Avaliação da vulnerabilidade da arquitetura RISC-V a ataques de execução especulativa

Edgar Luis Brissow
Universidade do Vale do Itajaí, Brasil
edgarbrissow@edu.univali.br

Paulo Roberto Oliveira Valim
Universidade do Vale do Itajaí, Brasil
pvalim@univali.br

Douglas Rossi de Melo
Universidade do Vale do Itajaí, Brasil
drm@univali.br

ABSTRACT

The RISC-V architecture emerged as an academic proposal, backed by the industry, to serve as an alternative to already established architectures. It is an open and free architecture aimed at simplifying implementations. In recent years, vulnerabilities exploiting hardware architectural advancements have been discovered. These advancements have increased the performance of modern processors but simultaneously opened the door to the exploitation of potential security flaws. Many attacks that exploited speculative techniques have arisen, providing means to extract data and execute malicious code, compromising the security and privacy of users of devices susceptible to these attacks. To enhance user security, mitigation measures for these vulnerabilities were proposed. However, these measures often resulted in a significant reduction in performance, making the challenge more complex. This project analyzed and implemented the Spectre vulnerability on the MangoPi board. The MangoPi board, which operated in conjunction with the Fedora operating system, was not susceptible to the attack due to the absence of a cache management instruction. Consequently, the commercial implementations analyzed in this project proved immune to the Spectre vulnerability.

PALAVRAS-CHAVE

Arquitetura de Computadores, Segurança, RISC-V

1 INTRODUÇÃO

A arquitetura de processadores RISC-V está revolucionando a indústria, permitindo personalização de instruções por fabricantes para otimizar tarefas específicas. Empresas como Western Digital, Samsung e Huawei formaram um consórcio para impulsionar essa arquitetura [1]. Enquanto isso, a Intel, conhecida pela arquitetura x86, expande seu portfólio ao adquirir empresas especializadas em RISC-V [2].

No campo da segurança, a execução especulativa, que otimiza o desempenho do software no hardware sem alterações de código, tem sido estudada extensivamente [3]. Embora reduza a latência e o processamento desnecessário, essa técnica enfrenta desafios de segurança, permitindo ações maliciosas que comprometem a proteção de dados, como demonstrado pelas vulnerabilidades Spectre e Meltdown em processadores x86 e ARM. Mitigar essas falhas resultou em custos de desempenho ou modificações no sistema operacional [4].

Descobertas mais recentes, como Fallout [5], RIDL [6] e Zombieload [7] continuam explorando a execução especulativa para atacar a segurança de processadores x86 e ARM. Com a ascensão da arquitetura RISC-V, questiona-se se ela é suscetível às mesmas vulnerabilidades [8].

Neste contexto, este trabalho apresenta uma avaliação das vulnerabilidades de execução especulativa em processadores RISC-V comerciais. Como premissa, foi buscado avaliar as implementações originais utilizando as ferramentas disponibilizadas para os processadores testados.

2 SOLUÇÃO PROPOSTA

Este trabalho utiliza o código do ataque Spectre [4] modificado para que ele possa ser executado na arquitetura RISC-V. O objetivo é através da implementação, transpor a checagem do tamanho do vetor e conseguir ler a memória protegida.

Essa implementação utiliza instruções de manipulação de cache populares nas arquiteturas x86 e ARM, como por exemplo o *cflush* na x86. No entanto, o RISC-V não possui uma instrução equivalente [9]. A Figura 1 representa a modelagem de fluxo do ataque Spectre.

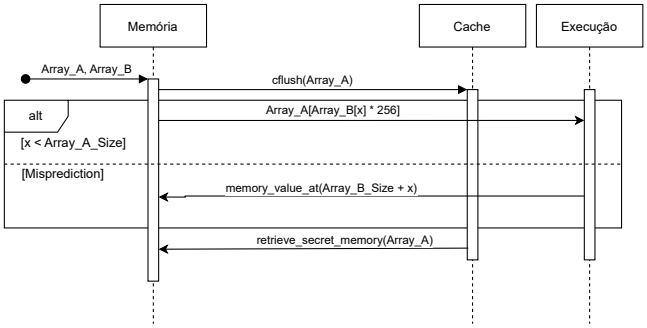

Figura 1: Modelagem do ataque Spectre

No ataque Spectre, um `Array_A` é carregado para a cache através do método de *flushing*, depois os `Array_A` e `Array_B` são carregados nos registradores. O ponto crucial do ataque se dá ao checar os limites do `Array_A`, ao acontecer um atraso no acesso do `Array_B`, a predição de desvios executa o código que faz uma cópia do conteúdo do endereço de `Array_B_Size + x` para o `Array_A` que já está presente na cache. Por fim, quando o processamento do `Array_B` chega ao fim, o estado do cache não é revertido. Por isso, o valor do conteúdo de memória pode ser recuperado da cache verificando qual entrada foi buscada.

Para a implementação do ataque foi utilizado a placa MangoPi, as linguagens C e Assembly e o sistema operacional Fedora.

3 RESULTADOS E ANÁLISE

Um dos requisitos para a implementação do ataque Spectre é explorar os tempos de cache de memória, utilizando dessa informação para conseguir extrair segredos que estão na cache. No código original do ataque Spectre é utilizado a instrução *cflush* para efetuar a limpeza da cache (*flush*) antes do acesso à memória.

Porém, essa instrução é encontrada apenas em processadores de arquitetura x86. A definição da arquitetura RISC-V não contempla uma instrução de limpeza de cache, sendo de responsabilidade dos fabricantes implementar esse tipo de instrução. Dessa forma, foram exploradas algumas alternativas para obter o mesmo resultado de limpeza da cache e obtenção dos tempos de acesso.

Flush por tempo Muitas caches L1 possuem políticas de substituição (por exemplo, LRU - Least Recently Used) que substituirão automaticamente os dados mais antigos;

Flush por alocação Esse método sobrescreve os dados existentes na cache L1, ao tentar carregar e acessar novos dados substituindo os dados antigos;

Flush por sistema operacional Implementação do *flush* de memória no sistema operacional Linux, utiliza uma biblioteca padrão do sistema operacional chamada *cachectl*, que implementa funções para o gerenciamento da cache do processador;

Flush por instrução fence.i A instrução *fence.i* é usada na arquitetura RISC-V para garantir a ordem de execução de instruções, especialmente em cenários envolvendo memória compartilhada e múltiplos núcleos de CPU; e

Flush por instrução de fabricante A arquitetura RISC-V pode ser customizada para atender diversos projetos, no caso do processador estudado há uma implementação do *flush* da cache presente na documentação.

Para descobrir se o processador é vulnerável ao ataque de especulação, foram testados os métodos de *flush* abordados utilizando o trecho de código presente no Quadro 1. O código inicializa um vetor, limpa ele da cache usando o *flush*, e acessa duas posições fixas, mantendo-as na cache, ao final mensura o tempo de acesso a todas as posições do vetor em ciclos.

Quadro 1: Código para verificação do processador

```
  ...
  for(i=0; i<10; i++) array[i*10]=100; # Cria o vetor
  flush();  # impa a cache
  array[3*10] = 100; # Carrega a posicao na cache
  array[7*10] = 200; # Carrega a posicao na cache
  ...
  addr = &array[i*10]; # Acessa a posicao
  time = rdcycle() - time1;  # Calcula o tempo de acesso
  printf("CPU cycles\n", (int)time); Tempo de acesso
}
```

A Figura 2 representa uma série de 10 execuções de cada abordagem de limpeza da cache, mensurando os ciclos levados para o acesso das posições que estão na cache, o eixo Y representa a porcentagem de acerto e o eixo X representa as execuções. A implementação do *flush* por tempo obteve o melhor resultado, mantendo os elementos esperados na cache em 6 das 10 execuções, mas essa taxa de acerto de 60% e o tempo gasto para a limpeza da cache inviabiliza essa abordagem para o uso no ataque, enquanto as outras tiveram menos de 50% de acerto no melhor cenário. Nenhuma das técnicas discutidas pode funcionar como substituta para a instrução *cflush* no ataque original. Portanto, a impossibilidade de medir o tempo de acesso torna inviável a implementação do ataque no processador em questão.

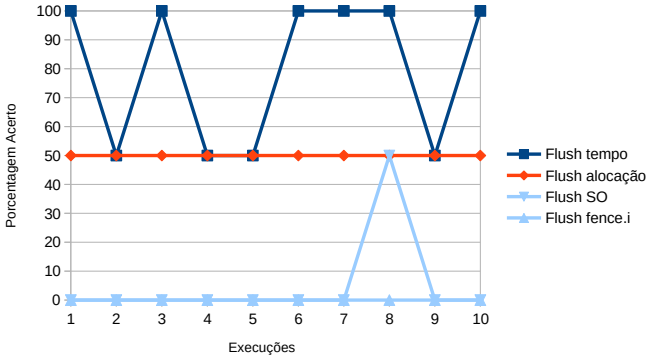

Figura 2: Acertos por execução

4 CONSIDERAÇÕES FINAIS

Este trabalho analisa a viabilidade de explorar vulnerabilidades, como o Spectre, na arquitetura processadores de especificação aberta RISC-V. Foi utilizada a placa MangoPi para a avaliação, por essa possuir os recursos adequados para implementar com sucesso o ataque.

Os resultados revelaram que instruções alternativas para o gerenciamento de cache e medição de tempo de acesso não proporcionaram os mesmos resultados que instruções específicas para esses fins, dificultando a obtenção de dados de acesso aos recursos. A personalização de instruções na arquitetura RISC-V limitou a eficácia das vulnerabilidades, exigindo um conhecimento detalhado das arquiteturas específicas e a superação de medidas de segurança já presentes nos sistemas operacionais para mitigar acessos não autorizados à memória.

Para trabalhos futuros, propõe-se a exploração de vulnerabilidades em diversas arquiteturas de processadores, diferentes versões do kernel Linux, o aprimoramento de estratégias de mitigação para arquiteturas abertas como RISC-V, a análise do impacto em ambientes de produção e a criação de ferramentas de detecção avançadas.

REFERÊNCIAS

[1] RISC-V Foundation. Member directory, 2019. URL https://riscv.org/membership/. Acesso em: 14 jun. 2019.

[2] Mark Gurman, Edward Ludlow, Ian King, and Katie Roof. Chipmaker sifive is said to draw intel takeover interest, Jun 2021. URL https://www.bloomberg.com/news/articles/2021-06-10/chipmaker-sifive-is-said-to-draw-intel-takeover-interest.

[3] Ronaldo Augusto de Lara Goncalves. Arquiteturas multi-tarefas simultâneas: Sempre: arquitetura smt com capacidade de execução e escalonamento de processos. 2000.

[4] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[5] Marina Minkin, Daniel Moghimi, Moritz Lipp, Michael Schwarz, Jo Van Bulck, Daniel Genkin, Daniel Gruss, Berk Sunar, Frank Piessens, and Yuval Yarom. *Fallout: Reading Kernel Writes From User Space*, 2019.

[6] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. In *S&P*, May 2019.

[7] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-privilege-boundary data sampling. In *CCS*, 2019.

[8] Jeff Dean, David Patterson, and Cliff Young. *A new golden age in computer architecture: Empowering the machine-learning revolution*, 38(2):21–29, 2018.

[9] David Patterson and Andrew Waterman. *The RISC-V reader: An open architecture atlas.* Strawberry Canyon, 2017. ISBN 099924910X, 9780999249109.