **to** 1 **do**
  if $d_i = 0$ **then**
    $R_1 \leftarrow$ mult(R_0, R_1)
    $R_0 \leftarrow$ square(R_0)
  else
    $R_0 \leftarrow$ mult(R_0, R_1)
    $R_1 \leftarrow$ square(R_1)
  end
end


SO CONSTANT-TIME
SUCH SECURE

Pas si sécurisé que ça ?

- Le code du if et du else sont dans des lignes de cache différentes.
- Avec notre oracle, cela permet de retrouver tous les bits de l'exposant, comme décrit dans **Yarom2014**

Exponentiation rapide en temps constant – version sécurisée

Un branchement conditionnel ou l'indice d'accès d'un tableau ne doit pas dépendre du secret (prouvé dans **Barthe2014**)

Algorithme 2 : SecureConditionalSwap

Data : $(a, b) \in \mathbb{N}^2$, $c \in \{0, 1\}$
Result : (a, b) si $c = 0$ ou (b, a) si $c = 1$
`/* On exploite le fait que -1 est représenté avec tous ses bits à 1 */`
$t \leftarrow (a \oplus b) \wedge (-c)$
$a \leftarrow a \oplus t$
$b \leftarrow b \oplus t$

Algorithme 3 : Secure Montgomery Ladder

Data : $d = d_n \dots d_{1[2]} \in \mathbb{N}$, $x \in G$
Result : $R_0 = x^d$
$R_0 \leftarrow 0$
$R_1 \leftarrow x$
for $i \leftarrow n$ **to** 1 **do**
  `SecureConditionalSwap(`$(R_0, R_1), d_i$`)`
  $R_1 \leftarrow$ `mult(`R_0, R_1`)`
  $R_0 \leftarrow$ `square(`R_0`)`
  `SecureConditionalSwap(`$(R_0, R_1), d_i$`)`
end

But du stage

- Trouver automatiquement les adresses qui fuitent de l'information **exploitable**.
- Cas d'étude : l'implémentation de OpenSSL de ECDSA sur la courbe secp256k1 (exponentiation *w*NAF)

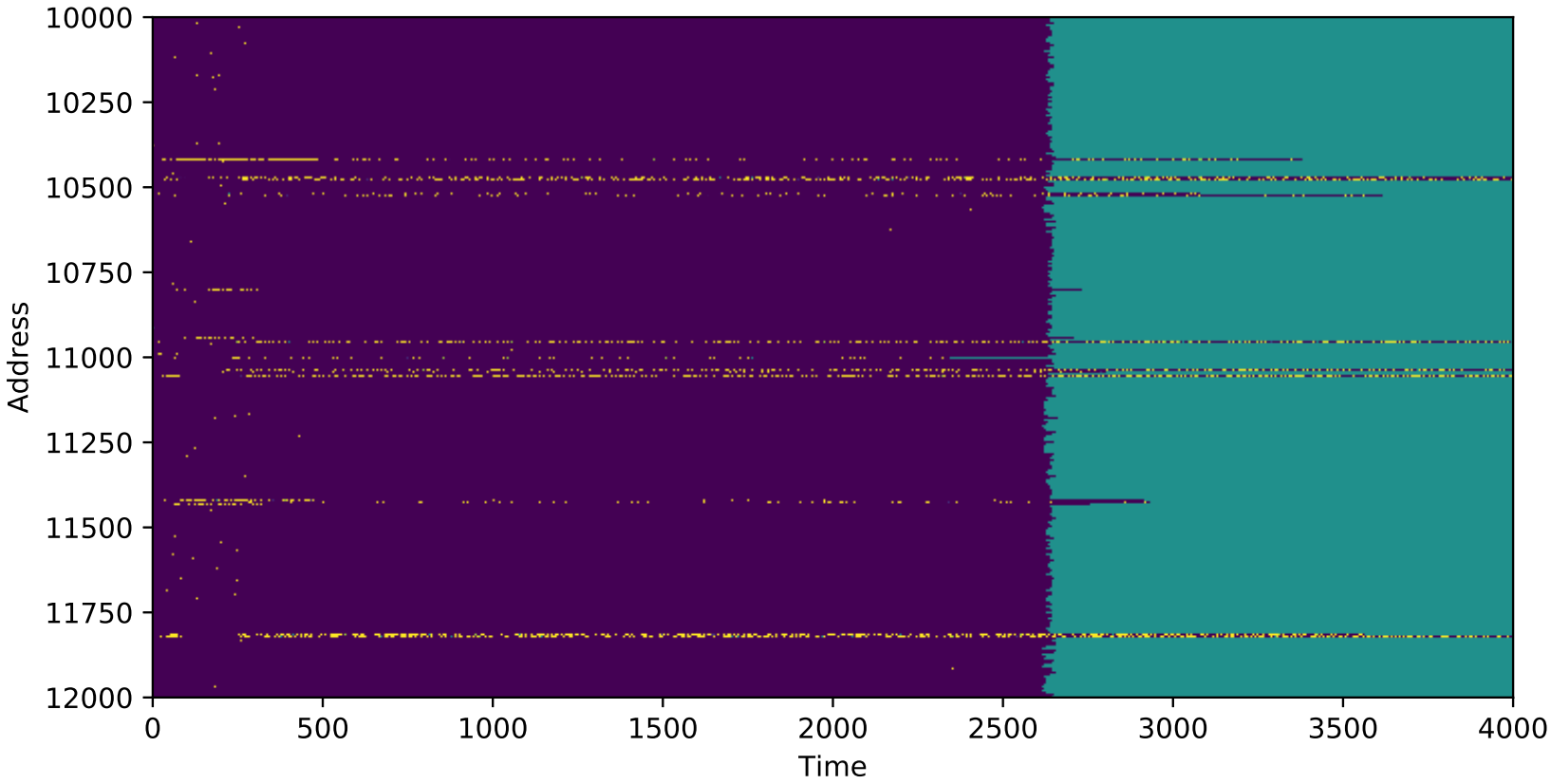
Analyse d'une librairie partagée

Préliminaires
- Le cache
- Les librairies partagées
- Exploitation d'une fuite d'information par le cache
- But du stage

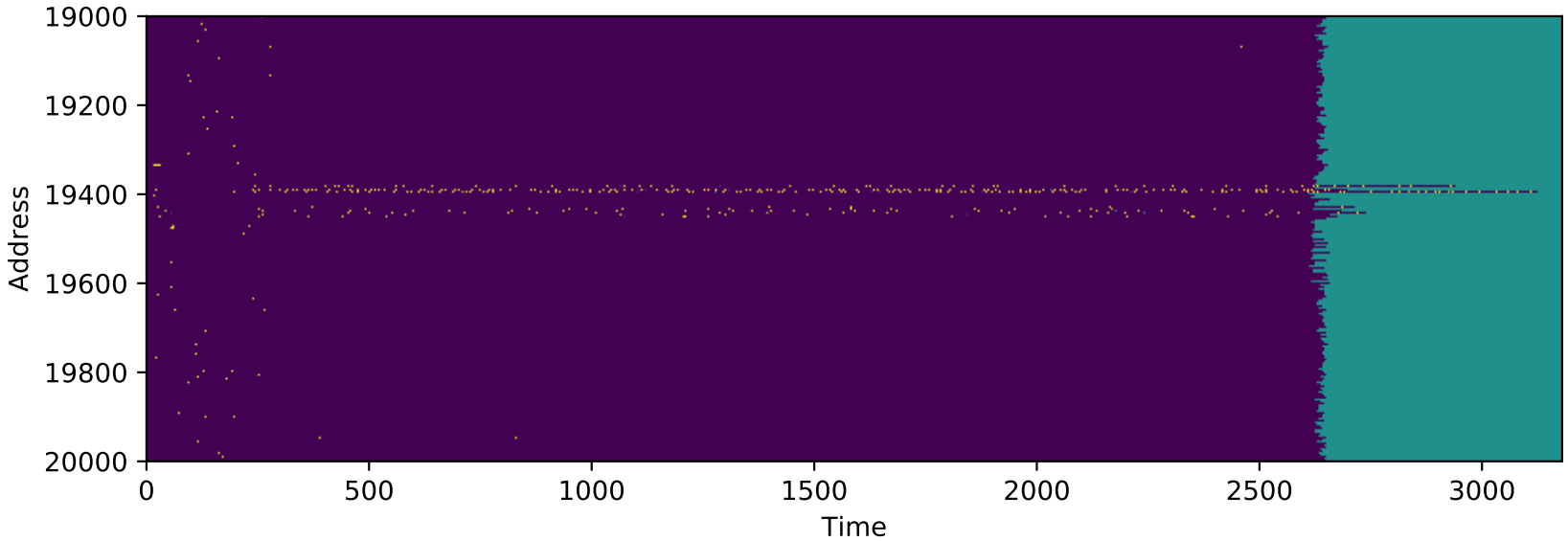
Analyse d'une librairie partagée
- Détection des adresses utilisées
- Détection des adresses qui fuitent de l'information
- Détection des ensembles d'adresses qui fuient de l'information exploitable

Conclusion

Encéphalogramme d'une librairie partagée : Big Numbers



Encéphalogramme d'une librairie partagée : addition et doublement sur courbe elliptique



Premier élagage des adresses

Par la suite, on ne considère que les k adresses les plus actives.
(nous avons pris $k = 200$)

Méthode pour détecter la fuite d'information

Principe admis : Si une adresse fuit de l'information exploitable, alors elle permet de facilement reconnaitre un secret parmi un groupe restreint de secrets.

- Pour m secrets différents, on génère n traces (en pratique : $m = 10$, $n =$ autant que possible)
- On regarde la performance de l'algorithme des k plus proches voisins sur cette adresse

Quelle distance choisir

Problème : on ne peut pas comparer point à point
Méthode 1 : utiliser DTW (dynamic time warping)

- Corrige les translations / dilatations mais ne les quantifie pas
- Permet de retrouver `EC_POINT_add` mais pas `EC_POINT_dbl`

Méthode 2 : comparer les basses fréquences de la transformée de Fourier

- Corrige les translations / dilatations en les quantifiant
- Permet de retrouver `EC_POINT_add` et `EC_POINT_dbl`

Deuxième élagage des adresses

Par la suite, on ne considère que les k adresses qui fuitent le plus d'information.
(nous avons pris $k = 20$)

Détection du premier bit de la clé

On cherche à déterminer le premier bit de la clé avec du machine learning.
On a pour cela utilisé Keras et scikit-learn.
On arrive à avoir 75% de réponses correctes avec deux adresses sélectionnées à la main.

Conclusion

Préliminaires
- Le cache
- Les librairies partagées
- Exploitation d'une fuite d'information par le cache
- But du stage

Analyse d'une librairie partagée
- Détection des adresses utilisées
- Détection des adresses qui fuitent de l'information
- Détection des ensembles d'adresses qui fuient de l'information exploitable

Conclusion

Axes de recherche / travail futur

75%, ce n'est pas assez pour mener une attaque réaliste.
Cela montre quand même que les adresses sélectionnées fuient une réelle information.
On pourrait donc avoir un nouvel oracle qui restreint encore plus le nombre d'adresses intéressantes.

Conclusion

Les autres approches qu'on a trouvé pour le problème sont par analyse statique → bien du point de vue du défenseur.
L'approche dévelopée pendant le stage est dynamique, et découvre spécifiquement ce qui est utile pour une attaque.