Side-Channel Attack

Leonardo Costa Santos - 10783142
Lucas Paiolla Forastiere - 11221911
Julia Leite - 11221797

7 de dezembro de 2020

1 Introdução

Side-Channel Attacks, que do inglês significa Ataques por Canal Lateral, são jeitos explorar vunerabilidades físicas de componentes eletrônicos, como a CPU de um computador, através de mecanismos físico alheios à lógica do componente.

O nome vem do fato de ser uma técnica que não ataca "pela porta da frente", mas sim algum "rastro" físico que um componente deixa ao fazer determinadas ações. Como a energia gasta pelos transistores de uma CPU ao fazer uma soma.

Um SCA não necessariamente tem a ver com componentes eletrônicos, pois podemos, por exemplo, descifrar 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 idênticos). Daí então o nome *canal lateral*.

Para dar outro exemplo, suponha que uma pessoa está escrevendo em um papel com uma caneta e se deseja saber o que está escrito (uma senha, por exemplo). O modo mais direto é olhar para o papel escrito e ler o conteúdo, mas imagine que essa possibilidade não existe porque a pessoa está de costas para você. Suponha então que se cria um programa que, dado um vídeo de uma pessoa de costas escrevendo em um papel, consegue decifrar o que está escrito usando os movimentos do braço direito da pessoa. Isso seria um exemplo de ataque por canal lateral. Outros ataques, entretanto, são possíveis. Pode-se gravar o som e usá-lo para descifrar, ou então usar uma câmera que grava imagens em frequências de luz não visível. As possibilidades são vastas.

Trazendo para o mundo do computador e dos algoritmos, Daniel Genkin, cientista da computação na Universidade de Michigan e líder em pesquisas em side channel attacks, dá uma ótima explicação do que são eles ao dizer que geralmente quando se desenvolve um algoritmo, se pensa nas entradas e saídas e não em o que acontece quando o programa é executado. O problema é que computadores não são executados em papeis, mas sim em física, diz ele. [6]

Recentemente, dois SCAs trouxeram fama para o ramo ao revelar que praticamente qualquer computador pessoal moderno está sujeito a tais vulnerabilidades, independentemente do Sistema Operacional utilizado. São eles o *Meltdown* e o *Spectre*, descobertos independentemente por uma série de pesquisadores, destacando-se o grupo Project Zero da Google. [10]

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 eminido, ataques que monitoram o som emitido (como o exemplo do teclado), ataques que recuperam dados excluidos do disco entre muitos outros. Esta monografia se propõe como uma introdução ao assunto, explicando os diferentes tipos de ataques e exemplificando alguns deles mais a fundo.

2 História

Um dos mais notórios ataques análogo ao que hoje conhecemos como ataque por canais laterais, foi chamado, pela *National Security Agency* (NSA) de Tempest[11] [17], e ocorreu em 1943, quando pesquisadores do *Bell Lab* descobriram que, utilizando um osciloscópio, era possível recuperar 75% do texto da mensagem emitita por um *teletype machine* há 80 pés (aproximadamente 24.3m) de distância.

Entretanto, o termo *Side Channel Cryptanalysis* sugiu apenas em 1998, em um artigo [9] onde uma equipe formada por criptógrafos da Counterpane Systems e pesquisadores da *Universy of California at Berkley* descreveu como ataques por canais laterais podem ser usados para quebrar sistemas de criptografia.

Em 2014, pesquisadores da *Tel Aviv University* [4] desenvolveram um dispositivo capaz de descobrir senhas criptografadas de laptop's próximos motitorando sua emissão elétrica. Além disso, em agosto do mesmo ano, esse grupo de pesquisadores publicou um artigo mostrando que o padrão sons emitidos por um computador durante o processo de descriptografia, detectável por um microfone, varia de acordo com a chave RSA utilizada, apesar disso, não ficou claro como extrair os bits da chave RSA.[5]

Desde 2018, o termo *side channel attack* se popularizou graças à descoberta de vunerabilidades presentes na maioria dos processadores Intel fabricados nas últimas duas décadas, exploradas por SCA como Meltdown, Spectre, entre outros.

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 subdividos 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-chanell attacks em classes diferentes. Entre elas há:

- **Ataques de tempo:** São o tipo de ataque mais clássico. Eles se baseam 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    }
7
```

  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 significante 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 abordágem pode conseguir resultados mesmo se medidas de proteção forem tomadas.

- **Ataques eletromagnéticos:** Esses ataques se aproveitam de sinais eletromagnéticos que um dispositivo acaba emitindo. Um exemplo clássico foram os ataques TEMPEST, um dos primeiros tipos de Side Channel Attacks a serem estudados.

- **Ataques acústicos:** Eles se aproveitam de barulhos emitidos pelo computador para extrair informações importantes. Um exemplo bastante interessante é ouvir o barulho das teclas do teclado e deduzir quais foram as teclas pressionadas. (AI que deduz as teclas) Outro exemplo de ataque acústico é aquele que explora os pequenos sons emitidos por um computador devido ao estresse mecânico causado pelo calor. Esses sons podem ser captados e utilizados junto com alguma análise estatística para gerar informação. [16]

- **Ataques óticos:** Utilizando câmeras de alta resolução apontadas para partes específicas do computador é possível extrair informações de quais instruções o computador está executando. Por exemplo, podemos usar uma câmera infravermelha apontada para o processador e fazer um mapa de calor.

- **Ataques de dados remanecente:** Esses ataques conseguem ler dados mesmo depois de terem sido deletados. Muitas vezes, quando deletamos um arquivo utilizando

uma chamada para o sistema, a única mudança realizada pelo SO é marcar aquelas posições de memória do disco como livres, para que outros dados possam sobrescrevê-las no futuro. Isso abre, portanto, uma brecha para que se possa recuparar os dados supostamente deletados.

Outro exemplo é o chamado *cold boot attack*, em que um atacante com acesso físico ao computador performa um *dump* da memória RAM ao dar um *hard reset* da máquina. Ao contrário do que se pensa, a memória volátil não perde todos os dados assim que cortamos a energia do computador. Na verdade, alguns dados podem ficar até noventa minutos na memória após o computador ser desligado. Consequentemente, um atacante pode performar o chamado *cold boot*, ligando a máquina através de um sistema operacional especial pré-instalado em um USB, CD-ROM ou pela rede para que esses dados possam ser recuperados e utilizados pelo atacante para encontrar informação sensível. [7, 2]

- **Ataques de análise de falha:** Esse tipo de ataque introduz falhas propositalmente em mecanismos de criptografia para revelar como eles funcionam internamente. Um chip de cartão de crédito, por exemplo, pode ser exposto a altas temperaturas ou voltagens, a campos eletromagnéticos forte ou até mesmo a radiação para influenciar o processador na tentativa de deduzir quais são as instruções em execução.

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 divide 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 pertecem 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 mapea toda a memória física, permitindo que o núcleo altere posições de memória do usuário quando ele faz chamadas ao sistema.

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.

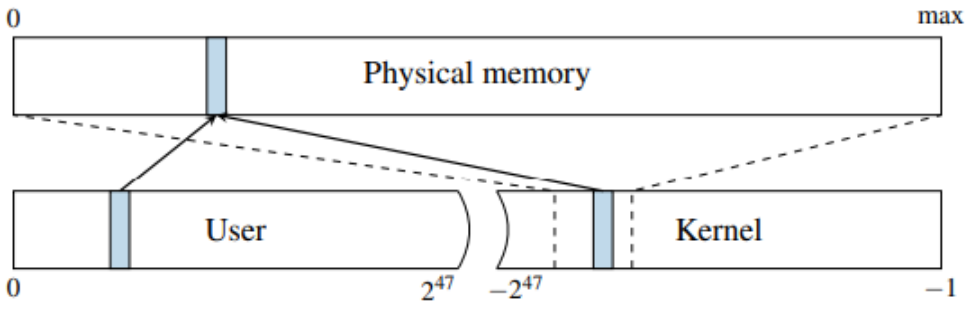

Figura 1: A memória física é completamente mapeada pelo núcleo em um certo ponto da sua memória virtual. Os endreços físicos (em azul) são mapeados pelo usuário, mas também pelo núcleo, sendo acessível pelos dois com eficiência.

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 vunerabilidade causa 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 sequenciamente, 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 determina 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 [15].

A vunerabilidade 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

CacheOut é um ataque de *Microarchitectural Data Sampling* (MDS), ou Amostragem de Dados Microarquitetural, capaz de evitar as medidas de segurança contra MDSs dos processadores da Intel. Este ataque é capaz de vazar dados atravéz de barreiras de segurança como o isomento de mémoria entre processos, entre *user/kernel spaces*, *SGX enclaves* e máquinas virtuais. Ao causar contenção de linhas de cache, CacheOut causa a expulsão de dados do cache e lê estes dados dos LFBs com um ataque TAA, conseguindo vazar páginas inteiras de memória. [12]

4.3.1 Line Fill Buffers

Line Fill Buffers (LFBs) são *buffers* microarquiteturais usados para armazenar dados durante acessos ao cache L1, tratando de pedidos ao cache em *cache misses* e temporariamente armazenando dados em acessos à memória e operações de I/O. Também podem ser usados em *cache hits* e para encaminhar dados para operações de leitura e escrita no cache. [8]

4.3.2 Transactional Synchronization Extensions

Transactional Synchronization Extensions (TSX) é uma implementação de transações de memória, que agrupa instruções em transacões executadas de modo atômico, executando todas as instruções da transação especulativamente e consolidando os resultados apenas após a execução de sua última instrução. Se alguma instrução causa uma *memory fault*, a transação inteira é descartada. [12]

4.3.3 TSX Asynchronous Abort

TSX Asynchronous Abort é um tipo de ataque que zera linhas de cache antes de uma transação que carrega dados dessas linhas, causando uma falha na transação. Alocando espaço no LFB antes da transação, os dados do LFB são encaminhados para a instrução que causa a *fault*. Como a transação não é completada, a instrução é executada com dados de uma transação anterior, permitindo a amostragem desses dados. [12]

4.4 ZombieLoad

Em 14 de maio de 2019 um grupo formado por pesquisadores de diversas universidades, dentre elas a austríaca Graz University of Technology e a belga Catholic University of Leuven, e por equipes das empresas de segurança Oracle, Cyberus, entre outras, juntamente com a Intel, reportou uma novo tipo de SCA, nomeado ZombieLoad [13], que explora vulnerabilidades presentes na maioria dos processadores Intel fabricados após 2011.[1]

O ZombieLoad, assim com o Spectre, Meltdown, Foreshadow, entre outros, pertence à classe dos *transient-execution attacks* [3], ataques que exploram vulnerabilidades resultantes de técnicas utilizadas para melhorar a performance do computador, como a execução fora de ordem e a execução especulativa. A primeira é um paradigma que permite que a CPU, ao invés de executar as microoperações de um conjunto de instruções sequencialmente, execute-as em paralelo, mesmo que a microoperação anterior não tenha sido finalizada, e as reorganize depois, decidindo aproveitar ou descartar os resultados. Já a

segunda consiste na CPU executar instruções especulativamente, ou seja, antes delas aparecerem na lista de instruções, utilizando análise do fluxo de dados e predicção de desvio, assim como na técnica anterior, os resultados das instruções executadas especulativamente podem ser aproveitados ou descartados.

Nesse caso, chamamos essa instrução executada fora de ordem ou especulativamente, cujo resultado foi descartado de *transient instruction*. Os efeitos da *transient execution* são descartados, contudo, utilizando canais laterais como o *CPU cache subsystem*, é possível extrair dados de outros processos carregados no mesmo core da CPU, como senhas, tokens, histórico de navegação do browser, entre outros.

A Intel liberou correções em microcódigo para os processadores vulneráveis e a 8ª e 9ª geração de processadores possuem correção em hardware. Contudo, essas mitigações reduzem a velocidade do computador em 3% e a potência em 9%. Os pesquisadores, no entanto, afirmam que essas medidas são insuficientes para evitar que um computador esteja vulnerável a esse ataque e que a solução mais segura seria desabilitar o *hyperthreading*. [14]

4.5 Foreshadow

5 Conclusão

Referências

[1] Em: URL: https://www.wired.com/story/intel-mds-attack-speculative-execution-buffer.

[2] Ranbir Singh Bali. *Cold Boot Attack on Cell Phones*. Jul. de 2018. DOI: 10.13140/RG.2.2.13560.14088. URL: https://www.researchgate.net/publication/326211565.

[3] Claudio Canella et al. *A Systematic Evaluation of Transient Execution Attacks and Defenses*. 2019. arXiv: 1811.05441 [cs.CR]. URL: https://arxiv.org/pdf/1811.05441.pdf.

[4] Daniel Genkin et al. "Physical Side-Channel Key-Extraction Attacks on PCs". Em: 2015. URL: https://www.tau.ac.il/~tromer/radioexp/.

[5] Daniel Genkin et al. "RSA Key Extration via Low-Bandwidth Acoustic Cryptanalysis". Em: 2014. URL: https://www.tau.ac.il/~tromer/acoustic/.

[6] Andy Greenberg. *Hacker Lexicon: What Is a Side Channel Attack?* 2020. URL: https://www.wired.com/story/what-is-side-channel-attack/. (acesso em: 07/12/2020).

[7] J. Alex Halderman et al. "Lest we remember: cold-boot attacks on encryption keys". Em: *Communications of the ACM* 52.5 (mai. de 2009), pp. 91–98. DOI: 10.1145/1506409.1506429. URL: https://www.usenix.org/legacy/event/sec08/tech/full_papers/halderman/halderman.pdf.

[8] Intel. *Microarchitectural Data Sampling*. https://software.intel.com/security-software-guidance/deep-dives/deep-dive-intel-analysis-microarchitectural-data-sampling. Online; Acessed: 24-November-2020.

[9] David Wagner John Kelsey Bruce Schneier e Chris Hall. "Side Channel Cryptanalysis of Product Ciphers". Em: (1998). URL: https://www.schneier.com/wp-content/uploads/2016/02/paper-side-channel.pdf.

[10] Moritz Lipp et al. "Meltdown: Reading Kernel Memory from User Space". Em: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.

[11] NSA. "TEMPEST: A Signal Problem". Em: 2007. URL: https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/cryptologic-spectrum/tempest.pdf.

[12] Stephan van Schaik et al. *CacheOut: Leaking Data on Intel CPUs via Cache Evictions*. 2020. arXiv: 2006.13353 [cs.CR].

[13] Michael Schwarz et al. "ZombieLoad: Cross-Privilege-Boundary Data Sampling". Em: *CCS*. 2019.

[14] Cyberus Technology. "ZombieLoad: Cross Privilege-Boundary Data Leakage". Em: 2019. URL: https://www.cyberus-technology.de/posts/2019-05-14-zombieload.html.

[15] 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.

[16] Daniel Genkin; Adi Shamir; Eran Tromer. *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*. 2013. URL: https://www.tau.ac.il/~tromer/acoustic/. (acesso em: 01/12/2020).

[17] "What is a Side Channel Attack". Em: URL: https://www.wired.com/story/what-is-side-channel-attack/.