

Side-Channel Attack

Leonardo Costa Santos - 10783142

Lucas Paiolla Forastiere - 11221911

Julia Leite - 11221797

29 de novembro de 2020

1 Introdução

Side-Channel Attacks, que do inglês significa, Ataque por Canal Lateral são jeitos explorar vulnerabilidades físicas de componentes eletrônicos, como a CPU de um computador.

O nome vem do fato deles não atacarem ”pela porta da frente”, mas sim algum ”rastros” físico que um componente deixa ao fazer determinadas ações.

Um SCA não necessariamente tem a ver com componentes eletrônicos, pois podemos, por exemplo, decifrar uma senha de alguém captando os sons do teclado. Em geral, o SCA ataca um ponto fraco de um componente que não tem nada a ver com o seu funcionamento em si (como no exemplo do teclado, o teclado teoricamente não tem a responsabilidade de deixar os barulhos de cada tecla iguaizinhos). Daí então o nome *canal lateral*.

Os dois principais SCA, que tornaram o “ramo” famoso foram o *Meltdown* e o *Spectre*, descobertos por independentemente por uma série de pesquisadores, mas destacando-se o grupo Project Zero da Google [1].

Entretanto, existem muitas classes de SCA, como ataques ao cache (que é o caso dos dois exemplos citados), ataques que monitoram a energia consumida pelo computador, ataques que monitoram o eletromagnetismo emitido, ataques que monitoram o som emitido (como o exemplo do teclado), ataques que recuperam dados excluídos do disco entre muitos outros.

2 História

Primeira pessoa a usar o termo (Side Channel Cryptanalysis of Product Ciphers) Primeiro ataque descoberto (TEMPEST 1942) Popularização do termo Mais descobertas de ataques

3 Classificações de Side-Channel Attacks

Na classificação mais clássica de Side-Channel Attacks, temos que eles podem ser classificados quanto a duas categorias:

1. Invasivo *vs.* não-invasivo: Ataques invasivos são aqueles que abrem o aparelho sobre ataque. Um exemplo disso é conectar um cabo a um barramento do computador para ver os dados transferidos por aquele barramento. Os ataques invasivos por sua vez podem ser subdivididos em **completamente invasivos** ou **parcialmente invasivos**. Os parcialmente invasivos são aqueles que deixam o componente intacto após o ataque, enquanto os completamente invasivos são aqueles que danificam o componente para que o ataque possa ser realizado.
2. Ativo *vs.* passivo: Os ataques passivos são aqueles que se restringem a observar os comportamentos de um dispositivo para realizar o ataque, enquanto os ativos são aqueles que manipulam o dispositivo para que possam obter as informações desejadas. Eles podem fazer isso, por exemplo, injetando vários tipos de falhas elétricas, óticas, etc.

Além dessas classificações, pode-se classificar os side-channel attacks em classes diferentes. Entre elas há:

- **Ataques de tempo:** São o tipo de ataque mais clássico. Eles se baseiam na diferença de tempo na execução de algoritmos dependendo do dado. Por exemplo, considere um algoritmo que verifica se uma senha está correta.

```
1 bool check_password(char *passwd) {  
2     for (int i = 0; i < pass_len; i++)  
3         if (passwd[i] != stored_passwd[i])  
4             return false;  
5     return true;  
6 }
```

Observamos que o algoritmo devolve falso assim que descobre um caracter que não bate com o da senha e retorna verdadeiro caso todos os caracteres estejam corretos. Ou seja, caso se tente uma senha na qual o primeiro caracter esteja correto, o algoritmo `check_password` levará um pouco mais de tempo do que no caso em que o primeiro caracter esteja errado. Ao analisar, portanto, o tempo caso, obtem-se um método eficiente de descobrir qual é a senha desejada.

- **Ataques de análise de gasto energético:** Nesse tipo de ataque, o canal lateral observado é o gasto energético do dispositivo. Como as instruções são enviadas e executadas por cadeias de transistores, é possível obter um padrão característico de cada instrução.

Esses ataques são divididos em **simples** e **diferenciais**. O simples envolve uma análise manual significativa e ele pode ser evitado gerando gastos aleatórios de energia (por exemplo, inserindo ciclos extras que não fazem nada além colaborativo). No diferencial, entretanto, o atacante mede os gastos energéticos várias vezes e depois cria um modelo de gasto teórico de energia de um algoritmo com um pequeno número de bits chutados. Depois disso, o atacante cria modelos estatísticos para dizer quão perto os chutes estão próximos das respostas corretas. Devido ao alto número de medições e métodos estatísticos poderosos, essa abordagem pode conseguir resultados mesmo se medidas de proteção forem tomadas.

- Ataques eletromagnéticos:
- Ataques acústicos:
- Ataques óticos:
- Ataques de dados remanecente:
- Ataques de análise de falha:

4 Exemplos de Side-Channel Attacks

4.1 Meltdown

Meltdown é um SCA relacionado ao processador e a um efeito colateral da execução fora-de-ordem feita por ele. Graças a ela, o Meltdown consegue quebrar a hierarquia entre o *espaço do usuário* e o *espaço do núcleo*, podendo ler informações que não deveriam ser acessíveis por qualquer usuário, como senhas e dados pessoais.

Atualmente, o principal mecanismo de defesa de qualquer sistema operacional é a *isolação da memória*, dividindo a memória principal entre os diversos processos em andamento de forma que as regiões de memória em uso por um não possam ser acessadas por outros.

Para conseguir isso e outras propriedades importantes do Sistema Operacional, ele se utiliza da chamada *memória virtual*, que é uma abstração para a memória física. Essa memória virtual é dividida em páginas de memória que podem ser individualmente mapeadas em regiões da memória física através de uma *tabela de tradução de páginas*.

Essa tabela não só tem como função mapear as páginas de memória, mas também dividi-las entre as páginas que pertencem ao usuário e às que pertencem ao núcleo (e daí surgem os termos *espaço do usuário* e *espaço do núcleo*).

Através da tabela, o SO garante que o usuário não conseguirá acessar espaços de acesso restrito ao núcleo. Entretanto, o núcleo pode e deve ter acesso a toda a memória física em si (inclusive a parte em que se mapeam as páginas de usuário). Isso significa na prática que dentro da memória virtual do núcleo, existe uma região que mapeia toda a memória física, permitindo que o núcleo altere posições de memória do usuário quando ele faz chamadas ao sistema.

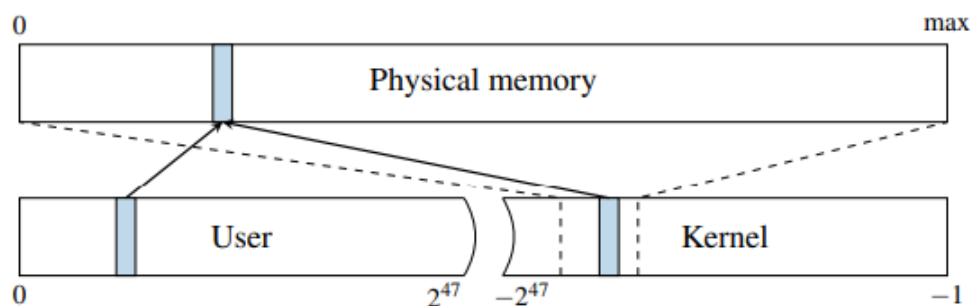


Figura 1: A memória física é completamente mapeada pelo núcleo em um certo ponto da sua memória virtual. Os endreços físicos (em azul) são mapeados pelo usuário, mas também pelo núcleo, sendo acessível pelos dois com eficiência.

Além disso, outro mecanismo crucial de defesa é a separação entre o *espaço do usuário* e o *espaço do núcleo*, ou seja, uma divisão entre quais processos (e quais partes da memória) pertencem a processos do usuário e quais pertencem ao sistema operacional em si.

Essa divisão é tipicamente realizada por um bit supervisor do processador que define se uma página de memória do núcleo pode ou não ser acessada. E a ideia por trás desse bit é que ele será mudado para 1 quando um processo precisa fazer chamadas ao sistemas (*syscalls*), portanto parando de executar código do usuário e passando a executar código do núcleo, e mudado novamente para 0 quando saímos do modo núcleo.

Esse bit é utilizado com uma ideia de eficiência, pois permite que o SO mapeie o núcleo no espaço de endereço do processo que fez a *syscall*. Consequentemente, não há nenhuma mudança no mapeamento da memória quando mudamos do modo usuário para o modo núcleo (o que é crucial para garantir mais velocidade).

O Meltdown explora uma vulnerabilidade causada pela *execução fora-de-ordem* para ler dados mapeados no espaço de endereço do núcleo, o que inclui a memória física inteira em sistemas Linux, Android e OS X e uma grande parte da memória física em ambientes Windows.

As CPUs modernas possuem essa técnica de otimização chamada de *execução fora-de-ordem* que permite que os núcleos passem mais tempo trabalhando, mesmo quando uma determinada operação precisa esperar algum recurso (por exemplo, trazer um valor da memória). Basicamente, o que acontece é que ao invés de a CPU executar as instruções sequencialmente, ela vai as executando assim que todos os recursos necessários para uma determinada instrução estiverem disponíveis.

Na prática, isso significa que a CPU *especula* que uma determinada instrução será executada no futuro e, então, faz a sua execução antes mesmo de ter certeza disso. O desenvolvedor do algoritmo que possibilitou a *execução fora-de-ordem* foi Tomasulo em 1967 [2].

A vulnerabilidade encontrada pelo Meltdown se deve ao fato de que ao tentar executar uma instrução de acessar uma região de memória que não pertence ao programa, a *execução fora-de-ordem* acabará fazendo o acesso e armazenando o valor no *cache*. Apenas depois que esse dado é armazenado no *cache*, a CPU percebe que a instrução não deveria ser executada e não de fato entrega esse valor ao programa que solicitou. Contudo, como o dado está em cache, o programa pode fazer um *ataque ao cache* para recuperar essa informação, acessando, portanto, uma região da memória que não pertence ao programa atacante.

4.2 Spectre

4.3 CacheOut

4.4 SGXxe

4.5 ZombieLoad

4.6 Foreshadow

5 Há jeito de se previmir?

6 Há como saber se estou sofrendo um SCA?

No. =(=

7 Conclusão

Referências

- [1] Moritz Lipp et al. “Meltdown: Reading Kernel Memory from User Space”. Em: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.
- [2] R. M. Tomasulo. “An Efficient Algorithm for Exploiting Multiple Arithmetic Units”. Em: *IBM Journal of Research and Development* 11.1 (1967), pp. 25–33. DOI: [10.1147/rd.111.0025](https://doi.org/10.1147/rd.111.0025).