Universidade Estadual de Maringá
Centro de Tecnologia - Departamento de Informática
Curso de Bacharelado em Ciência da Computação

Gabriel Thiago Henrique Dos Santos

Ataque de canal lateral e vulnerabilidades em sistema operacional para dispositivos IoT

TCC-2021

Maringá - Paraná
2021

Gabriel Thiago Henrique Dos Santos

Ataque de canal lateral e vulnerabilidades em sistema operacional para dispositivos IoT

Trabalho de Conclusão de Curso apresentado à Universidade Estadual de Maringá, como parte dos requisitos necessários à obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Luciana Andréia Fondazzi Martimiano

Coorientador: Anderson Faustino da Silva

Maringá - Paraná

2021

Gabriel Thiago Henrique Dos Santos

Ataque de canal lateral e vulnerabilidades em sistema operacional para dispositivos IoT

Trabalho de Conclusão de Curso apresentado à Universidade Estadual de Maringá, como parte dos requisitos necessários à obtenção do título de Bacharel em Ciência da Computação.

Luciana Andréia Fondazzi Martimiano
Orientadora

Anderson Faustino da Silva
Coorientador

Ronaldo Augusto de Lara Gonçalves
Convidado 1

Paulo Roberto de Oliveira
Convidado 2

Maringá - Paraná

2021

Resumo

Mitigações para vulnerabilidades existem com o intuito de reduzir ou aliviar ameaças que podem comprometer as bases da segurança da informação. No entanto, as mitigações podem não resolver o problema por completo, sendo inviável a aplicação em todos os ambientes. Esta situação ocorre para a vulnerabilidade *Spectre*, que está presente em processadores Intel, Apple, ARM, AMD e IBM, e permite ataques de canal lateral. Além disso, mitigações podem influenciar o desempenho do sistema, pois elas podem realizar mais instruções de checagens e manutenções. Paralelo a isso, a quantidade de dispositivos de Internet das Coisas (IoT) para automação e para uso rotineiro cresceu, e como muitos destes dispositivos podem utilizar processadores vulneráveis à *Spectre*, há necessidade de aplicar mitigações via software, uma vez que os proprietários tendem a prolongar a troca física destes dispositivos. Neste contexto, este trabalho tem por objetivo realizar uma análise do impacto no desempenho que as mitigações das variantes 1 (CVE-2017-5753) e 2 (CVE-2017-5715) da vulnerabilidade *Spectre* causam em sistemas operacionais para dispositivos IoT. Para a análise, fora utilizado o *Benchmark LMBench* por possuir um conjunto de *microbenchmarks*, possibilitando a análise de desempenho dos recursos das principais operações dos sistema. Os experimentos foram realizados em ambientes de virtualização *Quick Emulator* (QEMU) em conjunto com o Módulo *Kernel-based Virtual Machine* (KVM) para a máquina virtual do sistema operacional *Ubuntu Core* em duas máquinas, uma com processador Intel e outra com processador AMD, e três cenários para cada máquina: com apenas a mitigação v1 ativada; com apenas a mitigação v2 ativada; e com ambas mitigações desativadas, sendo a base de comparação. Com isso, foram realizados trinta testes utilizando todas operações do *BenchMark* LMBench para cada cenário, e com os resultados foram realizados comparações das operações em cada ambiente afim de observar os efeitos positivos e negativos entre elas quando cada uma das mitigações está desativada discutindo sobre das com efeitos maiores de vinte porcento.

Palavras-chave: Segurança; Sistemas Operacionais; Ubuntu Core; Spectre; IoT; Mitigação; LMBench; Virtualização;

Abstract

Vulnerability mitigations exist in order to reduce or alleviate threats that can compromise the foundations of information security. However, mitigations may not solve the problem completely, making it impossible to apply in all environments. This situation occurs for the Specter vulnerability, which is present on Intel, Apple, ARM, AMD, and IBM processors, and allows for side-channel attacks. In addition, mitigations can influence the performance of the system, as they can carry out more checks and maintenance instructions. Parallel to this, the number of Internet of Things (IoT) devices for automation and for routine use has grown, and as many of these devices may use processors vulnerable to Specter, there is a need to apply mitigations via software, since owners tend to prolong the physical exchange of these devices. In this context, this work aims to perform an analysis of the performance impact that the mitigations of variants 1 (CVE-2017-5753) and 2 (CVE-2017-5715) of the Specter vulnerability cause in operating systems for IoT devices. For the analysis, the LMBench Benchmark was used because it has a set of microbenchmarks, allowing access to cost resources of the main system operations. The experiments were carried out in Quick Emulator (QEMU) virtualization environments in conjunction with the Kernel-based Virtual Machine (KVM) Module for the Ubuntu Core operating system virtual machine on two machines, one with an Intel processor and the other with an AMD processor, and three scenarios for each machine: with only v1 mitigation enabled; with only v2 mitigation enabled; and with both mitigations deactivated, the latter being the basis of comparison. Thus, thirty tests were performed using all *BenchMarck* LMBench operations for each scenario and the results comparisons of the operations in each environment are carried out in order to observe the positive and negative effects between them when each one of the mitigations is disabled, discussing over those with effects greater than twenty percent.

Keywords: Security; Operational Systems; Ubuntu Core; Spectre; IoT; Mitigation; LMBench; Virtualization.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT	Internet das Coisas
UCP	Unidade Central de Processamento
SO	Sistema Operacional
DMA	Acesso Direto à Memória
RAM	Memória de Acesso Randômico
ROM	Memória Somente Leitura
IP	Internet Protocol
DRAM	Memória Dinâmica de Acesso Aleatório
SRAM	Memória Estática de Acesso Aleatório
MB	MegaByte
AMD	Advanced Micro Devices
CWE	Common Weakness Enumeration
HDMI	High-Definition Multimedia Interface
ACLC	Ataque de Canal Lateral em Cache
AET	Ataque de Execução Transitória
BTB	Branch Target Buffer
KVM	Kernel-based Virtual Machine
IBPB	Indirect Branch Prediction Barrier
STIBP	Single Thread Indirect Branch Predictor
RSB	Return Stack Buffer

IBRS	Indirect Branch Restricted Speculation
LTS	Long-Term Support
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
RPC	Remote Procedure Call
GRUB	GRand Unifield Bootloader

Sumário

1 Introdução

O desenvolvimento, tanto dos sistemas operacionais quanto dos componentes de *hardware* dos sistemas, está em crescimento constante para melhorar e garantir mais segurança. Tendo como foco os processadores modernos, estudos têm sido realizados nos últimos anos no sentido de melhorar o desempenho por completo para as execuções, havendo o desenvolvimento de diversas técnicas possíveis.

Duas das principais estratégias para otimização no desempenho do processador descritas por [1] são as de execução fora de ordem e as por especulação, que ocorrem na execução de programas, na qual a UCP (Unidade Central de Processamento), de acordo com as instruções já passadas, tende a supor o resultado de futuras condições dentro do código, e com a execução fora de ordem do *pipeline* os trechos são executados se não houverem dependências e relações entre instruções. Este processo tende a melhorar o aproveitamento e o tempo de execução quando há o acerto, porém, quando há erros, as instruções executadas são desfeitas e são realizadas as operações corretas. No entanto, tais estratégias estão sujeita a vulnerabilidades, pois os resultados destas execuções especulativas e fora da ordem são mantidos na memória, fato que resulta principalmente na permissão dos ataques de canal lateral, já que a verificação deste acesso para ser executada fora de ordem não ocorre.

Em 2017, houve a descoberta, e no ano seguinte a divulgação com as descrições, das vulnerabilidades de análise de canal lateral de processadores identificadas como: *Rogue Data Cache Load*, apelidada por *Meltdown* e registrada na CVE-2017-5754 [2], e a *Spectre*, com duas variantes, a *Bounds Check Bypass*, registrada na CVE-2017-5753 e a *Branch Target Injection*, registrada na CVE-2017-5715 [1]. Essas vulnerabilidades atingem os processadores desenvolvidos nos últimos vinte anos, causando um grande impacto, pois torna difícil a atualização física de todos equipamentos, tendo ainda novas variantes se desenvolvendo [1, 3]. Neste contexto, então, faz-se necessário realizar estudos no sentido de entender como essas vulnerabilidades acontecem e como é possível corrigi-las via software.

Em paralelo, considerando o avanço da Internet das Coisas (IoT, *Internet of Things*), é importante analisar se os sistemas operacionais e os componentes de *hardware* desenvolvidos para estes dispositivos estão seguros e não possuem tais vulnerabilidades

ativas sem as determinadas mitigações. Tem-se, ainda, que dificuldades adicionais podem ocorrer em dispositivos IoT com processadores vulneráveis, pois nos serviços oferecidos há troca ou armazenamento de informações sensíveis, como ocorre nas aplicações importantes em hospitais, câmeras de segurança, carros, industrias ou até mesmo os de uso pessoal. Nestes casos, a solução completa seria a troca de processadores vulneráveis para outros mais seguros, porém sendo mais custoso e, muitas vezes, inviável.

Como a vulnerabilidade *Spectre* atinge diversos tipos de processadores, é necessário um estudo dos impactos das mitigações, principalmente para os dispositivos IoT nos quais o sistema operacional pode ser afetado por mudanças no desempenho.

1.1 Objetivos

O objetivo principal deste trabalho é avaliar os efeitos das mitigações para as variantes 1 e 2 da vulnerabilidade *Spectre* no desempenho no sistema operacional *Ubuntu Core* com uso dos conjuntos de métricas do *benchmark* LMBench.

Como objetivos específicos, têm-se:

- Avaliar o desempenho que as devidas mitigações selecionadas podem causar e seus devidos impactos no *Ubuntu Core* para equipamentos IoT.
- Identificar quais métricas têm efeito positivo ou negativo quando as mitigações estão ativadas.
- Contribuir para o desenvolvimento de Sistemas Operacionais para equipamentos IoT mais seguros.

1.2 Organização da monografia

Esta monografia está organizada da seguinte forma: a Seção 2 descreve os conceitos fundamentais necessários para o desenvolvimento do trabalho, juntamente com o ambiente do sistema operacional selecionado; a Seção 3 descreve os materiais e métodos que foram utilizados, bem como a metodologia; a Seção 4 descreve os experimentos que foram realizados; a Seção 5 apresenta a avaliação dos resultados, respondendo às questão de

pesquisa definidas juntamente com as discussões sobre os experimentos; na Seção 6 são apresentadas as conclusões e as sugestões de trabalhos futuros. Por fim, são apresentadas as referências.

2 Fundamentação Teórica

Esta seção apresenta uma descrição dos conceitos básicos necessários para o desenvolvimento do Trabalho de Conclusão de Curso. Os seguintes tópicos são abordados: sistemas operacionais para IoT, ataque de canal lateral e execuções transitórias e a vulnerabilidade *Spectre*.

2.1 Sistemas Operacionais para Internet das Coisas

Os Sistemas Operacionais (SO) são os responsáveis por permitir a interconexão entre o usuário e o hardware, sendo o responsável pelo gerenciamento dos recursos do sistema. Segundo Tanenbaum [4], existem dois modos de definir um SO:

- *Top-Down*: sendo a visão de cima para baixo, realizando uma abstração do hardware a partir do ponto de vista dos usuários, sendo o responsável pela interação amigável entre o *software* e o *hardware*.
- *Bottom-Up*: sendo a visão de baixo para cima, sendo o SO o responsável pelo gerenciamento de recursos, controlando os processos para qualquer funcionalidade do sistema.

De acordo com [5], IoT trata-se de um ecossistema com um grupo de dispositivos interconectados através de uma rede para realizar troca, armazenamento e coleta de dados. Neste contexto da Internet das Coisas, os sistemas operacionais possuem algumas especificações e limitações, tendo o foco, por exemplo, nas otimizações de *footprint* (baixo consumo de memória FLASH, RAM e ROM) e na execução de tempo real. Os sistemas operacionais procuram trazer melhor aproveitamento do sistema em tarefas especiais, com foco na conectividade para envio e recebimento de requisições, no endereçamento, na comunicação sem fio e na produtividade.

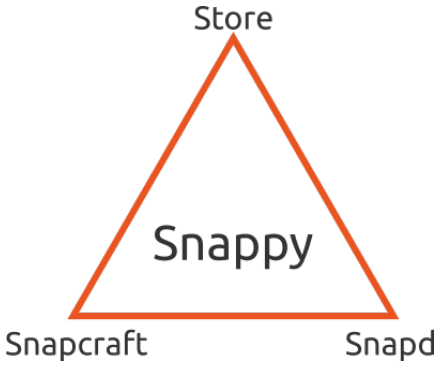
Com o avanço da IoT, algumas questões se tornaram primordiais no desenvolvimento, como é o caso da segurança dos dispositivos, pois diversos ataques utilizando equipamentos IoT surgiram e também se tornando o foco de estudos para os atacantes [6]. A segurança, então, para além do hardware, está evoluindo, sendo o sistema operacional um

dos principais responsáveis pela segurança e proteção [7] para os usuários e equipamentos IoT, além da responsabilidade pelo gerenciamento e pela comunicação entre os processos e equipamentos.

2.1.1 *Ubuntu Core*

O *Ubuntu Core* é um sistema operacional de código aberto desenvolvido pela *Canonical* para dispositivos IoT e embarcados, possuindo a sua base de código do Ubuntu LTS. O sistema possui uma base de Imagem de em torno 260MB, favorecendo o carregamento de pacotes que o usuário/desenvolvedor deseja utilizar, facilitando a integrações e realização de tarefas dos dispositivos desejados [8]. Este SO se diferencia das distribuições do Debian e Ubuntu padrões, sendo inteiramente baseado em pacotes *Snaps*, conforme é representado como pilares na Figura 1.

Figura 1 – Pilares para a construção do Ubuntu Core



Fonte: https://askubuntu.com/

Os *Snaps* são um formato específico de pacote definido como uma Imagem *squashfs*, a qual contém um arquivo *meta/snap.yarn* para construção de um pacote, ou seja, eles podem ser definidos como pacotes de software em *contêiner*, que simplificam a construção e instalação. Com isso, também tem-se por padrão a *Snap Store*, a qual é o repositório de *Snaps* que disponibiliza os programas para instalação dos usuários. Já o *Snapcraft* é uma ferramenta de linha de comando para construção de pacotes *Snaps*, tendo-se assim o *Snapd*, sendo o responsável indiretamente por realizar o processo que ocorre da construção, com o *Snapcraft*, e o pacote. Portanto, define-se o *Snapd* como um *daemon*[1] que é necessário para construir os *Snaps*, sendo sempre necessário para o intermediário da instalação final, até

[1] executa como um processo em plano de fundo.

mesmo na linha de comando com os *Snaps* disponibilizados pela loja, sendo o responsável pela comunicação da construção e chamada [9].

2.2 Ataque de Canal Lateral e Execuções Transitórias

As técnicas de ataque de canal lateral, diferentemente das vulnerabilidades usuais, não têm como alvo o software, mas sim os efeitos dos processos em execução no *hardware*, filtrando ou obtendo informações por meio do efeito colateral que a execução causa.

Para entender o ataque de canal lateral, considere um desenvolvedor que implementa um algoritmo em sua máquina pessoal, e para os testes de execução apenas se preocupa com as entradas e as saídas do seu código, o que, em teoria, é o correto. Porém, quando o algoritmo é executado, têm-se caminhos que são percorridos entre a inicialização do programa, execução e finalização que não são depuradas com o código em si. Neste cenário, é definido o conceito do canal lateral, que tem efeitos causados pela implementação no *hardware* no qual o algoritmo está sendo executado e não no algoritmo em si. O ataque se baseia nestas informações trafegadas.

De modo geral, existem as seguintes classes de ataque de canal lateral:

- Ataque de Cache (ACLC): baseado no acesso do atacante à cache do sistema compartilhado da vítima, causado por efeito colateral da CPU na execução das tarefas ou em caso de execução por especulação deixando rastros dos acessos, para então o monitoramento de dados da vítima tanto por ambientes físicos, virtualizado ou em serviços de nuvem;

- Ataque Cronometrado: baseado em avaliar a quantidade de tempo para execução das tarefas, como por exemplo, na comparação da senha digitada por um atacante com a senha verdadeira do usuário é necessário realizar os cálculos inteiros e sua devida comparação, para senhas próximas, em que há poucas diferenças entre a suposta e a senha verdadeira da vítima, o tempo é estimado e analisado, sabendo se está se aproximando, já que o tempo deve aumentar quando a aproximação acontece. Para senhas que estão longe de serem quebradas, o sistema realiza em termos menores, pois se no primeiro caractere já há diferença entre a senha enviada e a original, é

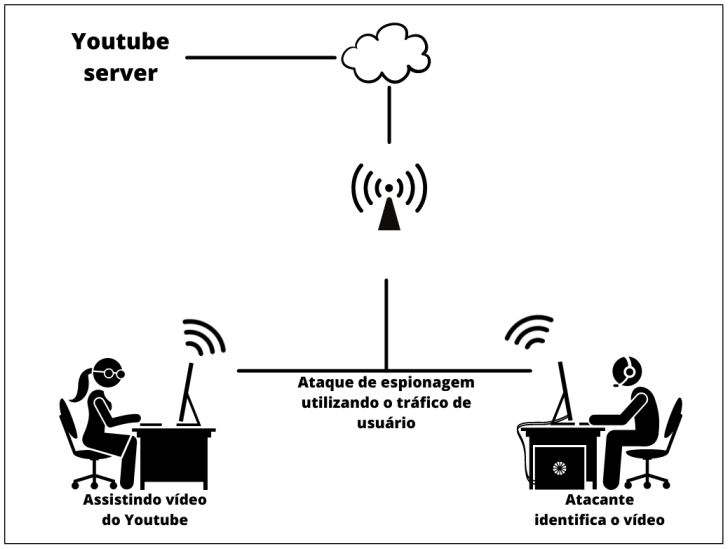
retornado o erro de senha inválida. Com as avaliações este ataque auxilia no ataque de Força Bruta;

- Ataque de Monitoramento de Energia: este ataque visa analisar o consumo de energia que varia no *hardware* devido às tarefas computacionais;
- Ataque Eletromagnético: tal ataque, ilustrado na Figura 2, lida com a radiação eletromagnética vazada das aplicações para obter informações diretas, como textos ou imagens. Tem-se o exemplo do *TEMPEST* [10], em que a conectividade entre um monitor pode permitir a captura da imagem através das ondas eletromagnéticas transmitidas pelo cabo HDMI (*High-Definition Multimedia Interface*);
- Criptoanálise Acústica: este ataque explora o acústico, ou seja, o som emitido/ produzido nos dispositivos em momentos das operações, tais como ocorre com os ataques de consumo de energia;
- Ataque de Remanência de Dados: este ataque foca na leitura de dados que já deveriam ter sido excluídos do sistema, estando em memória (DRAM - *Dynamic Random Acess Memory* e SRAM - *Static Random Acess Memory*) e acessadas por DMA (*Direct Memory Access*);
- Ataque de falha iniciado por software: sendo uma classe mais rara, explora a partir de falhas ocasionadas intencionalmente por softwares, como exemplo, o *Row Hammer* [11], no qual a memória fora dos limites é alterada, possibilitando acesso à memória vizinha.

Em todas as classes citadas, é possível ter a relação de que os ataques acontecem a partir dos efeitos do funcionamento de um cripto sistema, ou seja, na segurança da construção destes sistemas, sendo interpretado como "lateral", já que não são o foco principal do programa em execução.

Este trabalho tem como foco a classe de ataque de canal lateral em cache, que explora a suposição do preditor de ramificação da UCP para a execução por especulação das instruções. A predição pode resultar em: i) suposição errônea, assim, o caminho percorrido

Figura 2 – Exemplo da classe de ataque Eletromagnético



Fonte: Autor

é desfeito e as operações e resultados realizados são descartados; ii) suposição correta, otimizando o *pipeline* das instruções.

A execução transitória acontece quando as instruções são executadas por especulação, com o uso da suposição de ramificação. Nesta execução há a alteração dos dados na microarquitetura antes da confirmação de acerto. Quando a especulação ocorre incorretamente, os resultados das operações armazenados são desfeitas, porém não adequadamente, e os efeitos colaterais da microarquitetura ainda são acessíveis, e os ataques que buscam explorar estes dados são declarados de Ataques de Execução Transitórias (AET).

Visto que toda instrução que gera impacto no bloqueio do *pipeline* gera a execução transitória, por si só o bloqueio não necessariamente ocasiona falha de segurança, porém, quando combinadas com ataques focados neste cenário específico há quebra de segurança. Assim, um ataque completo acontece quando o Ataque de Canal Lateral em Cache é utilizado para acessar os dados da execução transitória causada a partir da execução especulativa.

2.2.1 *Spectre*

A vulnerabilidade definida como *Spectre* consiste então na exploração pelo canal lateral presente na execução especulativa, tendo em todas as variantes a capacidade de ler o

conteúdo da memória em nível de *Kernel*. Esta exploração ocorre, pois, devido ao desvio da predição de ramificação, instruções que deixam rastros na memória cache são executadas de forma especulativa, permitindo que os invasores acessem tais informações.

A vulnerabilidade foi descoberta por equipes de pesquisadores em 2017 e divulgada publicamente em 2018 [1], tendo seus impactos comprovados em microprocessadores da Intel, AMD, ARM, Apple e IBM. No momento da divulgação[2], duas variações do ataque são descritas e, de modo geral, ambas se diferem apenas na forma como a especulação é conduzida e em como as informações são vazadas.

Para o ataque, inicialmente na descoberta era preciso que o atacante executasse os códigos localmente em seu alvo, sendo então também acessível em navegadores e sendo demonstrados com o uso de *Javascript* [1, 12], como também com o código nativo [1, 13, 14], porém a situação se ampliou sendo comprovado a realização do ataque também por rede, exemplo do *NetSpectre* [15] realizando a leitura remota de conteúdo de memória sem a execução do código localmente.

Como detalhado em [1], para a execução das técnicas de ataque há duas fases primordiais: Configuração e Exploração.

Na Configuração, deve-se treinar o preditor de ramificação para atender a condição desejada, no caso pode ocorrer uma predição incorreta para o dado que será o objetivo. Este treino pode ser caracterizado por diversos saltos positivos parecidos com o que se planeja utilizar ao momento do ataque, fazendo então o preditor predizer que no futuro o salto será realizado. Nesta fase também ocorre a limpeza das caches para análises ao final.

Após a configuração, ocorre a Exploração, na qual o preditor tenderá a considerar verdadeiro para ocorrer a execução especulativa das instruções. Nesta, então, haverá as instruções para acessos e comparações de informações confidenciais a partir da ação da vítima, ou até mesmo a execução especulativa de *exploits*. No final da etapa, o processador verificará que o desvio considerado na ramificação é falso e se desfaz das alterações das instruções, porém, o ataque de canal lateral de cache acontece.

O próximo passo é descobrir qual valor que ficou armazenado na cache. Para

[2] A divulgação das CWEs (*Common Weakness Enumeration*) referentes ao ataque foi feita em 01/04/2018.

isto, há diversas técnicas populares de Ataque de Canal Lateral em Cache (ACLC) como descrito em [16]. A mais utilizada é a *Evict+Reload* descrita por [17], na qual há o método de limpar a cache primeiramente e, após a execução, realizar a investigação dos dados nos quais houverem alterações, sendo, então, nestas posições as instruções exigidas.

Considerando a parte da exploração e da configuração do ataque, têm-se as seguintes variantes:

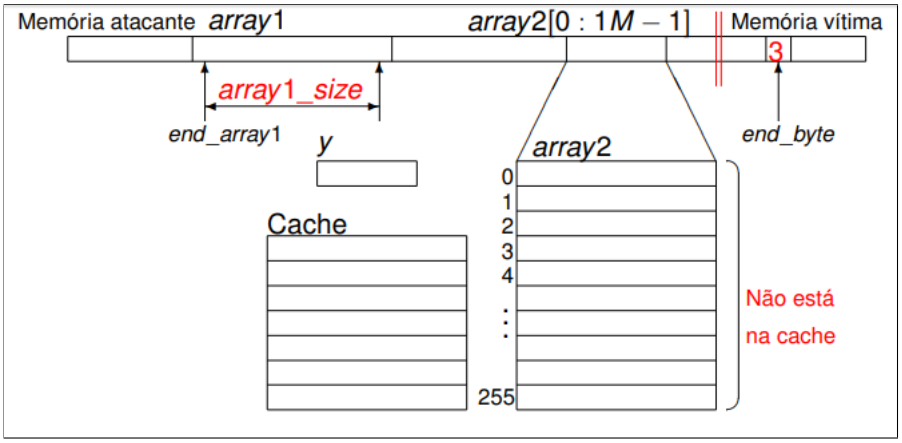
- Variante 1 (v1) - Explorando Ramificações Condicionais (*Exploiting Conditional Branches*): esta primeira variante explora as *Branches* condicionalmente, havendo pelo invasor a criação de tendência para o preditor de ramificação prever erroneamente a direção da ramificação, o que permite a execução especulativa indevida de instruções que não ocorreriam. Em seguida, são definidos os acessos pelo canal lateral. O exemplo a seguir do trecho de código (KOCHER et al., 2018) no qual a ramificação tende a especular positivamente a condição.

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

  Neste trecho, o valor de x é um inteiro legítimo, porém, de uma fonte não confiável, do invasor. A validade do acesso ao *array1* deste trecho é garantido verificando se o valor de x está dentro de um limite permitido (*array_size*), caso contrário uma exceção é disparada e o código é interrompido. Porém, o invasor pode burlar esta verificação quando for executada a manipulação do preditor de desvio, tendendo a considerar a condição verdadeira, e com isso chegando a segunda linha. Nesta condição ocorre o alvo para o ataque de canal lateral, tendo o *array1* como índice para o *array2*, acessando dados do 1 e armazenando no 2, podendo ser verificado com o ataque citado *Evict+Reload*[17]. A Figura 3 mostra a configuração dos *arrays* para o ataque.

  Este ataque em aplicações Web com *JavaScript* pode ocasionar no acesso indevido de leitura da memória do processo e *cookies* e dados sensíveis de outras abas do usuário[12].

Figura 3 – Configuração de ataque da *Spectre v1*



Fonte: Prof. Siang Wun Song

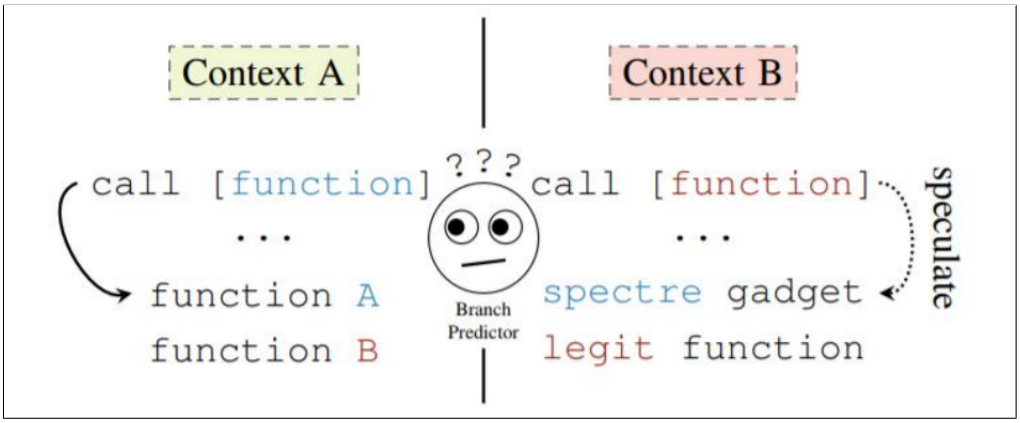
- Variante 2 (v2) - Explorando Ramos Indiretos (*Exploiting Indirect Branches*). Nesta variante há a utilização do *Branch Target Buffer*, que é o responsável pelo armazenamento dos endereços de origem e destino dos desvios indiretos mais recentes. Para o ataque, o invasor, na fase de configuração, treina o BTB para que quando a vítima execute um desvio indireto, ele tenha o controle do destino, sem corrupção de memória, para então encaminhar a execução para o endereço de um *gadget*[3], resultando na execução especulativa do *gadget*. Para se obter as informações resultantes do ataque, o estado da cache é acessado. A Figura 4 exemplifica a fase do treinamento para o BTB, o qual causa a confusão nas técnicas de Previsão de Desvio.

- Variante 4 (v4) - Desvio do Armazenamento Especulativo (*Speculative Store Bypass*. em processadores há solicitações de *buffer* para armazenar e carregar os devidos dos endereços de memórias com o uso especulativo para garantir o melhor desempenho nos acertos. Antes da escrita da memória, o processador verifica se algum dos endereços usados no carregamento já foi usado recentemente para os mesmos endereços, para evitar erros. Caso fora utilizado, os dados são descartados e carregados novamente para garantir a não alteração. O problema é que esta especulação ocorre em uma área compartilhada não protegida, permitindo o acesso de usuários sem privilégios, sendo o caminho para os atacantes acessarem se passando por usuário legítimo e então realizarem uma analise de canal de canal lateral [18]. A sua mitigação ocorre para

[3] Código que aciona o controle do invasor.

processadores Intel e AMD desativando por completo o desvio do armazenamento especulativo, o qual pode causar um maior impacto no desempenho. Já para os processadores ARM, apenas alguns núcleos são afetados, como o Cortex-A57, Cortex-A72, Cortex-A73 e Cortex-A75 e os fabricantes podem optar pela desabilitação da desambiguação de memória na inicialização, o de já mitiga a falha.

Figura 4 – Treinamento da previsão de ramificação na *Spectre* v2



Fonte: Kocher et. al., 2018, p. 6

Outras variações são possíveis, em que as fases para desenvolvimento do funcionamento permanece, alterando a forma de especulação e o foco no vazamento de dados do ataque. Como também é o caso da variante v3 categorizada como *Meltdown*, tendo além que diferentes variações estão sendo encontradas e divulgadas [1, 3, 19], porém, estas não são foco deste trabalho.

A solução completa e mais recomendada para mitigar o ataque seria apenas em modo hardware, já que se trata da arquitetura do processador, porém, a troca completa dos processadores é inviável, sendo esperado que nas gerações futuras de microprocessadores as arquiteturas desenvolvidas estejam já livres de tais vulnerabilidades. Como esta troca pode levar anos, foram desenvolvidos mitigações via *Kernel* do sistema operacional para se mitigar os ataques. No entanto, a mitigação via sistemas operacionais podem causar perdas no desempenho de diferentes modos nas execuções de programas e tarefas, tendo também que na vulnerabilidade *Spectre* se deve desenvolver mitigação especifica para cada variante, pois a técnica de ataque varia juntamente, possibilitando o aumento das verificações de saltos, autentificações ou limitando as predições nas execuções dos serviços [1].

3 Materiais e Métodos

A metodologia adotada para o desenvolvimento deste trabalho foi a experimental, conduzida em um ambiente controlado com uso de virtualização completa com o emulador de processador *QEMU* (*Quick Emulator*) e em conjunto com o módulo *KVM* (*Kernel-based Virtual Machine*) para a máquina virtual do sistema operacional *Ubuntu Core*. Para o ambiente controlado, dois *notebooks* foram utilizados com as especificações conforme mostra a Tabela 1.

Tabela 1 – Configurações dos notebooks utilizados para os experimentos

Especificação	**Notebook 1**	**Notebook 2**
SO	GNU/Linux Debian 11	GNU/Linux Ubuntu 20.04
Processador	AMD Ryzen 5-3500U	Intel Celeron
Memória RAM	8 GB's DDR4	8 GB's DDR3
Memória SSD	295 GB's	240 GB's
Placa de Vídeo	AMD Radeon Rx Vega 8	-

Em cada *notebook* foi realizada a virtualização do sistema operacional *Ubuntu Core* com o foco em dispositivos IoT vulneráveis e não vulneráveis para as análises quanto ao impacto no desempenho com o uso de um *benchmark* especializado.

3.1 Benchmarks

Benchmarks são conjuntos de programas pré-programados para realização de análises a partir de métricas que permite verificar os impactos em diferentes cenários. Para este trabalho, foi selecionado o *LMBench* [20], pois ele possui um conjunto objetivo de métricas que permitiu avaliar o desempenho a partir das alterações de ativações das mitigações em cada cenário de teste.

O LMbench, com Licença Pública Geral GNU (*GNU General Public License*), possui uma série de micro *benchmarks* para UNIX/POSIX focados na avaliação do sistema operacional básico e com métricas do sistema de *hardware*. Suas métricas básicas estão contidas em três grupos, sendo: i) Medidas de Latência; ii) Medidas de largura de Banda; iii) Outras Medidas; [20]. A Tabela 2 apresenta as classes e grupos para as métricas utilizadas neste trabalho.

Tabela 2 – Classes dos recursos do *benchMark LMbench*

Classe	**Grupo(s)**	**Recursos**
Largura de Banda	Larguras de Banda de Comunicação local	Leitura de arquivo da Cache Cópia de Memória Leitura de Memória Escrita em Memória Pipe TCP e RPC
Latência	Troca de Contexto Latências do sistema de Arquivos e VM Processador Latências de Comunicação Local Latência de Memória	Troca de contexto *Networking* Arquivos de sistemas Criação de processos Tratamento de Sinais Chamada de sistema *overhead* Leitura de Memória
Outras	Operações Básica de Inteiros Operações Básica de Float Operações Básica de Double	*Networking* mhz tlb Line Cache Stream par_mem[4] par_ops[5]

3.2 Ambiente dos experimentos

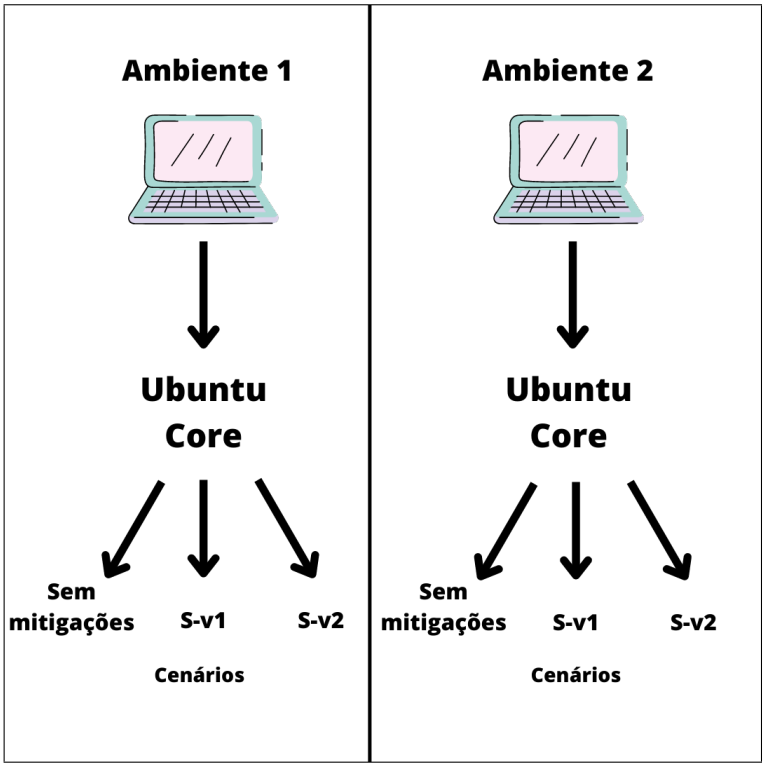
Foram definidos dois ambientes de experimentação conforme mostra a Tabela 1. Para cada ambiente, foram configurados três cenários. Os cenários foram configurados para avaliar o impacto da ativação das mitigações das duas variantes da vulnerabilidade, a *Spectre v1* e a *Spectre v2*, no *Kernel* do *Ubuntu Core*. Os cenários são:

1. Cenário 1: Mitigação *Spectre v1* e *Spectre v2* desativadas (Sem Mitigações);
2. Cenário 2: Mitigação *Spectre v1* ativada (S-v1);
3. Cenário 3: Mitigação *Spectre v2* ativada (S-v2).

Ambos os cenários e variações para os experimentos são ilustrado na Figura 5.

[4] paralelismo do subsistema de memória
[5] paralelismo básico de operação do processador.

Figura 5 – Ilustração dos ambientes e cenários de experimentação



Fonte: Autor

As seguintes mitigações padrões são inseridas no *Kernel* do sistema operacional para mitigar a vulnerabilidade *Spectre*:

Para a V1:

- *usercopy/swapgs barriers _user pointer sanitization*: proteção caso a caso com limpeza explícita de ponteiros, barreiras *LFENCE usercopy* e barreiras *LFENCE swapgs*.

Para a V2:

- *IBPB conditional*: adição de isolamento entre processos de diferentes usuários;
- STIBP: adição de isolamento entre *threads* de UCP em execução no mesmo núcleo;
- RSB: proteção de retorno de *buffer* da pilha na troca de contexto;
- *Retpoline*: esta mitigação trabalha como um "trampolim"de retorno, fazendo o uso de um *loop* infinito para se evitar que uma UPC especule o alvo de um salto indireto.

Para cada cenário, foram realizadas 30 execuções independentes para avaliar o desempenho de acordo com as métricas definidas pelo *BenchMark LMbench*.

3.2.1 Questões de pesquisa

Os resultados dos experimentos foram analisados a fim de responder as seguintes questões principais:

Q01. O sistema operacional possui estratégias e suporte para as mitigações das variantes da vulnerabilidade *Spectre*? Se sim, quais?

Q02. As mitigações são ativadas por padrão na instalação do Sistema Operacional? As ativações/desativações são simples para serem feitas?

Q03. Quais os impactos no desempenho do sistema com as mitigações habilitadas e desabilitadas?

Q04. Quais métricas têm impacto positivo no desempenho do sistema quando comparadas ao ambiente sem mitigações?

Q05. Em relação às classes das métricas, quais impactos positivos ou negativos no desempenho do sistema são possíveis avaliar?

A elaboração das perguntas se deu com o objetivo de contribuir de modo abrangente com a investigação de como estão sendo tratadas as questões de segurança e mitigações nos sistemas operacionais para dispositivos IoT.

4 Experimentos

Esta seção apresenta os experimentos realizados com base no metodologia detalhada na Seção 3.

Para obter validade nos resultados da análise de desempenho do estudo, foram realizadas trinta execuções independentes por métrica do *benchmark LmBench* para cada ambiente (notebook 1 e notebook 2) e os três cenários de configuração (Sem Mitigações; S-v1; S-v2).

4.1 Métricas

Cada classe de métricas apresentada na Tabela 2 é, ainda, dividida em grupos menores contendo as suas operações possíveis, conforme mostram as Tabelas de 3 à 6. Assim, as trinta execuções foram realizadas para cada métrica de cada um dos grupos.

Tabela 3 – Operações de Cálculo

Grupo	Métricas		Grupo	Métricas		Grupo	Métricas
Operações Básicas de Inteiro (BI)	intgr mul (1) intgr bit (2) intgr add (3) intgr mod (4) intgr div (5)		Operações Básicas de Float (BF)	float bogo (1) float mull (2) float add (3) float div (4)		Operações Básicas de Double (BD)	double div (1) double mul (2) double bogo (3) double add (4)

Tabela 4 – Operações com Memória e Arquivo

Grupo	Métricas		Grupo	Métricas
Latências de Memória (M)	Rand mem (1) L2 $ (2) Main mem (3) L1 $ (4) Guesses (5)		Latências do Sistema de Arquivos e VM (F)	10K Create (1) File Delete (2) File Delete (3) Page Fault (4) OK Create (5) Prot Fault (6) Nmap Latency (7) 100fd selct (8)

Tabela 5 – Operações do Processador e Troca de Contexto

Grupo	Operação		Grupo	Operação
Processador (P)	null call (1) fork proc (2) sh proc (3) open clos (4) exec proc (5) stat (6) sig hndl (7) sig inst (8) null I/O (9) slct TCP (10)		Troca de Contexto (CS)	16p/64K ctxsw (1) 2p/64K ctxsw (2) 8p/16K ctxsw (3) 8p/64K ctxsw (4) 16p/16K ctxsw (5) 2p/16K ctxsw (6) 2p/0K ctxsw (7)

Tabela 6 – Latência de Comunicações

Grupo	Operação		Grupo	Operação
Latências de Comunicação Local (CL)	UDP (1) Pipe (2) TCP (3) TCP conn (4) AF UNIX (5) 2p/0K ctxsw (6) RPC/UCP (7) RPC/TCP (8)		Larguras de Banda de Comunicação Local (CB)	Pipe (1) Bcopy (libc) (2) File reread (3) Mem write (4) Bcopy (hand) (5) AF UNIX (6) Mem read (7) TCP (8) Nmap reread (9)

4.2 Resultados obtidos

Com todas as métricas coletadas, foi possível separar os resultados em dois grupos principais: as que tiveram efeito e as que não tiveram efeito para cada ambiente de teste.

Ressalta-se que neste trabalho são apresentados apenas os resultados das médias e desvios padrão para os grupos das operações. As configurações dos ambientes e o conjunto completo dos dados podem ser acessados através do *GitHub*[6].

As Tabelas de 7 a 14 mostram os resultados, sendo para v1 e v2 desativadas chamada de "Sem Mitigações", para Somente v1 ativada chamada de “S-v1“ e Somente v2 ativada chamada de “S-v2“. O cenário “Sem Mitigações“ foi utilizado como base de comparação, sendo que efeito positivo significa que houve a melhora de desempenho quando Sem Mitigações, e que efeito negativo significa que houve a piora.

[6] https://github.com/gth1ago/artefato-tcc

Tabela 7 – Sem mitigações x S-v1 - Efeito Negativo - Notebook 1

stat	float div	AF UNIX	File reread	TCP
sig hndl	intgr mod	2p/0K ctxsw	slct TCP	Bcopy (libc)
sig inst	intgr div	Prot Fault	2p/0K ctxsw	Pipe
null I/O	double add			

Tabela 8 – Sem mitigações x S-v2 - Efeito Negativo - Notebook 1

stat	double bogo	TCP	double mul	8p/16K ctxsw
float bogo	2p/16K ctxsw	Main mem	File Delete	File reread
float add	AF UNIX	L2 $	Bcopy (hand)	AF UNIX
float div	Prot Fault	TCP conn	File Delete	OK Create
double add	double div			

Tabela 9 – Sem mitigações x S-v1 - Efeito Positivo - Notebook 1

fork proc	float mull	Pipe	OK Create	16p/64K ctxsw
sh proc	float add	TCP	Rand mem	2p/64K ctxsw
open clos	double div	TCP conn	L2 $	8p/16K ctxsw
exec proc	double mul	Mem write	Main mem	8p/64K ctxsw
intgr mul	double bogo	Bcopy (hand)	L1 $	16p/16K ctxsw
intgr bit	2p/16K ctxsw	File Delete	AF UNIX	File Delete
float bogo	UDP	Page Fault	Mem read	null call
10K Create				

A partir dos resultados, foi realizada uma classificação considerando a porcentagem de efeito de acordo com seis classes: i) 0% - 10%; ii) 11% - 20%; iii) 21% - 40%; iv) 41% - 60%; v) 61% - 80%; e vi) 81% +.

As Tabelas 15 e 16 apresentam as porcentagens para o Ambiente 1 e as Tabelas 17 e 18 para o Ambiente 2.

Para a avaliação, foram selecionadas as métricas do grupo de operações que causaram um efeito positivo maior que 20% com as mitigações desativadas em cada um dos ambientes e cenários. Esta escolha se deve ao objeto de filtragem das métricas com baixo nível de diferença, mantendo o foco do estudo naquelas de melhoras maiores. Sendo então apenas estas devidas operações descritas a seguir:

- null call: a chamada nula realiza a chamada/invocação do sistema, pegando o número do processo, por exemplo;
- sig hndl: captura e processa o sinal;

Tabela 10 – Sem mitigações x S-v2 - Efeito Positivo - Notebook 1

slct TCP	float mull	UDP	L1 $	TCP
null I/O	16p/64K ctxsw	Page Fault	Rand mem	L1 $
sig inst	Pipe	Mem write	8p/64K ctxsw	Rand mem
fork proc	2p/0K ctxsw	TCP	null call	sig hndl
sh proc	16p/16K ctxsw	Pipe	open clos	intgr div
intgr bit	2p/0K ctxsw	intgr mod	intgr mul	2p/64K ctxsw
exec proc	8p/64K ctxsw	10K Create		

Tabela 11 – Sem mitigações x S-v1 - Efeito Negativo - Notebook 2

sig hndl	null I/O	float mull	File Delete	AF UNIX
open clos	null call	float div	Page Fault	Bcopy (libc)
slct TCP	intgr mod	double div	OK Create	Pipe
sig inst	intgr div	16p/16K ctxsw	Bcopy (hand)	16p/64K ctxsw
fork proc	intgr mul	2p/16K ctxsw	10K Create	Rand mem
8p/16K ctxsw	Pipe	UDP		

Tabela 12 – Sem mitigações x S-v2 - Efeito Negativo - Notebook 2

stat	2p/64K ctxsw	OK Create	TCP	16p/64K ctxsw
sh proc	8p/64K ctxsw	L2 $	Main mem	double div
intgr mul	8p/16K ctxsw	Rand mem	L1 $	Prot Fault
float bogo	16p/16K ctxsw	Bcopy (libc)	Page Fault	double add
float div	Pipe			

- intgr mod: cálculo de resto em operação de inteiros;
- open clos: representa o resultado de cada chamada do sistema em um arquivo temporário para abertura e fechamento;
- intgr div: cálculo de divisão em operação com inteiros;
- intgr mul: cálculo de multiplicação em operação com inteiros;
- exec proc: execute a chamada *execve* (executa o programa referido pelo nome do caminho), realiza um *fork* e então há a saída;
- 10K *Create*: realiza a criação de 10 mil documentos;
- Xp/YK ctxsw: realiza troca de contexto, sendo os tempos de alternância de contexto entre diferentes números de processos (X) com diferentes tamanhos de conjunto de trabalho (Y). Exemplo: 8p/64K apresenta 8 processos que lidam com dados de 64K em paralelo;

Tabela 13 – Sem mitigações x S-v1 - Efeito Positivo - Notebook 2

exec proc	double mul	TCP	File reread	8p/64K ctxsw
sh proc	double add	2p/0K ctxsw	Mem write	float bogo
stat	double bogo	File Delete	L2 $	Nmap reread
float add	2p/0K ctxsw	Prot Fault	Main mem	TCP
AF UNIX	TCP conn	Mem read	L1 $	2p/64K ctxsw

Tabela 14 – Sem mitigações x S-v2 - Efeito Positivo - Notebook 2

null I/O	intgr div	TCP conn	Bcopy (hand)	Mem read
sig inst	intgr bit	TCP	sig hndl	2p/0K ctxsw
null call	intgr mod	AF UNIX	fork proc	File Delete
open clos	float add	UDP	double mul	2p/0K ctxsw
exec proc	float mull	10K Create	File Delete	AF UNIX
Pipe	Mem write	File reread	Nmap reread	2p/16K ctxsw
slct TCP				

Tabela 15 – Sem mitigações x S-v1 - notebook 1

Efeito				
0% - 10%	fork proc sh proc open clos exec proc intgr mul intgr bit float bogo	float mull float add double div double mul double bogo 2p/16K ctxsw UDP	Pipe TCP TCP conn Mem write Bcopy (hand) File Delete Page Fault	OK Create Rand mem L2 $ Main mem L1 $ AF UNIX Mem read
11% - 20%	16p/64K ctxsw 2p/64K ctxsw	8p/16K ctxsw 8p/64K ctxsw	16p/16K ctxsw File Delete	null call
21% - 40%	10K Create			
41% - 60%				
61% - 80%				
81% +				

- slct TCP: teste de rede;
- Stat: obter o estado do documento.

As Tabelas 19, 20 e 21 apresentam os resultados das métricas de operações com maiores porcentagens.

Tabela 16 – Sem mitigações x S-v2 - notebook 1

Efeito				
0% - 10%	slct TCP null I/O sig inst	float mull 16p/64K ctxsw Pipe	UDP Page Fault Mem write	L1 $ Rand mem
11% - 20%	fork proc sh proc	2p/0K ctxsw 16p/16K ctxsw	TCP Pipe	intgr bit 2p/0K ctxsw
21% - 40%	null call open clos	sig hndl intgr div	intgr mod	10K Create
41% - 60%	intgr mul	2p/64K ctxsw		
61% - 80%	8p/64K ctxsw			
81% +	exec proc			

Tabela 17 – Sem mitigações x S-v1 - notebook 2

Efeito				
0% - 10%	exec proc sh proc stat float add float bogo Nmap reread	double mul double add double bogo 2p/0K ctxsw TCP conn AF UNIX	TCP 2p/0K ctxsw File Delete Prot Fault Mem read TCP	File reread Mem write L2 $ Main mem L1 $
11% - 20%	2p/64K ctxsw	8p/64K ctxsw		
21% - 40%				
41% - 60%				
61% - 80%				
81% +				

Tabela 21 – Médias Maiores Porcentagem - notebook 2

Cenário	Tipo	slct TCP	2p/16K ctxsw
Sem Mitigações	Média	6,5567	8,9530
	Desvio	0,2028	2,8692
S-v1	Média	6,4747	8,8777
	Desvio	0,1267	2,9292
S-v2	Média	13,3690	10,9257
	Desvio	0,1257	6,7534

Tabela 18 – Sem mitigações x S-v2 - notebook 2

Efeito				
0% - 10%	null I/O sig inst null call open clos exec proc Pipe Mem read	intgr div intgr bit intgr mod float add float mull Mem write	TCP conn TCP AF UNIX UDP 10K Create File reread	Bcopy (hand) sig hndl fork proc double mul File Delete Nmap reread
11% - 20%	2p/0K ctxsw	File Delete	2p/0K ctxsw	AF UNIX
21% - 40%	2p/16K ctxsw			
41% - 60%				
61% - 80%				
81% +	slct TCP			

Tabela 19 – Médias Maiores Porcentagem 1 - notebook 1

Cenário	Tipo	null call	open clos	sig hndl	exec proc	10K Create
Sem Mitigações	Média	0,0545	4,0059	0,7052	1079,5517	31,3724
	Desvio	0,0083	1,3507	0,2087	75,3784	10,8640
S-v1	Média	0,0617	4,1740	0,7047	1119,9000	43,1100
	Desvio	0,0178	1,7040	0,2356	138,0266	46,0452
S-v2	Média	0,0720	4,9073	0,8570	1127,9259	40,0000
	Desvio	0,0288	2,0986	0,3684	137,3759	34,6467

Tabela 20 – Médias Maiores Porcentagem 2 - notebook 1

Cenário	Tipo	intgr mul	intgr div	intgr mod	2p/64K ctxsw	8p/64K ctxsw
Sem Mitigações	Média	0,8997	6,0521	4,4248	13,7659	32,9807
	Desvio	0,1074	1,6406	1,2581	5,1150	19,1883
S-v1	Média	0,9340	5,8797	4,3123	16,3687	38,2573
	Desvio	0,1656	0,9497	0,9442	8,2039	18,0763
S-v2	Média	1,3100	8,5300	5,9200	20,9000	57,8000
	Desvio	0,9500	5,8490	4,1817	12,8627	39,1647

5 Análise dos Resultados

Para a análise dos resultados obtidos, são respondidas as questões definidas na Seção 3.2.1. Em seguida, são apresentados os detalhes dos valores das médias das métricas de operações em cada ambiente.

5.1 Respondendo às Questões

Q01. O sistema operacional *Ubuntu Core* possui suporte oficial para as arquiteturas comprometidas pela vulnerabilidade *Spectre*, sendo referente a *AMD64* e *i386*. Já as placas *Raspberry* (Pi 2; Pi 3; *Dragonboard* 410c), que utilizam processadores *Cortex*(A7; A53), possuem dispositivos que não suportam execução especulativa e, portanto, não são afetados pelos ataques e não possuem atualizações para mitigações. Assim, as mitigações estão disponíveis para o *Kernel* do Linux.

Para a variante 1 da vulnerabilidade *Spectre*, tem-se:

- *usercopy/swapgs barriers and ___user pointer sanitization.*

Para a variante 2 da vulnerabilidade *Spectre*, têm-se:

- *Full AMD/generic retpoline*;
- *STIBP*;
- *RSB filling.*

Q02. Quando a instalação do sistema operacional é feita seguindo o padrão da imagem do *Ubuntu Core*, as mitigações para as variantes v1 e v2 da vulnerabilidade *Spectre* estão ativadas por padrão.

Para que ocorra a alteração de determinada mitigação, não é tão simples após o início da execução do sistema operacional. Uma vez que utiliza-se por padrão o *bootloader* GNU GRUB, a customização é realizada por argumentos na linha de comando do *kernel* [21], definidos pelo *snapd* e realizados na configuração da inicialização quando ocorre a montagem da imagem do sistema [22]. Portanto, para realizar a alteração de um argumento, é preciso reconstruir a imagem do sistema, impactando no tempo e trabalho a ser desenvolvido.

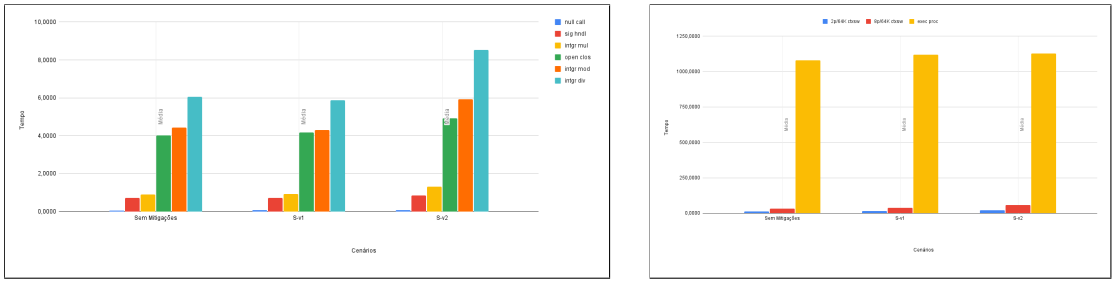
Q03. É possível observar a quantidade positiva de operações com valores médios positivos quando as mitigações estão desativadas. Para o ambiente 1: (i) Sem mitigações e S-v1, com 36 operações com melhora e 17 com efeito negativo; (ii) Sem mitigações e S-v2, com 33 operações com melhora e 22 com efeito negativo. Já para o ambiente 2: (i) Sem mitigações e S-v1, o com 25 operações com melhora e 28 com efeito negativo; (ii) Sem mitigações e S-v2, com 31 operações com melhora e 22 com efeito negativo.

No entanto, em relação à porcentagem de melhora, a maior parte se manteve em até 10%, isso se deve principalmente pelo seguinte motivo: i) o grande estudo nas atualizações para melhora de cada mitigação desde a descoberta das vulnerabilidades, no qual em 2018 as primeiras mitigações para Spectre-v2 causava uma perda de desempenho de até 35% aos processadores Intel e 14.5% aos da AMD. Já nas mais recentes, tanto a AMD quanto a INTEL, preocupadas com o impacto no desempenho das mitigações, documentaram a IBRS mais eficientes para suas microarquiteturas [23, 24], causando assim um impacto mais controlado desde as descobertas da vulnerabilidade.

Q04. Nas Tabelas 19 e 20 são apresentadas as médias das operações com maiores porcentagens. A Figura 6 mostra alguns resultados com o notebook 1. Percebe-se que há melhora quando compara-se com a base (*Spectre-v1* e *Spectre-v2* desativadas).

Para o notebook 2, a Tabela 21 mostra os valores, que são comparados por meio da Figura 8. As operações de porcentagens com melhora no desempenho maiores de 20% são explicadas a seguir.

Figura 6 – Médias das operações com maiores porcentagens com o notebook 1



Null Call: o uso das mitigações para as variantes do *Spectre* impacta diretamente nas chamadas ao sistema, já que os objetivos em cada mitigação utilizada são

Figura 7 – Desvio padrão das operações com maiores porcentagens com o notebook 1

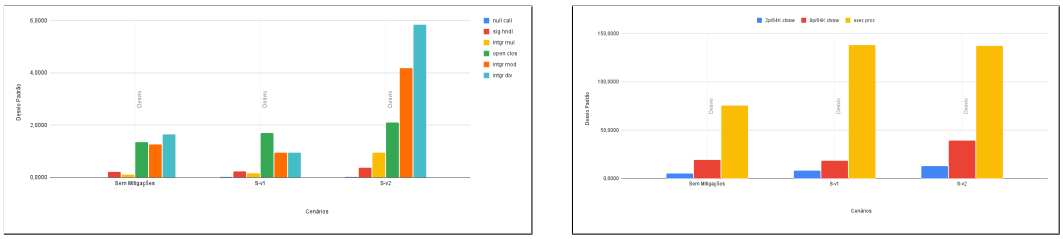
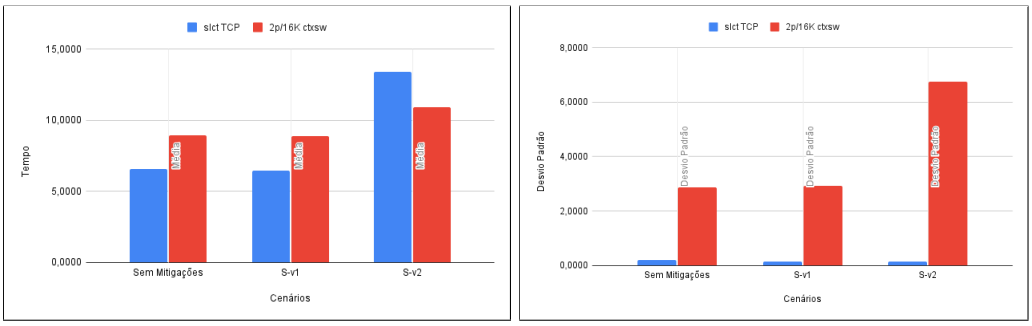


Figura 8 – Média e Desvio padrão das operações com maiores porcentagens com o notebook 2



de: (i) realizar mais checagens para confirmações da permissão do usuário; (ii) desabilitar técnicas de otimizações para execução por especulação. Com isso, quando as mitigações estão desabilitadas, tende-se a realizar otimizações padrões do sistema para qualquer execução necessária no sistema, e as chamadas ocorrem por padrão. Para o experimento, houverem variações em cada cenário, porém, com desempenho positivo em ambos os ambientes para as operações relacionadas às chamadas de sistema.

sig hdl e *exec proc*: dentro do grupo dos processos, a operação de mudança de sinal (*sig hdl*) e de execução de um processo com o uso do *fork* (*exec proc*) podem acarretar nas checagens de permissões de processos e em armazenamento de valores na memória, sendo acessível como efeito de mitigações como STIBP e IBPB, nas quais ocorre o isolamento entre *threads* e *processos* em momentos de execução para maior proteção, porém, com baixo nível de impacto às máquinas comuns já que o processamento sem o uso máximo pode não ser muito perceptível.

slct TCP: nesta operação de teste de rede, com o modo *select*, há a medição da latência de comunicação entre os processos, e então com as mitigações a segurança

abordando no envio e recebimento, garantindo o isolamento de cada um para que apenas o *token* enviado seja acessado pelo processo cliente, podendo então impactar no tempo total de ida e volta da comunicação.

Open-Close: nestas operações de abertura e fechamento de arquivos, o uso das mitigações pode impactar nas checagens adicionais de permissão do usuário, tendo como exemplo usuários com permissões de leitura em determinados arquivos, com isso para se evitar das tentativas de leitura em determinado arquivo diretamente ou indiretamente através da execução por especulação.

Operações de Cálculos: as operações são feitas com Inteiro (*Integers*), Fracionários (*Float*) e Dobrados (*Double*), realizando operações de soma, multiplicação e divisão. Com as mitigações ativadas, para a v1 não ocorreu impacto na performance, pois a limpeza de ponteiros, barreiras *usercopy* e *swapgs* não são utilizadas, porém para a v2, ocorreu impacto pelo isolamento entre threads *STIBP*, a qual causaria um melhor desempenho ao ser trabalhadas juntas.

Troca de Contexto (ctxsw): neste caso o impacto entre diferentes tempos de alternância e de números de processos é observado. O impacto pode ser maior com a mitigação para a variante 2, isso porque a troca de contexto é altamente utilizado no ataque a esta vulnerabilidade, a qual acessa a Tabela cache correspondente dos endereços armazenados para que ocorra o domínio consciente da cache. Com isso, a mitigação para o *Spectre-v2* com o RSB realiza a proteção de retorno de *buffer* da pilha na troca de contexto, ocorrendo um atraso para a realização completa das trocas. Quando a mitigação está desativada, a realização ocorre como planejado e o desempenho tendendo a melhorar. Em relação a mitigação da *Spectre-v1*, a proteção e limpeza explicita caso a caso tende a impacta em quesitos do experimento, já que quando ocorre a troca há a nova utilização de ponteiros.

Q05. Para as classes de Latência e Largura de Banda, foi calculada a média de seus tempos de execução respectivos de todas as métricas em cada uma das classes, tendo a média final, representando a classe, apresentada nas Tabelas de 22 a 25, para cada ambiente e cenário.

Tem-se que para os valores de Largura de Banda, sendo que quanto maior, melhor, já

que indica a capacidade de um determinado meio de transmissão em um determinado tempo. Já para a latência, ocorre o inverso, quanto menor são os valores, melhor são as avaliações no desempenho, pois representa o intervalo de tempo necessário para a execução por completo da ação.

Cenário	Média Final
Sem Mitigações	6561,675
S-v1	6455,664
S-v2	6827,069

Tabela 22 – Largura de Banda - Notebook 1

Cenário	Média Final
Sem Mitigações	137,884
S-v1	146,472
S-v2	146,560

Tabela 23 – Latência - Notebook 1

Cenário	Média Final
Sem Mitigações	4021,271
S-v1	3978,333
S-v2	3946,347

Tabela 24 – Largura de Banda - Notebook 2

Cenário	Média Final
Sem Mitigações	151,582
S-v1	152,640
S-v2	150,886

Tabela 25 – Latência - Notebook 2

Para a largura de banda, as mitigações podem não ter influenciado em grande escala, mantendo uma baixa variação, sendo algo esperado já que as mitigações ativadas não resultam em limitações diretas à largura de banda, sendo que as checagens são realizadas neste caso apenas no momento pré-envio e após o recebimento de informações.

Já na latência, tem-se a influência do uso da técnica de execução fora de ordem e/ou especulativa, já que o número de ciclos de instruções quando utilizadas tais técnicas tende a diminuir o tempo de latência por ser técnicas de otimização.

Como as mitigações ativadas pode realizar alteração das permissões das execuções das instruções e em alguns casos até desabilitar tais técnicas, a realização em tempos maiores de ciclos tende a ocorrer, com isso quando desabilitadas o seu tempo de latência pode ter um ganho. Podendo assim ser levado a decisão para qual uso das mitigação ao responsável do sistema, para caso o foco esteja em sempre manter o máximo da latência, a importância de se ter o uso de mitigações que não impactam nas técnicas oferecidas na otimização do processador.

5.2 Limitações

Vale ressaltar que nas execuções dos testes, a ferramenta *LMBench*, por motivos desconhecidos, não realizou algumas das operações esperadas, conforme descrito a seguir:

- Notebook 1: Em todos cenários de mitigações: intgr add; RPC/UCP; RPC/TCP; Nmap Latency; 100fd; selct; Nmap reread; Guesses;
  Apenas com v1 desativada: Bcopy (libc); Mem read.
- Notebook 2: Em todos cenários de mitigações: intgr add; RPC/UCP; RPC/TCP; Nmap Latency; 100fd; selct; Guesses;
  Apenas com v1 desativada: intgr bit;
  Apenas com v2 desativada: double bogo.

5.3 Considerações Finais

Esta seção apresentou a análise dos resultados obtidos e as respostas das questões de pesquisa definidas. Foram apresentadas e analisadas as métricas de operações com maiores taxas de variações positivas, porém, suas explicações podem ocorrer para as demais operações relacionadas, tendo para cada uma delas o impacto de mitigações da variantes.

6 Conclusão e Trabalhos Futuros

A descoberta da vulnerabilidade *Spectre* e suas variantes tem sido um problema desde 2018, não pelo seu modo crítico de ataque e sim pelo grande impacto aos processadores desenvolvidos nas últimas décadas. A exploração destas vulnerabilidades pode acarretar em vazamento de informações, como dados pessoais e senhas, sendo necessário conhecimento prévio dos conceitos e funcionamentos da arquitetura de computadores para sua mitigação.

Para limitar os ataques por meio da vulnerabilidade, foram desenvolvidas mitigações em nível de *Firmware* e/ou em nível de *Software*, mais especificamente no sistema operacional, uma vez que a troca do *hardware* em todos ambientes é algo custoso e moroso.

Como consequência, *patchs* de segurança podem causar perda de desempenho nos sistemas. Apesar da prioridade das empresas e dos desenvolvedores ser a segurança dos sistemas e dos usuários, é preciso manter e sempre melhorar o desempenho dos sistemas para que seja possível usufruir do máximo do processamento. Assim, é importante avaliar o quanto mitigações podem influenciar em perdas ou ganhos de desempenho em determinadas circunstâncias.

Com o crescimento do uso dos dispositivos de Internet das Coisas, é necessário avaliar se as mitigações da vulnerabilidade *Spectre* podem influenciar o desempenho de sistemas operacionais usados neste dispositivos.

Neste contexto, este trabalho realizou experimentos em dois ambientes com os processadores AMD e Intel, com o sistema operacional *Ubuntu Core* e as métricas do *benchmark LmBench*. A partir dos resultados, notou-se que as métricas de operações geram variações positivas de desempenho quando as mitigações da *Spectre-v1* e *Spectre-v2* estão desativadas, havendo ainda maior perda de desempenho quando a mitigação para a *Spectre-v2* estava ativada. No entanto, não é possível afirmar que estes resultados irão se repetir para todos os casos, pois a pesquisa experimental foi realizada em apenas dois ambientes.

Os resultados apontam que há o impacto no desempenho a partir das alterações nas mitigações no sistema operacional selecionado. No entanto, em quesitos específicos dos grupos das operações, se tem que a maior parte das operações do *benchmark* ocorre

com variação de até 10% apenas, tanto positivamente quanto negativamente, levando a possibilidade de escolha para as empresas responsáveis na disponibilidade dos dispositivos, porém, tendo a preocupação prioritariamente na segurança de se proteger contra as vulnerabilidades.

Como sugestões para trabalhos futuros, destacam-se:

- Utilizar outros sistemas operacionais para IoT para o estudo e avaliações no impacto de desempenho causado pelas mitigações;
- Utilizar diferentes *benchmarks* para os testes, como é o caso do *Phoronix*, o qual possui valores e gráficos de experimentos aplicados ao desempenho com execuções de programas de diversas áreas, buscando uma análise mais abrangente, porém, com um custo maior de tempo de execução e preparação do ambiente para integração ao sistema operacional selecionado;
- Realizar experimentos para a vulnerabilidade *Meltdown*, com processadores específicos da Intel para execução dos testes com e sem as mitigações;
- Realizar testes focados na avaliação em apenas um grupo de mitigação de uma variante, por exemplo, para a mitigação da v2 e com os cenários sendo variações desativadas entre IBPB *conditional*, STBP e/ou *Retpoline*, para se ter estudo mais específico de seus impactos;
- Estender o estudo de impacto de desempenho para as novas variantes (v4 e/ou v5, por exemplo) do *Spectre* e *Meltdown*, as quais possuem mitigações específicas.

Referências

[1] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019. 10, 18, 21

[2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018. 10

[3] J. H. Google Project Zero and K. J. Microsoft Security Response Center, "Spectre V4/Speculative Store Bypass (SSB)/CVE-2018-3639." https://bugs.chromium.org/p/project-zero/issues/detail?id=1528, 2018. Online; acessado em 6 de Setembro de 2021. 10, 21

[4] A. S. Tanenbaum, *Sistemas Operacionais Modernos*. São Paulo: Pearson Universidades, Prentice Hall, 4 ed., 2015. 13

[5] P. Carrion and M. Quaresma, "Internet da coisas (iot): Definições e aplicabilidade aos usuários finais," *Human Factors in Design*, vol. 8, pp. 049–066, mar. 2019. 13

[6] M. A. Razzaq, S. H. Gill, M. A. Qureshi, and S. Ullah, "Security issues in the internet of things (iot): A comprehensive study," *(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, 2017. 13

[7] B. L. Stuart, *Princípios de Sistemas Operacionais - Projetos e aplicações*, vol. 1. Editora Cencage Learning, 2015. 14

[8] Canonical, "Datasheet: Ubuntu Core 20." https://assets.ubuntu.com/v1/b2c770ea-Ubuntu+Core20+Datasheet.pdf, 2020. Online; acessado em 20 de Março de 2022. 14

[9] M. Huttleworth, "Announcing Ubuntu Core, with snappy transactional updates." https://www.markshuttleworth.com/archives/1434, 2014. Online; acessado em 6 de Setembro de 2021. 15

[10] E. L. T. Esoj, "Spectre e Meltdown - Entendendo vulnerabilidades de processadores." https://www.mentebinaria.com.br/artigos/spectre-e-meltdown-entendendo-vulnerabilidades-de-processadores-r76, 2018. Online; acessado em 6 de Setembro de 2021. 16

[11] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," in *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, NO. 8*, vol. 39, 2020. 16

[12] L. H. C. M. Marques and A. F. da Silva, "Meltweb: A transient execution attack to capture data in fill line buffer," in *IEEE LATIN AMERICA TRANSACTIONS, VOL. 14,*, 2021. 18, 19

[13] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre attacks: Stealing intel secrets from sgx enclaves via speculative execution." https://arxiv.org/abs/1802.09085, 2018. Online; acessado em 14 de Maio de 2022. 18

[14] C. Trippel, D. Lustig, and M. Martonosi, "Meltdownprime and spectreprime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols." https://arxiv.org/abs/1802.03802, 2018. Online; acessado em 14 de Maio de 2022. 18

[15] M. Schwarz, M. Schwarzl, M. Lipp, J. Masters, and D. Gruss, "Netspectre: Read arbitrary memory over network." https://misc0110.net/web/files/netspectre.pdf, 2019. Online; acessado em 14 de Maio de 2022. 18

[16] L. H. C. M. Marques and A. F. da Silva, "Ataques de execução transitória: Uma visão geral," in *Revista de Informática Teórica e Aplicada - RITA*, 2018. 19

[17] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches.," in *USENIX Association, new york, USA, p. 897–912*, 2015. 19

[18] S. Khandelwal, "New Spectre (Variant 4) CPU Flaw Discovered—Intel, ARM, AMD Affected." https://thehackernews.com/2018/05/fourth-critical-spectre-cpu-flaw.html, 2018. Online; acessado em 5 de Novembro de 2021. 20

[19] L. Kernel, "Hardware vulnerabilities - Spectre Side Channels." https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/spectre.html, 2018. Online; acessado em 20 de Março de 2022. 21

[20] L. McVoy, S. Graphics, C. Staelin, and H.-P. Laboratories, "lmbench: Portable tools for performance analysis," in *Proceedings of the USENIX 1996 Annual Technical Conference*, 1996. 22

[21] U. C. documentation, G. Morrison, and I. Johnson, "Customising UC20 kernel command line arguments." https://ubuntu.com/core/docs/kernel-options, 2021. Online; acessado em 21 de Dezembro de 2021. 33

[22] U. C. documentation and C. W. Fei, "Image building." https://ubuntu.com/core/docs/image-building, 2021. Online; acessado em 21 de Dezembro de 2021. 33

[23] Intel, "Indirect Branch Restricted Speculation." https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/indirect-branch-restricted-speculation.html, 2018. Online; acessado em 27 de Março de 2022. 34

[24] AMD64_Technology, "Indirect Branch Control Extension - Revision 4.10.18." https://developer.amd.com/wp-content/resources/Architecture_Guidelines_Update_Indirect_Branch_Control.pdf, 2018. Online; acessado em 27 de Março de 2022. 34