Hardware-based Attacks and Countermeasures

1. Setting up Reading Primitivies

1.1. Time and Security

Il tempo richiesto per eseguire una operazione rileva informazioni importanti sull'operazione e sui dati correlati. Nell'esempio, confrontiamo due stringhe, appena due char sono diversi esco. Ovviamente, a seconda delle stringhe, esco prima o dopo nel ciclo, possiamo osservare differenze di tempo!

1.2. Timing a Cache

Il fattore temporale influisce molto sulla cache, la quale sappiamo essere molto veloce. Se troviamo un modo di avere la cache in un certo stato, possiamo prendere delle tempistiche. Il livello più basso di cache è condiviso.

1.3. Discovering code paths

L'algoritmo Montgomery è usato nelle curve ellittiche, e quindi in casi come AES. Abbiamo *n* scalare, è un *nonce*, usato solo una volta, cambia ad ogni iterazione. Abbiamo un loop, che ci permette di prendere il timing. Dentro un if/else, per ogni bit. Con più di 64 bytes, i due casi di if else vengono mappati su >2 linee di cache. Se c'è un miss, ci vuole più tempo per caricare l'istruzione. Ma quale blocco dei due if/else? Se il primo blocco è lento, il valore è 0, se il secondo blocco è lento, il valore è 1. Per questo motivo AES è stato rotto, perchè si eseguono "grossi blocchi" a seconda dell'esito dell'if. Prima cancelliamo le 4 linee di cache (due per blocco, ma possiamo cancellarne anche solamente una per blocco). quando ne verrà ricaricata una possiamo prendere il tempo (ad esempio, viene caricata una linea cache del blocco *if* quando è *true*). Ora, se successivamente misuriamo un accesso della stessa latenza, abbiamo nuovamente acceduto allo stesso blocco, se la latenza è significativamente diversa, allora è l'altro blocco.

1.4. Side Channel Attacks

Memoria che permette di leggere altre memorie o di individuare pattern di accesso. Metto la cache in uno stato noto, e poi osservo se cambia tale stato.

Nel pre attack acquisisco il target (singola linea di cache, o set), e definisco delle soglie se necessarie. Poi metto la cache in stato noto, aspetto che la vittima faccia un accesso in memoria, analizzo tale accesso, vedendo la traccia della vittima, ripeto fino a che il dato risulti leakato. Dobbiamo ricordare che le cache possono essere sia virtuali sia fisiche, possono essere condivise (in base al livello), e possono essere interconnesse tra processi.

1.5. Evict + Time

Sfruttiamo il fatto che, se faccio reload della cache (cancellata), ci mette più tempo. Possiamo capire quale aree risultano usate. Notoriamente, dati acceduti recentemente o in uso sono in cache. A questo punto, cancellando una linea di cache di interesse. Se la vittima riaccede, avremo un delay di tempo maggiore, rispetto alle altre linee non toccate, dandoci l'informazione di quale zona è stata acceduta nuovamente. Permette di trovare *code paths*.