# AutoGuard: A Secure Implementation of the Conditional Branch Instruction

Zeru Lan[1], Chunlu Wang[1], Pengfei Qiu[1,3*], Yu Jin[1], Yihao Yang[1], Dongsheng Wang[2,3], Gang Qu[4]
[1]Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education
[2]Tsinghua University, [3]Zhongguancun Laboratory, [4]University of Maryland, College Park

***Abstract*—Side-channels are one of the major security threats to processors, which use the unintentional leakage of time, power consumption, and electromagnetic radiation from computers as a basis for obtaining important secret data in the computers. The Branch Target Buffer (BTB) and Branch History Buffer (BHB) side-channels are the recently discovered side-channel vulnerabilities that can be utilized to implement many high-threat hardware vulnerability attacks like Spectre and BranchScope. However, although there are many effective defense methods for BTB-based and BHB-based side-channel attacks, the relative isolation of each method from each other seriously affects the processor's performance and also increases resource consumption.**

**In this paper, we propose a secure conditional branching architecture (we call it AutoGuard), which is a novel mitigation mechanism for BTB-based and BHB-based Side Channel Attacks (SCAs). Meanwhile, AutoGuard is also capable of effectively mitigating Spectre attacks (including variant 1 and variant 2). AutoGuard first ensures that only the correct addresses can be executed during the conditional branch by address blocking and adding partial instruction dependencies. Subsequently, AutoGuard modifies the microarchitectural execution process of the partial conditional branch to safely execute instructions with the assistance of the Return Stack Buffer (RSB). Finally, AutoGuard determines the final instruction to be executed by determining the instruction address at the top of the RSB stack and the top of the execution stack. Through the above process, we design a security gadget in AutoGuard that can automatically identify and modify conditional branches to improve their security. We successfully deployed AutoGuard on multiple processors and Linux kernels. Through experimental evaluation, we find that AutoGuard consumes fewer resources and has a lesser performance impact than other existing BTB-based and BHB-based side-channel attack protection methods.**

***Index Terms*—Side-channel Attack, Micro-architecture, Transient Execution, Branch Instruction**

## I. Introduction

The processor is the hardware foundation of the computer system, and its security vulnerability will directly affect the security of the entire computer system. With the widespread use of computers, there is a huge pressure on the performance of the processor. To solve this problem, modern processors have introduced a variety of performance optimization techniques to improve performance, such as out-of-order execution and speculative execution. Although the performance of the processor has improved significantly, processor design lacks the consideration of security. Attackers can exploit the hardware vulnerabilities introduced by these optimization techniques to achieve attacks and obtain secret data from the processor, thus defeating the processor's security protection from the inside.

Processor vulnerabilities are not available to leak data directly and require the exploitation of side-channel attacks to leak data [1], [2]. Side-channel attacks are a type of attack method that uses the time, power consumption, and electromagnetic radiation of the hardware as a basis for obtaining important and confidential information from the computer. Cache side-channel attacks are the most exploited side-channel attacks and there are some effective defenses against them [1], [2]. In order to bypass the defenses against cache side-channel attacks and achieve data recovery, attackers have discovered that other microarchitectural components can also implement side-channel attacks, such as the Branch Target Buffer (BTB) and the Branch History Buffer (BHB). Existing studies have shown that BTB and BHB also have corresponding side-channel information and these can be exploited by attackers to recover sensitive data [3], [4]. The most powerful Spectre variant 1 attacks and variant 2 attacks can exploit the side-channel information of these two components to implement attacks [3].

Currently, there are several mitigation approaches based on software and hardware that have been proposed for BTB-based and BHB-based side-channel attacks. The main side-channel attack protection methods presently available are adding obfuscation, randomizing instruction delays, and preventing the loading of sensitive information from shared microarchitectural components, etc [5], [6]. Although there are many effective defense methods available, each is relatively isolated from the other. The specialization of the different defense methods causes a significant consumption of hardware resources and also has a serious impact on processor performance. As a result, it is necessary to design a generalized BTB-based and BHB-based side-channel attack defense approach with low hardware resource consumption and low performance impact.

In this paper, we propose AutoGuard, a new method that can improve the security of conditional branch instruction and can effectively defend against BTB-based and BHB-based side-channel attacks while guaranteeing maximum performance. Our approach starts with blocking the potentially executable instructions and then loading the corresponding addresses into different registers. After the conditional judgment is completed, secure instruction execution is realized in combination with the Return Stack Buffer (RSB). Furthermore, we extend the basic AutoGuard to a multi-branch structure to realize safe

*Corresponding author

access to multiple loops. Eventually, we successfully deployed AutoGuard in LLVM and implemented a security gadget that automatically identifies and modifies conditional branches to improve their security.

We evaluate AutoGuard on multiple microarchitectures. Our experimental results demonstrate that AutoGuard can effectively mitigate BTB-based and BHB-based side-channel attacks on the Intel and AMD processors. Our method can save 34% execution time over the method based on `lfence` [7]. Compared with Retpoline [8], our method can additionally mitigate BTB-based and BHB-based side-channel attacks with a small performance difference.

In summary, this paper makes the following contributions:

- We propose a software-based secure implementation of the conditional branch instruction (we call it AutoGuard) that can effectively mitigate BTB-based and BHB-based side-channel attacks while guaranteeing maximum performance improvement and minimum computational consumption. Meanwhile, AutoGuard also effectively mitigates Spectre attacks (including variant 1 and variant 2).
- We implement AutoGuard on existing processors and extend the basic single-branch structure of the safety implementation to a multi-branch structure.
- We successfully deployed AutoGuard in LLVM and implemented a security gadget that automatically identifies and modifies conditional branches to improve their security.
- We perform a detailed experimental evaluation of AutoGuard. The experimental results show that our approach has less performance impact and consumes fewer hardware resources than other approaches.

## II. Background

### A. BTB-based Side-Channel Attacks

The Branch Target Buffer (BTB) stores the mapping between the address of the most recently executed branch instruction and the target address [3]. Current research has demonstrated that BTB can be utilized to implement side-channel attacks, in which the attacker trains the branch target buffer to incorrectly predict the branch from an indirect branch instruction to the gadget address, which leads to speculative execution of the gadget [3].

The BTB-based side channel attack is divided into three main stages. In the first stage (data preparation), the attacker trains the branch prediction results in advance for exploiting faulty speculative execution in subsequent attacks and then loads the secret data from memory. In the second stage (data transfer), the processor speculatively executes instructions to transfer the secret data from the victim context to the microarchitecture covert channel. In the third stage (data recovery), the attacker recovers the secret data using side-channel information stored in the microarchitecture layer.

### B. BHB-based Side-Channel Attacks

The Branch History Buffer (BHB) is a cache that stores the latest destination addresses for different branches and branch contexts [3]. The address tags for this cache are computed from the branch source address and the branch context. To create the context, the branch predictor takes the source address and the history of previously executed branches into account and stores them as hashes in the BHB.

Since the target address of an indirect branch is only available at runtime, modern processors try to predict the branch target and speculatively fetch and execute instructions at the predicted location [4]. Based on this, attackers can exploit this behavior to implement BHB-based side-channel attacks by misleading the attacked indirect branch to predict the target address of the contained gadget, thus disclosing secret data.

## III. Motivation and Threat Model

In this section, we describe the research motivation and threat model.

### A. Motivation

Currently, there are several mitigation approaches based on software and hardware that have been proposed for BTB-based and BHB-based side-channel attacks. Although there are many effective defense methods available, each is relatively isolated from the other. The specialization of the different defense methods causes a significant consumption of hardware resources and also has a serious impact on processor performance. As a result, it is necessary to design a generalized BTB-based and BHB-based side-channel attacks defense approach with low hardware resource consumption and low performance impact.

### B. Threat Model

In order to better study the defense approach of BTB-based and BHB-based side-channel attacks, we have made corresponding assumptions about the attack targets and scenarios of exploiting side-channel attacks in this study, as described in the following.

**Attack Target.** We assume that the attacker knows the secret data memory address but does not have permission to access it. Therefore, the attacker's goal is to load the secret data by utilizing speculative execution and then recover the information.

**Attack Scenario.** To consider the generality of the attack, we do not restrict the location of the attacker. The attacker can attack across cores, can utilize hyperthreading, and can even attack at a distant node. Our goal in these scenarios is to effectively prevent leakage of the victim's secret by implementing secure conditional branch instructions. In the next subsections, we will describe in detail the implementation of secure conditional branch instructions to design an efficient defense mechanism against BTB-based and BHB-based side-channel attacks.

## IV. AutoGuard

This section describes the design, implementation and deployment details of AutoGuard.

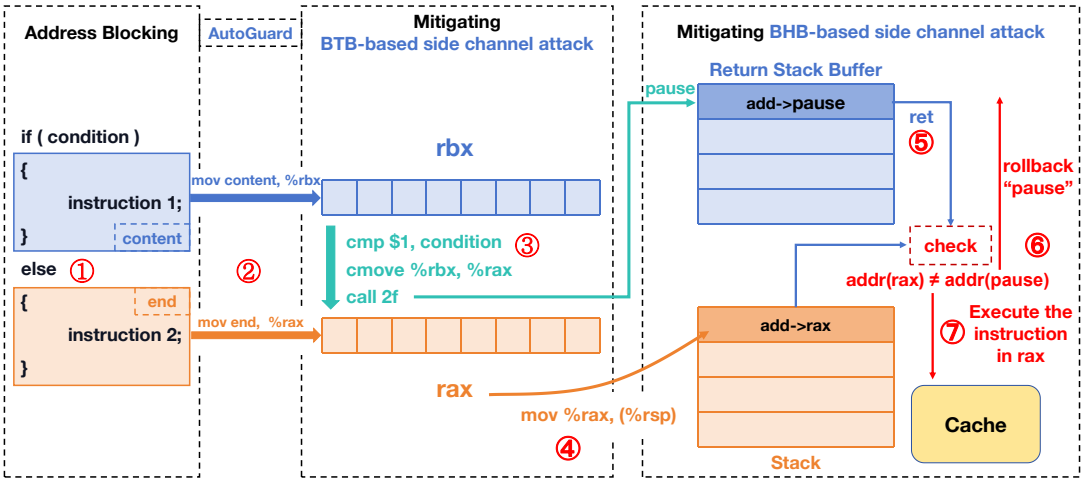


Fig. 1. **The working mechanism of AutoGuard.** ① Address blocking. ② Load the block address into different registers. ③ Determine the branch of subsequent execution and load the next instruction of the call instruction into the RSB. ④ Load the address of the branch of subsequent execution into the execution stack. ⑤ Call the ret instruction. ⑥ Compare the instructions at the top of the two stacks. ⑦ Execute the final executed instruction.

### *A. Design*

The design principle of AutoGuard is to effectively defend against BTB-based and BHB-based side-channel attacks while ensuring maximum processing performance and minimizing computational resource consumption. The overview of AutoGuard is shown in Fig. 1.

```
1  void *p = &&end, *q = &&content;
2  asm volatile(
3      "mov %0, %%rax\n\t"
4      "mov %1, %%rbx\n\t"
5      "cmp $1, %2\n\t"
6      "cmove %%rbx, %%rax\n\t"
7      "call 2f\n\t"
8      "1:\n\t"
9      "pause\n\t"
10     "2:\n\t"
11     "mov %%rax, (%%rsp)\n\t"
12     "ret\n\t"
13     :
14     : "r"(p), "r"(q), "r"(condition)
15     :);
16 content:
17     temp &= array2[array1[x] * 512];
18 end:
19     asm volatile("nop");
```

**Listing 1. Pseudocode for AutoGuard**

**Mitigating BTB-based Side-Channel Attacks.** As described in Section II, the traditional BTB-based side-channel attack exploits processor speculative execution to load the victim's secret data. As a result, there are two security risks in the execution of conditional branch instructions that cause the attacker can successfully implement the BTB-based side-channel attack. The first risk is that the conditional branch instruction discloses the address of each instruction during execution, which can be used directly by the attacker. The second risk is that the conditional branch instruction has a speculative execution optimization technique, which may lead to the early execution of instructions that do not satisfy the conditions, and these instructions may be attack instructions that access sensitive data.

In order to solve the security risks caused by the speculative execution of conditional branch instructions, we provide two optimizations to the original conditional branching instructions. The first optimization is to block the instructions that may be executed. We first divide the instructions executed on the different conditional branches into two code blocks. When executing the conditional branching judgment, the addresses of these two code blocks are loaded into the registers respectively. This method can ensure that in the subsequent execution process, we only operate on the address of the block, but not on the address of the instruction within the block. By utilizing this method, we can also prevent the attacker from directly accessing the secret address.

The second optimization is to use the `cmp+cmove+jmp` instruction to replace the original instruction execution process [9]. Conditional instructions such as `cmov` convert conditional branches into sequential code, thus effectively converting control dependencies into data dependencies [9]–[11]. The `cmp` instruction will add execution dependency to `cmov` instruction, so that the `cmove` instruction must be executed after the `cmp` conditional judgment is completed. Therefore, the `cmove` instruction will be forced to serialize during the execution process, which can prevent the attacker from loading the victim's secret data before the completion of the conditional branching judgment. When the `cmove` instruction has finished executing, we use the `jmp` instruction to jump to the address of the subsequent instruction. However, we found that utilizing the `jmp` instruction opens a transient window that can be trained, which can be utilized by the attacker to implement the BHB-based side-channel attack. In the following subsection, we will describe how to avoid the threat of BHB-based side-channel attacks introduced by the `jmp` instruction.

**Mitigating BHB-based Side-Channel Attacks.** To handle the threat of BHB-based side-channel attacks caused by the `jmp` instruction, we replace the `jmp` instruction with the `call+ret` instruction. The idea is to select a secure instruction (e.g., the `pause` instruction) to replace the vulnerable instruction, and then load the address of this secure instruction into the Return Stack Buffer (RSB) by utilizing the `call` instruction. When executing the `ret` instruction, the processor will speculatively execute the address of the security instruction loaded into the RSB earlier. With this design, we can effectively prevent attackers from utilizing the transient window opened by the `jmp` instruction to obtain data to mitigate the threat of BHB-based side-channel attacks.

Our method has similarities with the Retpoline [8] method in mitigating Spectre variant 2 attacks, but there are also some differences. First, the main difference is that our method can additionally mitigate the attack of BTB-based and BHB-based side-channel attacks. Second, our method stalls in the RSB for a shorter time than Retpoline. Finally, since the instructions submitted to the stack by AutoGuard satisfy the execution conditions, our approach can minimize the state rollback caused by incorrectly speculated execution.

```
1  void *p = &&end, *q = &&content, *pq = &&elseif;
2  asm volatile(
3      "mov %0, %%rax\n\t"
4      "mov %1, %%rbx\n\t"
5      "mov %2, %%rcx\n\t"
6      "cmp $1, %3\n\t"
7      "cmove %%rbx, %%rax\n\t"
8      "call 3f\n\t"
9      "1:\n\t"
10     "cmp $1, %4\n\t"
11     "cmove %%rcx, %%rax\n\t"
12     "call 3f\n\t"
13     "2:\n\t"
14     "pause\n\t"
15     "3:\n\t"
16     "mov %%rax, (%%rsp)\n\t"
17     "ret\n\t"
18     :
19     : "r"(p),"r"(q),"r"(pq),"r"(cond1),"r"(cond2)
20     :);
21 content:
22     temp &= array2[array1[x] * 512];
23 elseif:
24     asm volatile("nop");
25 end:
26     asm volatile("nop");
```

**Listing 2. Pseudocode for 3-branch AutoGuard**

### B. Implementation

**AutoGuard Implementation.** In Section IV-A, we introduce the design idea of AutoGuard in detail. According to the design idea, we implemented AutoGuard on multiple x86 processors and Linux kernels, and the specific implementation is shown in Listing 1. The first step is to block the instructions. We begin by blocking the instructions in both cases of conditional branches (lines 16-19). The second step is to perform BTB-based side-channel attack mitigation (lines 5-6). We utilize the `cmp` instruction to add a dependency to the `cmove` instruction to prevent the attacker from loading the victim's secret data by using the conditional branch's speculative execution. The third step is to perform BHB-based side-channel attack (lines 7-12). With the `call` instruction, we can load the `pause` instruction into the RSB. With the instruction at line 11, we push the address of the block of code we want to execute onto the stack. When we reach line 12, the `ret` instruction takes the address from the top of the RSB stack and compares it to the address pushed onto the stack at line 11. When the two addresses are not the same, the processor assumes that the RSB has made an error in its speculative execution and chooses to execute the instruction pushed into the stack at line 11. In this way, we can control the transient execution window during RSB speculative execution and prevent the attacker from realizing the attack.

**3-Branch AutoGuard Implementation.** In the previous subsection, we described the implementation of AutoGuard on multiple x86 processors and Linux kernels. On this basis, we enhance the safety of the multi-branch conditional branch instruction by extending the basic branch structure to a multi-branch structure based on the implementation of AutoGuard, as shown in Listing 2. We divide the three branches into three code blocks and load the addresses of these blocks into three registers. Then, we load the address that satisfies the execution condition into the stack. Finally, the processor continues to execute subsequent instructions.

### C. Deployment

We successfully deployed AutoGuard on the Linux system. Our choice of compilation toolchain was based on Clang/LLVM 17, and all libraries used in our tests were compiled using the AutoGuard compiler, which supports both static and dynamic linking to the protected application. We also designed a gadget that can automatically identify conditional branching instructions and modify unsafe branching instructions to safer alternatives. We have also successfully deployed it in the LLVM compilation environment, where it can be invoked via compilation options.

## V. Evaluation

In this section, we present the evaluation scheme design and evaluation results of AutoGuard.

### A. Setup

We evaluate the security and performance of the AutoGuard by comparing the following approaches: unmitigated, the method based on `lfence` [7] and Retpoline [8]. Unmitigated means that no mitigation schemes for the side-channel attacks are employed. The method based on `lfence` is to force the serialization of read instructions so that the processor must perform the data read operation after the conditional judgment is completed. AutoGuard is our proposed method. Finally, Retpoline is an effective way to mitigate Spectre variant 2 attacks using RSB. In the following subsections, we will describe the evaluation in detail.

### B. Evaluation of Security

We evaluate the security of AutoGuard by deploying it to multiple processors of x86 architecture, and the experimental results are shown in Table I. By observing Table I, we can conclude that AutoGuard can effectively mitigate BTB-based and BHB-based side-channel attacks on both Intel and AMD processors, which fully verifies the security of our method.

TABLE I: Results of the AutoGuard security evaluation.

| CPU | Vendors | BTB SCAs | BHB SCAs | Spectre v1 & v2 |
|---|---|---|---|---|
| i7-6700 | Intel | ✓ | ✓ | ✓ |
| i7-7700 | Intel | ✓ | ✓ | ✓ |
| i5-7300u | Intel | ✓ | ✓ | ✓ |
| i9-13900 | Intel | ✓ | ✓ | ✓ |
| i5-12450H | Intel | ✓ | ✓ | ✓ |
| Ryzen 5 5600G | AMD | ✓ | ✓ | ✓ |
| Ryzen 5 7600 | AMD | ✓ | ✓ | ✓ |

During our experiments, we found that Intel's Control-flow Enforcement Technology (Intel CET) [12] also has some security features. The principle is to use `ecrh64` instruction to increase the security of indirect branches, but the response speed is slow. Furthermore, we can still recover some secret data normally during the experiment with Intel CET enabled, so Intel CET is not completely secure. Compared with Intel CET, our method will have a faster response speed, and we can guarantee that no secret data will be recovered.

### C. Performance of Synthetic Workloads

We first evaluate the performance of AutoGuard with reference to the synthetic benchmark of the SpectreGuard [13] method. The benchmark consists of two parts, the first part is the client workload (S). The second component is the background communication activity (C) using the AES algorithm for data encryption.

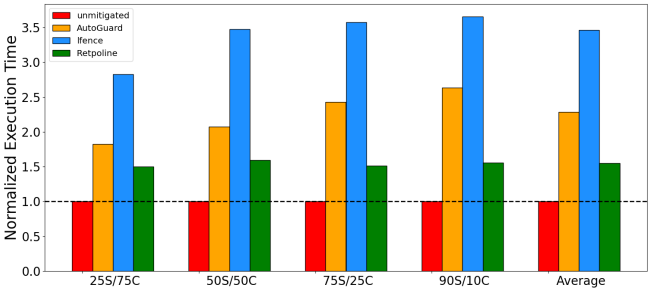


Fig. 2. Execution times of synthetic workloads. (90S/10C indicates that 90% of the time is spent in the first component and 10% in the second component.)

Fig. 2 shows the execution time of the synthetic workload for AutoGuard. By observing Fig. 2, we can find that AutoGuard significantly better than the method based on `lfence` [7]. It is experimentally verified that our method can save 34% execution time over the method based on `lfence`. Moreover, in multiple cases, the performance of AutoGuard is closer to Retpoline [8], which is the most effective method to mitigate the Spectre variant 2 attack. However, our method can mitigate not only Spectre variant 2 attacks, but also BTB-based and BHB-based side channel attacks and Spectre variant 1 attacks.

### D. Performance of Conditional Branch Instructions

We use two schemes to test the performance of the conditional branching instructions implemented in AutoGuard. The first scheme is to test the time consumed by the overall program when a large number of instructions are executed. We executed about 30000 instructions and executed them 100 times to calculate the average. The second scheme is the average time taken to execute an `if` branch instruction. We counted the execution time of 10000 identical instructions and took the average. To prevent micro-architectural components such as Cache from affecting the evaluation, we remove noise by flushing micro-architectural residual states before each instruction execution.

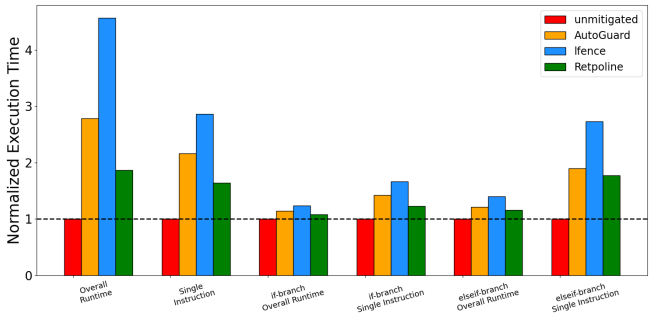


Fig. 3. Time evaluation of conditional branch instructions.

Fig. 3 shows the results. By observing the results, the performance of AutoGuard is superior. Compared to the method based on `lfence`, our method will save 39% of the overall time consumed in executing 30000 instructions and 24% of the time in executing each instruction on average. Since our method forces a shorter waiting time, this will minimize the performance loss. Compared to Retpoline, our method consumes a longer time, but it is also in our expectation. This is because AutoGuard needs to consume some time in mitigating BTB-based side-channel attacks, which is not possible with Retpoline.

### E. Performance of SPEC2017 Workloads

We also conducted experiments using a set of SPEC CPU2017 benchmarks. We tested the performance measurements of the unmitigated, `lfence`-based approach, Intel CET and our proposed approach in our experiments, and the results are shown in Fig.4.

The results show that our approach is effective and that there is a performance-security trade-off when using AutoGuard. For real-world application environments, we can further reduce the overhead by not modifying the conditional branches that do not involve sensitive data, and only modifying the insecure conditional branches.

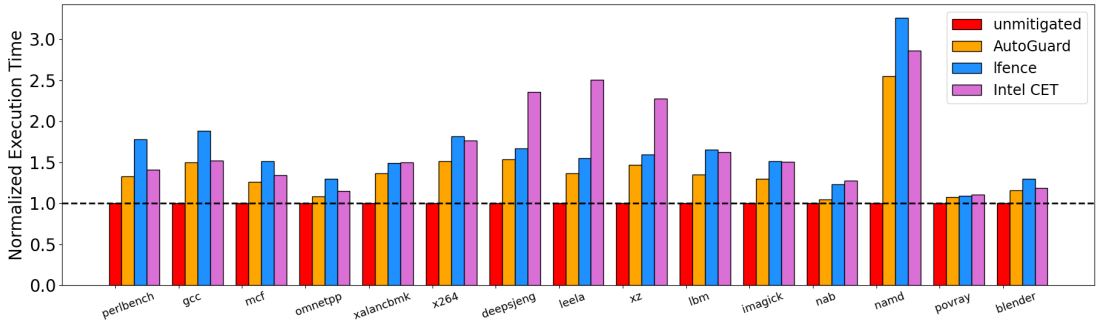


Fig. 4. Execution times of SPEC2017 workloads.

## VI. Related Work

The work related to our work is Switchpoline [14], the first automated Spectre-BTB and Spectre-BHB software solution that protects C and C++ userspace applications on ARMv8 from all variants of Spectre-BTB and Spectre-BHB. Our proposed AutoGuard differs from it in two ways, the first difference is the different target of protection. We focus our defense around side-channel attacks, whereas Switchpoline defends against Spectre attacks. The second difference is the chip architecture; our approach works on Intel and AMD x86 architectures, while Switchpoline works on ARMv8 architectures. As a result, our work is more generalizable.

## VII. Conclusion

In this paper, we propose AutoGuard that can effectively mitigate BTB-based and BHB-based side-channel attacks. AutoGuard can make security considerations and protections while ensuring maximum performance optimization. We utilize the `cmp+cmove` instruction to mitigate the secret loading from conditional branch prediction and utilize the `call+ret` instruction to mitigate the transient execution window opened by the `jmp` instruction to avoid introducing new attacks. We deployed our approach on Intel and AMD processors to evaluate the performance of our approach by experimentally comparing it with existing methods based on `lfence` [7], Retpoline [8] and Intel CET [12]. Our evaluation results show that our approach can effectively mitigate BTB-based and BHB-based side-channel attacks without consuming excessive hardware resources.

**Acknowledgements.** Authors from BUPT and Tsinghua University are supported in part by the Beijing Natural Science Foundation (Grant No. 4242026), National Natural Science Foundation of China (Grant No. 62072263 and 62372258), and the Fundamental Research Funds for the Central Universities (Grant No. 2023RC71). Furthermore, we gratefully thank the anonymous reviewers for their helpful comments and Haojie Zhang for his works in the experiments.

## References

[1] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, l3 cache Side-Channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[2] Y. A. Younis, K. Kifayat, Q. Shi, and B. Askwith, "A new prime and probe cache side-channel attack for cloud computing," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 2015, pp. 1718–1724.

[3] 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.

[4] E. Barberis, P. Frigo, M. Muench, H. Bos, and C. Giuffrida, "Branch history injection: On the effectiveness of hardware mitigations against Cross-Privilege spectre-v2 attacks," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 971–988. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/barberis

[5] Z. Xu, L. Yin, Y. Lyu, H. Wang, G. Qu, and D. Wang, "Cacheguard: A behavior model checker for cache timing side-channel security: (invited paper)," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 19–24.

[6] F. Mosquera, K. Kavi, G. Mehta, and L. John, "Guard cache: Creating noisy side-channels," *IEEE Computer Architecture Letters*, vol. 22, no. 2, pp. 97–100, 2023.

[7] A. Milburn, K. Sun, and H. Kawakami, "You cannot always win the race: Analyzing the lfence/jmp mitigation for branch target injection," 2022. [Online]. Available: https://arxiv.org/abs/2203.04277

[8] M. F. Abdul Kadir, J. K. Wong, F. Ab Wahab, A. F. A. Abidin Bharun, M. A. Mohamed, and A. H. Zakaria, "Retpoline technique for mitigating spectre attack," in *2019 6th International Conference on Electrical and Electronics Engineering (ICEEE)*, 2019, pp. 96–101.

[9] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 557–574. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-sangho

[10] Y. Choi, A. Knies, L. Gerke, and T.-F. Ngai, "The impact of if-conversion and branch prediction on program execution on the intel itanium processor," in *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34*. IEEE Computer Society, 2001, pp. 182–182.

[11] O. Oleksenko, B. Trach, T. Reiher, M. Silberstein, and C. Fetzer, "You shall not bypass: Employing data dependencies to prevent bounds check bypass," *CoRR*, vol. abs/1805.08506, 2018. [Online]. Available: http://arxiv.org/abs/1805.08506

[12] M. Kucab, P. Boryło, and P. Chołda, "Performance impact of control flow enforcement technology (cet)," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2022, pp. 96–100.

[13] J. Fustos, F. Farshchi, and H. Yun, "Spectreguard: An efficient data-centric defense mechanism against spectre attacks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[14] M. Bauer, L. A. Hetterich, C. Rossow, and M. Schwarz, "Switchpoline: A Software Mitigation for Spectre-BTB and Spectre-BHB on ARMv," 7 2024. [Online]. Available: https://publications.cispa.de/articles/conference_contribution/Switchpoline_A_Software_Mitigation_for_Spectre-BTB_and_Spectre-BHB_on_ARMv/25304857