2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

Modeling, Derivation, and Automated Analysis of Branch Predictor Security Vulnerabilities

Quancheng Wang[†], Ming Tang[†][*], Ke Xu[†], Han Wang[†]
[†] Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education,
School of Cyber Science and Engineering, Wuhan University, Wuhan, 430072, China
{wangquancheng, m.tang, kexuwhu, han.wang}@whu.edu.cn

***Abstract*—With the intensification of microarchitectural side-channel attacks targeting branch predictors, the security boundary of computer systems and users' security-critical data are under serious threat. Since the root cause of these attacks is the neglect of security issues in the microarchitecture design of branch predictors, an analysis framework that can exhaustively and automatically explore these concerns in the design phase is imminent. In this paper, we propose a comprehensive and automated evaluation framework for inspecting the security guarantees of branch predictors at the microarchitecture design stage. Our technique involves a three-step modeling approach that abstractly characterizes 19 branch predictor states and 53 operations that could affect these states. Subsequently, we develop a symbolic execution-based framework to investigate all three-step combinations and derive 156 valid attack patterns against branch predictors, including 89 novel attacks never considered in the previous work. Finally, we apply our framework to 8 secure branch predictor designs and four typical hardware-based countermeasures against speculative execution attacks to evaluate their security capabilities. The result demonstrates that these security branch predictors provide efficient security guarantees and outperform those hardware-based alleviations against speculative execution attacks, indicating that the security branch predictors are promising in mitigating branch predictor security vulnerabilities.**

I. Introduction

In recent years, microarchitecture design has undergone continuous optimization and improvement, resulting in significant performance increases for modern processors. Caches and TLBs reduce memory access latency and virtual memory translation overhead, thus alleviating the performance bottleneck between the CPU and memory. Additionally, techniques such as out-of-order execution and branch prediction improve instruction-level parallelism, further increasing the performance of modern processors.

However, the researchers have found that the timing differences caused by these microarchitectural optimizations can be exploited to construct microarchitectural timing side-channel attacks [51] to leak sensitive information and cryptographic keys of the victim processes. In particular, the Spectre attack [40] and its variants [5], [42], [45], [49], [58], which exploit the branch predictor to breach security boundaries of computer systems, have attracted widespread attention in academia and industry. Moreover, these branch predictor attacks can cause more severe consequences than traditional side-channel attacks, such as stealing RSA encryption keys [3], bypassing KASLR [25], and breaking the SGX security boundary [17]. Therefore, it is crucial to analyze the security of branch predictors in modern processors and mitigate the security vulnerabilities of branch predictors to ensure the confidentiality and integrity of computer systems.

[*]Corresponding author.

Although researchers have proposed various analysis and verification methods against microarchitectural attacks against branch predictors [6], [15], [19], [28], [31]–[33], [46], [47], [57], there are still several limitations in the previous work. Most work focuses on software security, such as Spectector [31], KLEESpectre [57], and SpecuSym [33]. While these methods model speculative execution at the software level and analyze the code security against speculative attacks, they only cover a subset of branch predictor security issues, are limited to specific applications (e.g., cryptographic libraries), and can only perform security analysis on known types of attacks.

Moreover, although Hu et al. [36] and He et al. [34] construct attack models to analyze the security of branch predictors from the hardware perspective, these models only cover known speculative attacks against branch predictors without deriving new side-channel attacks against branch predictors, such as BHI [5]. They also do not consider timing side-channel attacks and covert-channel attacks against branch predictors, which pose a real-world security threat to computer systems.

In addition, Deng et al. [20]–[22] propose a three-step modeling approach to analyze security vulnerabilities in caches and TLBs. They finally derive 88 timing-based attacks with 32 new attacks against caches and 24 timing-based attacks with 16 novel attacks against TLBs, highlighting the insufficiency of analyzing only previously discovered security vulnerabilities, as there may be many unknown microarchitectural side-channel attacks. Unfortunately, these methods cannot be directly applied to branch predictors due to differences in component designs and working principles.

To address these limitations, this paper proposes a comprehensive and automated security analysis framework for verifying security guarantees of branch predictors at the microarchitecture design stage. We introduce a new modeling approach for evaluating side-channel security properties of branch predictors and leverage symbolic execution to derive all potential security issues for a given branch predictor design.

We first analyze the fundamental workflow of the Pattern History Table (PHT), Branch Target Buffer (BTB), Branch

History Buffer (BHB), and Return Stack Buffer (RSB), which are the four main components of branch predictors in modern processors. Then, we abstractly define 19 different states of these four components and reduce the complexity of the subsequent analysis by abstracting the branch predictor components to the minimum units (e.g., a single entry of the PHT). Based on the abstracted 19 states of the branch predictor components, we propose a three-step modeling approach against branch predictors and model 53 operations of the attacker and the victim that could affect the state of the branch predictor in the three-step model, such as operations that change the state of the branch predictor and that observe the timing differences.

Afterward, we build a prototype framework based on symbolic execution to analyze all possible three-step combinations, which takes the 53 operations as input and outputs all vulnerable attack patterns. As a result, we derive 156 valid three-step attack patterns against branch predictors, including 89 attacks never considered in the previous work. Meanwhile, we also categorize these attacks into four different types, including hit-based internal interference attacks (IH), hit-based external interference attacks (EH), miss-based internal interference attacks (IM), and miss-based external interference attacks (EM), based on timing information observed by the attacker and the operations associated with the security-critical branches.

Finally, we model 8 existing secure branch predictor designs [12], [16], [43], [52], [56], [64]–[66] proposed in academia and conduct an automated security analysis of these designs using our proposed security analysis framework. Among these security branch predictors, Probabilistic Saturation Counter (PSC) [64] and HyBP [66] are the most effective secure branch predictors for mitigating PHT and BTB security vulnerabilities under ideal circumstances, respectively. Then, HyBP [66] also outperforms in mitigating known and newly derived attacks among these security branch predictors. Furthermore, we model four typical hardware countermeasures [39], [53], [60], [62] against speculative execution attacks and compare them with secure branch predictors. The evaluation result shows that Two-Level Encryption [43], Noisy-XOR-BP [65], LS-BP [16], and HyBP [66] can provide better security guarantees against speculative execution attacks than these hardware defenses.

The main contributions of this paper are as follows:

- We propose a three-step modeling approach for evaluating the security properties of branch predictors at the microarchitecture design stage. Our technique abstractly characterizes 19 branch predictor states and 53 operations of the attacker and victim that could affect these states.
- We develop a comprehensive and automated evaluation framework based on the proposed model that leverages symbolic execution to analyze all potential three-step combinations, yielding 156 valid attack patterns against branch predictors, with 89 novel attacks never discovered.
- We apply our security analysis framework to 8 existing secure branch predictor designs and four typical hardware alleviations against speculative execution attacks, and the results show that secure branch predictors are promising solutions for enhancing the security of branch predictors.

II. Background

In this section, we provide pertinent information on the design of branch predictors in modern processors, existing microarchitectural attacks against branch predictors, and related work on microarchitectural security analysis.

A. *Overview of Branch Predictors*

Branch predictors resolve pipeline stalls caused by control hazards [35]. There is no performance loss if the prediction is correct, and if the prediction is incorrect, the processor will roll back to the correct branch path for execution [40]. Although there are differences in the design details of branch predictors for different types of processors, they primarily work with similar principles and structures. Therefore, we provide an overview of the Pattern History Table (PHT), Branch Target Buffer (BTB), Branch History Buffer (BHB), and Return Stack Buffer (RSB) that have security issues in modern processors based on existing reverse-engineering works and attacks [27], [38], [40], [42], [45], [63] on branch predictors.

Pattern History Table (PHT). Pattern History Table (PHT) predicts the direction of conditional branches [40], and each of its entries is typically a 2-bit finite state machine [35], where two states indicate not-taken (00 and 01) and the other two states denote taken (10 and 11).

Branch Target Buffer (BTB). Branch Target Buffer (BTB) predicts the target address of a branch [40] and is organized similarly to a set-associative cache, which predicts branch targets based on set and tag matching [63]. Each entry in the BTB contains the address of a branch instruction and the target address of the branch instruction, both of which are compressed by a hashing algorithm.

Branch History Buffer (BHB). Branch History Buffer (BHB) records the history of the last few branches and is usually implemented as a global shift register. In branch prediction, the value in the BHB is logically manipulated with the branch instruction's address and used as an index to access the PHT or BTB to predict the branch address [63].

Return Stack Buffer (RSB). Return Stack Buffer (RSB) predicts the return address of function calls and is implemented as a stack in modern processors. When a function is called, the following return address is pushed into the RSB for future prediction. Then, when a return instruction is executed, the return address is popped out of the RSB as the predicted target address [42], [45].

Predicting Target Address. The branch predictor predicts the target address of branch instructions based on the PC value and the state of the above four units. Figure 1 illustrates the workflow for the prediction of different branch instructions. Then, the PHT entry is updated after each conditional branch prediction, and the BTB entry is updated after each branch prediction for call and indirect branches. Moreover, the RSB is updated after each call and return instructions. If the return address is predicted based on the BTB, the BTB entry is also updated after the return instruction. In addition, the BHB is updated after each branch instruction for all the conditional, unconditional, call, return, and indirect branches.

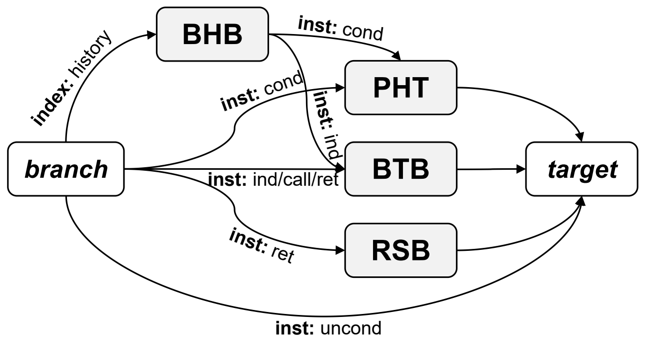


Fig. 1. The workflow for the prediction of different branch instructions.

B. Microarchitectural Attacks against Branch Predictors

Side-channel attacks [44], [51] exploit unintentional information leakage from computing devices or implementations to infer sensitive information. Traditional side-channel attacks mainly include electromagnetic side-channel attacks [4] and power side-channel attacks [41]. Microarchitectural side-channel attacks exploit optimization mechanisms introduced in the processor microarchitecture (e.g., out-of-order execution, speculative execution, caching) to infer security-critical information by observing the execution time of different instructions or programs.

TABLE I
SUMMARY OF EXISTING MICROARCHITECTURAL SIDE-CHANNEL ATTACKS AGAINST BRANCH PREDICTORS

Type	Unit	Attack	Steps
TSCA	PHT	Template Attack [8], [9]	$A \rightarrow V \rightarrow A$
		BranchScope [27]	$A \rightarrow V \rightarrow A$
		Bluethunder [38]	$A \rightarrow V \rightarrow A$
	BTB	Predicting Keys [1]–[3]	$A \rightarrow V \rightarrow A$
		JumpOverASLR [25]	$A \rightarrow V \rightarrow A$
	RSB	Predicting Keys [13]	$A \rightarrow V \rightarrow A$
TEA	PHT	Spectre V1 [40], [55]	$V \rightarrow V \rightarrow A$
		NetSpectre [49]	$V \rightarrow V \rightarrow A$
		SpectreRewind [29]	$V \rightarrow V \rightarrow A$
		BranchSpectre [18]	$V \rightarrow V \rightarrow A$
		Spec Interference [7]	$V \rightarrow V \rightarrow A$
		SpecHammer [54]	$V \rightarrow V \rightarrow A$
		PACMAN [48]	$V \rightarrow V \rightarrow A$
	BTB	Spectre V2 [40]	$A/V \rightarrow V \rightarrow A$
		SMoTherSpectre [11]	$A/V \rightarrow V \rightarrow A$
		SGXPectre [17]	$A/V \rightarrow V \rightarrow A$
		SpecROP [10]	$A/V \rightarrow V \rightarrow A$
		BHI [5]	$A \rightarrow V \rightarrow A$
		RetBleed [58]	$A \rightarrow V \rightarrow A$
	RSB	Spectre V5 [42], [45]	$A \rightarrow V \rightarrow A$
CCA	PHT	Residual State [24], [26]	$A \rightarrow V \rightarrow A$
		Contention [26], [37]	$A \rightarrow V \rightarrow A$

- Attack Types: TSCA represents Timing Side-Channel Attack, TEA represents Transient Execution Attack, and CCA represents Covert-Channel Attack.
- Attack Steps: A represents the $Attacker$, and V represents the $Victim$. Especially, we use V to represent the sender of the covert channel and A to represent the receiver.

According to the research perspective on branch predictor vulnerabilities, microarchitectural side-channel attacks against branch predictors can be classified into three types:

- Timing Side-Channel Attacks (TSCA) [44], [51]: this is a passive attack in which the attacker steals confidential data by observing the side timing information.
- Transient Execution Attacks (TEA) [14], [59]: this is an active attack in which the attacker steals sensitive data by unauthorized access.
- Covert-Channel Attacks (CCA) [51], [59]: this is an active attack in which the attacker bypasses privilege boundaries and transmits security-critical data by establishing a communication channel.

Table I summarizes the existing microarchitectural side-channel attacks on branch predictors, their corresponding attack types, and their three-step attack procedures.

C. Related Work

Symbolic execution is a well-known technique for verifying the security of programs and is also widely used in microarchitecture side-channel attacks. Wang et al. [57] and Guo et al. [33] validate the security of programs against speculative execution attacks by implementing KLEE-based symbolic execution prototype tools, namely KLEESpectre and SpecuSym, respectively. Moreover, Guarnieri et al. [31] introduce a semantic notion of "speculative non-interference" and develop Spectector to verify the security of codes against Spectre V1 attacks. Fabian et al. [28] then extend Spectector to support Spectre V5 detection by implementing novel speculative semantics for return instructions. In addition, Daniel et al. [19] propose a new relational symbolic execution framework for security evaluation against speculative execution attacks.

Several works model the microarchitecture of modern processors for security analysis from the hardware perspective. Guanciale et al. [30] propose a comprehensive formal model called Inspectre to model the behavior of out-of-order execution and speculative execution and then use this model to derive three new speculative vulnerabilities. Then, He et al. [34] define the concept of "security dependency" and propose an attack graph model for reasoning about speculative attacks. Moreover, Hu et al. [36] present a six-step modeling approach to analyze 20 hardware defenses against speculative execution attacks systematically. In addition, Yang et al. [61] introduce a model-checking framework called Pensieve to verify the security of existing hardware-based invisible countermeasures against speculative execution attacks. However, these models do not cover all timing side-channel attacks and covert-channel attacks exploiting branch predictors, nor do they entirely derive all potential speculative execution attacks.

Deng et al. [20]–[22] propose a three-step modeling approach to analyze security vulnerabilities in caches and TLBs. By modeling possible memory operations of the attacker and victim, they derive 88 timing-based attacks against caches (including 32 new attack patterns) [20], [22] and 24 timing-based attacks against TLBs (including 16 new attack patterns) [21]. Moreover, Deutsch et al. [23] introduce a comprehensive model called Metior to evaluate existing hardware-based mitigations against cache timing attacks and discover unexplored microarchitectural leakages. Although these works identify new timing-based attacks on caches or TLBs, they do not consider branch predictors with numerous vulnerabilities.

III. Modeling Branch Predictor States and Corresponding Operations

In this section, we abstractly model all possible states and corresponding operations of branch predictors according to the working mechanism of each unit. Then, we propose a three-step attack model for characterizing existing microarchitectural side-channel attacks against branch predictors and further derivation of all potential security vulnerabilities.

A. Threat Model and Assumptions

The microarchitecture side-channel attack against branch predictors involves an attacker and a victim. In our threat model, we use A and V to denote the attacker and victim programs, both of which can execute branch instructions and change the state of the branch predictor. The attacker and the victim can run on the same logical core or two different logical cores on the same processor. We also assume that the attacker and the victim can be at the same privilege level (e.g., user mode) or different privilege levels (e.g., user mode and kernel mode). In addition, the attacker can invoke the victim's branch instructions to mislead the branch predictor, i.e., the malicious operation is executed by V but controlled by A, or execute branch instructions in its own address space to poison the branch predictor, i.e., the malicious operation is performed and controlled by A.

Our threat model also makes other assumptions about the abilities of the attacker and the victim:

- First, we assume that the victim executes some security-critical branch instructions b, where the branch direction depends on the secret the attacker wants to learn. For example, when performing the RSA encryption function of the OpenSSL cryptographic library, where the value of the key bit (0 or 1) determines the direction of the branch. The execution time of such branch instructions is affected by branch prediction-related operations, which may reveal information about the encryption key.
- Second, we suppose that the attacker knows the implementation of the victim's program, such as the encryption algorithm of the victim's program and the location of the branches related to the secret, but the attacker does not know the specific confidential data.
- In addition, we also assume that the attacker understands the state machine logic of the various constituent units of the branch predictor. Although the attacker does not have access to the internal state of the branch predictor, it can measure the execution time of its or the victim's operations and determine whether the execution time is $fast$ or $slow$ to infer the state of the branch predictor.

We consider three types of branch predictor vulnerabilities in our threat model, including timing side-channel attack (TSCA), transient execution attack (TEA), and covert-channel attack (CCA). The difference between the transient execution attack and the covert-channel attack is that the former is related to the unauthorized transient execution of the branch predictor and may use other covert channels to disclose secrets, i.e., the $fast$ and $slow$ in such attacks are not directly related to the branch predictor. In contrast, the latter uses the branch predictor as a covert channel to leak information, i.e., the $fast$ and $slow$ in such attacks are directly related to the correct or incorrect prediction of the branch predictor.

B. Modeling States of Branch Predictors

Since the state of each entry is independent of other entries and the update logic is the same for each entry, we abstractly model each type of branch predictor unit by considering only a single entry for each branch predictor unit. Then, as a single entry is the minimum component of each branch predictor unit, a significant advantage of this approach is that it reduces the complexity of the subsequent analysis. In addition, since the BHB is only mixed with the PC value of the branch instruction and does not affect the state of other branch predictor units, we do not model the BHB separately but as an indexing approach integrated with the PHT and BTB.

Assuming that E_{val} denotes the branch predictor entry E is indexed by the value val, three possible ways of indexing in the branch predictor may affect the state of E:

- pc denotes that the attacker exploits the same PC value (the part used for indexing, not the entire address) as the target branch val to pollute entry E. This type of indexing exists in both the PHT and the BTB, so this may lead to PHT or BTB entries being polluted.
- his denotes that entry E is indexed by a mixed value of the PC value and the BHB value, and this operation may cause the BHB value to be poisoned, which may cause the entry of PHT or BTB to be polluted.
- $alias$ denotes that a different index value indexes entry E due to the hash collision in the BTB or the stack collision in the RSB, and the index value is different from val, which may cause the entry of BTB or RSB to be polluted.

Due to the limited capacity of the BTB or RSB, the target entry E may also be evicted from the branch predictor unit, and we use inv to denote this situation. Table II lists all 19 possible states of each entry in the branch predictor unit.

For a given branch instruction, every PHT entry has two different states: $valid$ and $mispredict$, where $valid$ denotes that the entry is valid in the PHT and the prediction is correct; $mispredict$ indicates that the entry exists in the PHT, but the prediction is incorrect. We do not choose $T(taken)$ and $NT(not\ taken)$ as the states because the attacker aims to leverage timing penalties for misprediction rather than a specific T/NT state. Our modeling avoids two symmetric cases (T/T and NT/NT categorized as $valid$, T/NT and NT/T classified as $mispredict$) to reduce the complexity of the subsequent analysis. Moreover, each BTB entry has some bits to store the branch type of the branch instruction, such as call, return, or jump, making it infeasible for each branch to pollute the branch target address of other instruction types even if there is a hash collision. Unlike the PHT entry state, for a given branch instruction, the BTB entry has three different

TABLE II
ALL POSSIBLE 19 STATES OF AN ENTRY IN THE BRANCH PREDICTORS

Unit	Branch	Entry	State	Description
PHT	cond	E_{val}	$valid$	The prediction is correct.
		E_{pc}	$mispredict$	The prediction is incorrect.
		E_{his}	$mispredict$	The prediction is incorrect.
BTB	ind	E_{inv}	$invalid$	The entry is not in the BTB.
		E_{val}	$valid$	The prediction is correct.
		E_{pc}	$mispredict$	The prediction is incorrect.
		E_{his}	$mispredict$	The prediction is incorrect.
		E_{alias}	$mispredict$	The prediction is incorrect.
	call	E_{inv}	$invalid$	The entry is not in the BTB.
		E_{val}	$valid$	The prediction is correct.
		E_{pc}	$mispredict$	The prediction is incorrect.
		E_{alias}	$mispredict$	The prediction is incorrect.
	ret	E_{inv}	$invalid$	The entry is not in the BTB.
		E_{val}	$valid$	The prediction is correct.
		E_{pc}	$mispredict$	The prediction is incorrect.
		E_{alias}	$mispredict$	The prediction is incorrect.
RSB	ret	E_{inv}	$invalid$	The entry is not in the RSB.
		E_{val}	$valid$	The prediction is correct.
		E_{alias}	$mispredict$	The prediction is incorrect.

- E_{val} denotes entry E is indexed by the valid value val.
- E_{inv} denotes entry E is evicted from the BTB/RSB.
- $E_{pc}/E_{his}/E_{alias}$ denotes entry E is poisoned by $pc/his/alias$.

states: $invalid$, $valid$, and $mispredict$, where $invalid$ denotes that the entry does not exist in the BTB and cannot be used for prediction; $valid$ means that the entry is valid in the BTB and the prediction is correct; and $mispredict$ indicates that the entry exists in the BTB, but the prediction is incorrect. In addition, RSB entries are only pushed into the RSB when the call instruction is executed, and there are also three different states for a given branch instruction: $invalid$, $valid$, and $mispredict$. The $invalid$ state suggests that the RSB entry does not exist in the RSB; the $valid$ state means that the RSB entry is valid, and the return address is correct; and the $mispredict$ state denotes that the RSB entry exists, but the return address is incorrect.

C. Three-Step Attack Model

After modeling the state of the branch predictor and the operation of the attacker and the victim, we propose a three-step attack model against the branch predictor and use $\{Step\ 1 \rightarrow Step\ 2 \rightarrow Step\ 3\}$ to denote the three-step combination of operations. In the first step, a branch operation is performed to set an entry of any branch predictor (PHT/BTB/RSB) to a known initial state (e.g., to set the saturation counter of an entry of the PHT to $taken$ or to flush a block from the BTB). Then, in the second step, another branch operation is performed to change the state of the corresponding entry of the branch predictor unit from the initial state. Moreover, in the third step, a timing observation operation is performed through the last branch operation or other covert channels. The attacker can infer the secret data from the timing observation operation.

Then, we prove that all the microarchitectural side-channel attacks against the branch predictor can be modeled as the three-step attack model:

- First, the attacker sets the state of the branch predictor to a known initial state to ensure that the attack can be executed correctly. In addition, a branch predictor entry E can only have one state at a time, and it only takes a branch sequence b (including $\{b_1, b_2, \cdots, b_n\}$) on that entry to change the state of E to the target state. Therefore, we can reduce these operations to an abstracted operation b, the only operation needed to set the initial state of the branch predictor in our model.
- Second, the state of the branch predictor entry E should be changed to a different state by multiple branch operations $\{b'_1, b'_2, \cdots, b'_n\}$ to leak secret information. Similarly, we can reduce these operations to an abstracted operation b' and model this operation as the second step of the attack.
- Finally, another essential step in the attack process is that the attacker infers the state of the branch predictor entry E or the leakage of secret information due to incorrect prediction. For the former, the attacker can observe the state of E through one abstracted branch operation b''. For the latter, the attacker can measure the leakage of secret information through other covert channels, and we model this operation as cc. Therefore, we can model the third step of the attack as b'' or cc.
- All existing microarchitectural side-channel attacks on branch predictors can be modeled as the three-step attack model $\{b \rightarrow b' \rightarrow b''/cc\}$.

Afterward, we model all 53 possible operations of attackers and victims that could affect the state of the branch predictor in the three-step attack model, as listed in Table III. We use A and V to denote the attacker's and the victim's operations, respectively. Moreover, we use $A_\star$ or $V_\star$ to denote that the attacker or the victim does not operate on the target branch predictor entry in this step. We then use A_{cc} to denote the attacker's operation to observe the state of the covert channel other than the branch predictors, which exists only in the third step of the transient execution attack.

Then, assuming that the target entry corresponds to the security-critical data (e.g., the encryption key), we use A_{val} or V_{val} to denote the operation to set this target entry to $valid$ (i.e., to access the security-critical data). We also use A_{inv}/V_{inv} to denote the operation to set the target entry to $invalid$ by evicting the target entry from the branch predictor. In addition, we use $A_{pc}/V_{pc}/A_{his}/V_{his}/A_{alias}/V_{alias}$ to denote the operation to change the state of the target entry to $mispredict$ by the branch instruction with index $pc/his/alias$.

In both timing side-channel attacks and covert-channel attacks, each step in the model represents the operation that changes the state of a branch predictor unit entry. For example, in the BranchScope attack [27] ($\{A_{pc} \rightarrow V_{val} \rightarrow A_{pc}\}$), the attacker first trains the PHT branch predictor and sets the state to $mispredict$ (e.g., $taken$) in $Step$ 1. Then in $Step$ 2, the victim executes several branch instructions to change the state of the corresponding PHT entry to $valid$ (e.g., from $taken$ to $not\ taken$). Finally, in $Step$ 3, the attacker observes the timing difference through the execution time of the same branch instruction in $Step$ 1 to learn the victim's execution pattern.

TABLE III
ALL 53 POSSIBLE OPERATIONS IN THE THREE-STEP ATTACK MODEL

Unit	Branch	Operation	State	Description
PHT	cond	A_{cc}	not changed	covert channel
		$A_{\star}$ or $V_{\star}$	not changed	no operation
		A_{val} or V_{val}	$valid$	make the entry $valid$
		A_{pc} or V_{pc}	$mispredict$	mispredict the entry
		A_{his} or V_{his}	$mispredict$	mispredict the entry
BTB	ind	A_{cc}	not changed	covert channel
		$A_{\star}$ or $V_{\star}$	not changed	no operation
		A_{inv} or V_{inv}	$invalid$	evict the entry
		A_{val} or V_{val}	$valid$	make the entry $valid$
		A_{pc} or V_{pc}	$mispredict$	mispredict the entry
		A_{his} or V_{his}	$mispredict$	mispredict the entry
		A_{alias} or V_{alias}	$mispredict$	mispredict the entry
	call	A_{cc}	not changed	covert channel
		$A_{\star}$ or $V_{\star}$	not changed	no operation
		A_{inv} or V_{inv}	$invalid$	evict the entry
		A_{val} or V_{val}	$valid$	make the entry $valid$
		A_{pc} or V_{pc}	$mispredict$	mispredict the entry
		A_{alias} or V_{alias}	$mispredict$	mispredict the entry
	ret	A_{cc}	not changed	covert channel
		$A_{\star}$ or $V_{\star}$	not changed	no operation
		A_{inv} or V_{inv}	$invalid$	evict the entry
		A_{val} or V_{val}	$valid$	make the entry $valid$
		A_{pc} or V_{pc}	$mispredict$	mispredict the entry
		A_{alias} or V_{alias}	$mispredict$	mispredict the entry
RSB	ret	A_{cc}	not changed	covert channel
		$A_{\star}$ or $V_{\star}$	not changed	no operation
		A_{inv} or V_{inv}	$invalid$	evict the entry
		A_{val} or V_{val}	$valid$	make the entry $valid$
		A_{alias} or V_{alias}	$mispredict$	mispredict the entry

In transient execution attacks, the first two steps of the model are still the operation that changes the state of a branch predictor unit entry. However, the third step of the model is other operations that observe the state of other covert channels since the covert channel is not branch prediction related. For example, in the Spectre attack [40] ($\{V_{pc} \rightarrow V_{val} \rightarrow A_{cc}\}$), the attacker first triggers the victim into training the PHT branch predictor with in-bounds values and sets the state to $mispredict$ (the prediction is wrong if the input is out-of-bounds) in $Step$ 1. Then, in $Step$ 2, the attacker sends the victim an out-of-bounds value and triggers unauthorized access to sensitive data. Finally, in $Step$ 3, the attacker observes the timing difference through a microarchitectural covert channel (e.g., caches, TLBs) to infer the sensitive data.

As mentioned earlier, since the entries of each branch predictor unit are updated according to the same state machine logic, it is sufficient to consider only a single entry since it is the minimum component of the branch predictor. Although different branch predictor implementations involve different address-to-entry mapping functions, this does not affect the three-step model for a single entry. Furthermore, these differences in mapping functions may make it more difficult for attackers and victims to access the same entry in an actual attack, but the attacker can still launch an attack in this case.

IV. Derivation of All Security Vulnerabilities against Branch Predictors

In this section, we construct a symbolic execution-based security analysis framework based on the proposed three-step attack model and the operation types of the attacker and the victim. We also introduce vulnerability specifications for deriving valid attack patterns and reduction rules for reducing the number of redundant vulnerabilities. Finally, we derive and categorize all potential branch predictor vulnerabilities.

A. Three-Step Branch Predictor Simulator

Three-Step Simulator. We develop a three-step branch predictor simulator, where each step represents a possible operation of the attacker or the victim, and analyze valid attack patterns based on the attacker's observations (branch timing or covert channel) in the third step. The three-step branch predictor simulator is implemented in Rust, where the input of the simulator is all 53 possible operations of the attacker and the victim in the three-step attack model, and the output is whether the three-step attack pattern is valid or not for each three-step combination. According to our previous modeling of branch predictors, there are $9 * 9 * 9 = 729$ different three-step combinations for the PHT, $13 * 13 * 13 + 11 * 11 * 11 + 11 * 11 * 11 = 4859$ three-step combinations for BTB, and $9 * 9 * 9 = 729$ three-step combinations for RSB. Thus, there are 6317 different three-step patterns under our modeling criteria.

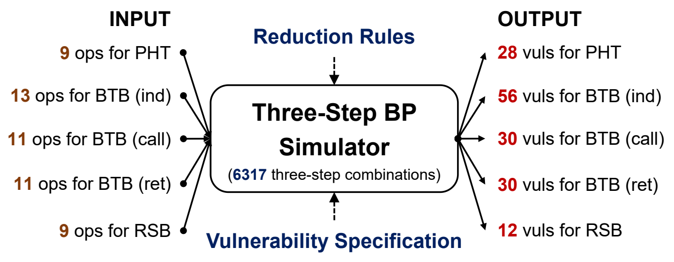


Fig. 2. The progress of finding vulnerabilities with the three-step branch predictor simulator.

State of Branch Predictor Entry. In the branch predictor simulator, we define a current state for the target branch predictor entry and assume that the initial state of the entry is $unknown$. Then, while each step of the three-step combination is executed in turn, the state of the target entry is updated according to the operation type based on Table III. For example, if the attacker performs A_{pc}/V_{pc}, the state of the target entry is updated to $mispredict$, and if the attacker performs A_{val}/V_{val}, the state of the target entry is updated to $valid$, and if the attacker performs $A_{cc}/A_{\star}$, the state of the target entry is not updated.

Path Extension. For the security-critical branch operation A_{val}/V_{val}, we extend it to two different states, one to update the branch predictor state and one to not update the branch predictor state. In addition, if the first two steps are the security-critical branch operation A_{val}/V_{val}, we only extend the latter operation with two different states. Moreover, since we consider the third step to be the timing observation step, we do not extend the states in this step even if the operation is A_{val}/V_{val}. Therefore, for the three-step combinations that contain A_{val}/V_{val}, there are two different three-step execution paths, and we use the attacker's ability to observe the timing difference between these two paths as the judging standard for whether it is a valid attack pattern.

For both timing side-channel attacks and covert-channel attacks, if the branch prediction in the third step is correct (e.g., performing V_{val} when the entry is *valid*, performing A_{pc} when the entry is *mispredict*), the three-step path is marked as $fast$. If the branch prediction in the third step is incorrect or the target entry state is *invalid*, we characterize the three-step path as *slow*. And if the target entry state is *unknown* in the third step, the three-step path is marked as *unknown*.

In addition, for transient execution attacks, we assume the covert channel is always the $fast$ path after encoding the secret data. Thus, if the state of the target entry is *mispredict* when performing the security-critical branch operation V_{val}, the three-step path is marked as $fast$. Then, if the state of the target entry is *valid* or *invalid* when performing V_{val}, the three-step path is marked as *slow*. Moreover, if the state of the target entry is *unknown* when performing V_{val}, the three-step path is marked as *unknown*.

Reduction Rules. We also develop some reduction rules to reduce the analytical complexity of the three-step simulator and minimize the output list of valid three-step combinations. A three-step combination will be reduced if it satisfies one of the following rules:

- First, since the attacker cannot access the security-critical branch in our threat model, we reduce all three-step combinations containing A_{val}. And since the attacker observes the timing difference between two execution paths to identify the security vulnerabilities, and only the security-critical operation V_{val} can extend two different execution paths, we also reduce all three-step combinations that do not contain V_{val}.
- Second, if two successive operations are of the same type and neither is a security-critical operation V_{val}, these two steps can be reduced to the latter operation. For example, if the three-step combination is $\{A_{pc} \rightarrow A_{pc} \rightarrow V_{val}\}$, then this pattern can be reduced to $\{A_\star \rightarrow A_{pc} \rightarrow V_{val}\}$.
- Moreover, if A_{cc} occurs in the first or second step, this step can be reduced to $A_\star$ because the transient execution cannot occur with only one branch operation. For example, if the three-step combination is $\{A_{cc} \rightarrow A_{pc} \rightarrow V_{val}\}$, then this combination can be reduced to $\{A_\star \rightarrow A_{pc} \rightarrow V_{val}\}$.
- Furthermore, if $A_\star$ or $V_\star$ occurs in the second or third step, which means that the attacker or the victim does not perform any operation, then we can swap it to the previous step of the three-step combination by the swap law of adjacent operations. Then, since $A_\star$ or $V_\star$ does not perform operations related to the target entry and update the prediction state of the entry, we can reduce both operations to $\star$ uniformly. For example, if the three-step combination is $\{A_{pc} \rightarrow A_\star/V_\star \rightarrow V_{val}\}$, then this combination can be reduced to $\{\star \rightarrow A_{pc} \rightarrow V_{val}\}$.

Then, we apply these four reduction rules in our three-step branch predictor simulator and obtain the list of all valid three-step attack patterns.

Vulnerability Specification. Finally, we perform vulnerability identification for all three-step combinations where two execution paths exist, with the following specification:

- $Vulnerable$: If the attacker can successfully distinguish the timing of two paths when performing the timing observation operation in the third step, i.e., one path is marked as $fast$ and the other path is marked as *slow*, then we consider this three-step combination vulnerable.
- $Not\ Vulnerable$: If the attacker fails to distinguish the timing of the two paths during the timing observation operation in the third step, i.e., both paths are marked as $fast$, *slow*, or *unknown*, we consider that this three-step combination has no vulnerability at this time. If one path is marked as *unknown* while the other path is marked as $fast$ or *slow*, i.e., the attacker may not be able to observe the timing difference in this three-step combination, we consider this pattern to be secure as well.

According to this specification, we can determine whether a three-step combination is vulnerable. And we only consider the three-step combinations in which the attacker can distinguish the timing differences created by the security-critical branch instruction as vulnerable.

Extensibility. Our three-step branch predictor simulator can also be extended to analyze the security vulnerabilities of new branch predictor mechanisms. For example, if a new design introduces new branch predictor states and corresponding operations, we can model these states and operations according to our modeling rules. Then, we can embed these operations into the three-step simulator and modify the state update constraints, path extension logic, and reduction rules. Finally, we can derive security vulnerabilities for the new design through the extended three-step branch predictor simulator.

B. Derivation and Categorization of Vulnerabilities

Based on our reduction rules and the vulnerability specification, we derive a total of 156 different security vulnerabilities using the three-step branch predictor simulator, including 28 conditional branch prediction vulnerabilities for the PHT, 56 indirect branch prediction vulnerabilities for the BTB, 30 call instruction prediction vulnerabilities for the BTB, 30 return address prediction vulnerabilities for the BTB, and 12 return address prediction vulnerabilities for the RSB.

Table IV lists all 156 three-step attack patterns against branch predictors. Among them, only 67 attack patterns corresponding to 9 types of attacks (other types of attacks listed in Table I can be reduced to these types) are those already found in previous studies, including four attack types against the PHT (BranchScope [27], Bluethunder [38], Spectre V1 [40], and BranchSpectre [18]), three attack types against the BTB (Predicting Keys [1]–[3], Spectre V2 [40], and BHI [5]), and two attack types against the RSB (Predicting Keys [13] and Spectre V5 [42], [45]). The remaining 89 attack patterns are all newly discovered in our study, including 16 new attack patterns against conditional branches of the PHT, 36 new attack patterns against indirect branches of the BTB, 15 new

TABLE IV
ALL 156 DERIVED SECURITY VULNERABILITIES AGAINST BRANCH PREDICTORS

Unit	Step1	Step2	Step3	Category	Type	Attack	Step1	Step2	Step3	Category	Type	Attack
PHT	V_{val}	A_{pc}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{pc}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	V_{val}	A_{his}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{his}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	A_{pc}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	A_{pc}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	(1)
	A_{pc}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	(1)	A_{pc}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	(1)
	A_{pc}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	(1)	V_{pc}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{pc}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**	V_{pc}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**
	V_{pc}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	**new**	V_{pc}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	**new**
	A_{his}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(2)	A_{his}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	(2)
	A_{his}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	(2)	A_{his}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	(2)
	A_{his}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	(2)	V_{his}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{his}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**	V_{his}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**
	V_{his}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	**new**	V_{his}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	**new**
	A_{pc}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	**new**	V_{pc}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(3)
	A_{his}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	**new**	V_{his}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(4)
BTB (ind)	A_{inv}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	V_{inv}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{val}	A_{pc}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{pc}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	V_{val}	A_{his}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{his}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	V_{val}	A_{alias}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{alias}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	A_{pc}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	A_{pc}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	(1)
	A_{pc}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	(1)	A_{pc}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	(1)
	A_{pc}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	(1)	A_{pc}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	(1)
	A_{pc}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	(1)	V_{pc}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{pc}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**	V_{pc}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**
	V_{pc}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	**new**	V_{pc}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	**new**
	V_{pc}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	**new**	V_{pc}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	**new**
	A_{his}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**	A_{his}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**
	A_{his}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**	A_{his}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	**new**
	A_{his}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	**new**	A_{his}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	**new**
	A_{his}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	**new**	V_{his}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{his}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**	V_{his}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**
	V_{his}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	**new**	V_{his}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	**new**
	V_{his}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	**new**	V_{his}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	**new**
	A_{alias}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	A_{alias}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	(1)
	A_{alias}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	(1)	A_{alias}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	(1)
	A_{alias}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	(1)	A_{alias}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	(1)
	A_{alias}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	(1)	V_{alias}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{alias}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**	V_{alias}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**
	V_{alias}	V_{val}	$A_{his}\ (slow)$	EM	TSCA/CCA	**new**	V_{alias}	V_{val}	$V_{his}\ (slow)$	IM	TSCA/CCA	**new**
	V_{alias}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	**new**	V_{alias}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	**new**
	A_{pc}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)	V_{pc}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)
	A_{his}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(3)	V_{his}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	**new**
	A_{alias}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)	V_{alias}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)
BTB (call/ret)	A_{inv}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	V_{inv}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{val}	A_{pc}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{pc}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	V_{val}	A_{alias}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{alias}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	A_{pc}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	A_{pc}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	(1)
	A_{pc}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	(1)	A_{pc}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	(1)
	A_{pc}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	(1)	V_{pc}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{pc}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**	V_{pc}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**
	V_{pc}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	**new**	V_{pc}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	**new**
	A_{alias}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	A_{alias}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	(1)
	A_{alias}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	(1)	A_{alias}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	(1)
	A_{alias}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	(1)	V_{alias}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{alias}	V_{val}	$A_{pc}\ (slow)$	EM	TSCA/CCA	**new**	V_{alias}	V_{val}	$V_{pc}\ (slow)$	IM	TSCA/CCA	**new**
	V_{alias}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	**new**	V_{alias}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	**new**
	A_{pc}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)	V_{pc}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)
	A_{alias}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)	V_{alias}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)
RSB	A_{inv}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	V_{inv}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{val}	A_{alias}	$V_{val}\ (slow)$	EM	TSCA/CCA	**new**	V_{val}	V_{alias}	$V_{val}\ (slow)$	IM	TSCA/CCA	**new**
	A_{alias}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	(1)	A_{alias}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	(1)
	A_{alias}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	(1)	V_{alias}	V_{val}	$V_{val}\ (fast)$	IH	TSCA/CCA	**new**
	V_{alias}	V_{val}	$A_{alias}\ (slow)$	EM	TSCA/CCA	**new**	V_{alias}	V_{val}	$V_{alias}\ (slow)$	IM	TSCA/CCA	**new**
	A_{alias}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	(2)	V_{alias}	V_{val}	$A_{cc}\ (fast)$	EH	TEA	**new**

- PHT: (1) BranchScope [27]; (2) Bluethunder [38]; (3) Spectre V1 [40]; (4) BranchSpectre [18].
- BTB: (1) Predicting Keys [1]–[3]; (2) Spectre V2 [40]; (3) BHI [5].
- RSB: (1) Predicting Keys [13]; (2) Spectre V5 [42], [45].

attack patterns against call instructions of the BTB, 15 new attack patterns against return addresses of the BTB, and 7 new attack patterns against return addresses of the RSB.

We then further categorize the derived three-step attack patterns based on whether the timing observed by the attacker in the observation phase is $fast$ or $slow$ and whether the attacker or the victim performs the operations associated with the security-critical branches and observations. If the attacker observes the $fast$ path for a path that accesses the security-critical branch, we consider this a hit-based attack pattern (H) and, conversely, a miss-based attack pattern (M). In addition, if the second and third steps of the three-step attack pattern are both performed by the victim, i.e., the operations of this attack on the security-critical branch do not involve the attacker's operations on the branch predictor, then we consider it an internal interference attack (I). Otherwise, if both the attacker's and the victim's operations are present in the second and third steps, we categorize these attack patterns as external interference attacks (E). Based on these two dimensions, we categorize the derived attack patterns into 28 hit-based internal interference (IH) attacks, 20 hit-based external interference (EH) attacks, 54 miss-based internal interference (IM) attacks, and 54 miss-based external interference (EM) attacks.

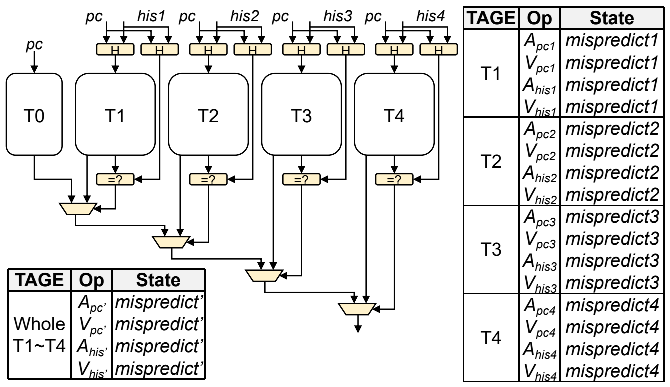
In internal interference attacks, the attacker infers sensitive information by observing one or more branches in the victim's code, meaning that either the attacker can trigger the victim to execute the to-be-observed branch or the victim itself can perform these conflicting branches. Such attacks are more feasible if the attacker can observe the victim's overall execution timing or the timing of secret branches. External interference attacks require the attacker to construct branches that conflict with the victim's security-critical branches (except for speculative attacks) and to be able to observe the timing of the conflict branches, secret branches, or covert channels, making this type of attack somewhat more common.

Afterward, we further analyze the characteristics of the derived attacks from the perspective of different poisoning operations. First, pc-based mistraining operations(A_{pc}/V_{pc}) populate the target entry by repeatedly executing the same branch, which has a higher poisoning success rate and a lower noise. Meanwhile, performing V_{pc} operations in the victim's address space does not require the attacker to make address collisions, so this poisoning approach has a higher chance of being exploited. Second, his-based(A_{his}/V_{his}) and $alias$-based(A_{alias}/V_{alias}) poisoning operations have lower success rates because the former is susceptible to noise from other branches, the latter requires the attacker to construct collision branches, making it more difficult to exploit. Compared to pc-based poisoning, these two types are more practical to be implemented in the attacker's carefully designed code. In addition, inv-based misleading operations(A_{inv}/V_{inv}) are similar to the "Prime" operation in cache attacks, i.e., the attacker fills in the target entries by executing a large number of branches mapped to the same set, which also has a high success rate and low noise. Our subsequent case study in Section IV-D also demonstrates that pc-based misleading is more effective than his-based mistraining.

Additionally, we find that many new attacks poison branch predictors via the victim's code, while most existing attacks rely on the attacker's code to mislead branch predictors. Therefore, it is possible to conduct code searches on existing cryptographic libraries according to our proposed three-step model to investigate whether these new attacks can be launched in real-world scenarios. For those new variants that can still mislead branch predictors with the attacker's code, we will demonstrate the practicality by launching an attack on OpenSSL through a case study in Section IV-E.

C. Case Study I: Extending Modeling Methodology to TAGE

To demonstrate the extensibility of our modeling methodology, we apply our three-step model to the 5-component TAGE branch predictor [50]. Figure 3 shows the modeling of the TAGE branch predictor.



TAGE	Op	State
Whole T1~T4	$A_{pc'}$	*mispredict'*
	$V_{pc'}$	*mispredict'*
	$A_{his'}$	*mispredict'*
	$V_{his'}$	*mispredict'*

TAGE	Op	State
T1	A_{pc1}	*mispredict1*
	V_{pc1}	*mispredict1*
	A_{his1}	*mispredict1*
	V_{his1}	*mispredict1*
T2	A_{pc2}	*mispredict2*
	V_{pc2}	*mispredict2*
	A_{his2}	*mispredict2*
	V_{his2}	*mispredict2*
T3	A_{pc3}	*mispredict3*
	V_{pc3}	*mispredict3*
	A_{his3}	*mispredict3*
	V_{his3}	*mispredict3*
T4	A_{pc4}	*mispredict4*
	V_{pc4}	*mispredict4*
	A_{his4}	*mispredict4*
	V_{his4}	*mispredict4*

Fig. 3. Structure and modeling of the 5-component TAGE branch predictor.

One modeling approach is to model the tagless table as a unit and four tagged tables as an entire unit. For the tagless table, A_{pc}/V_{pc} can be used to change the state of the target entry to $mispredict$ as our simplified PHT. For the abstracted tagged table, $A_{pc'}/V_{pc'}/A_{his'}/V_{his'}$ can be used to pollute the target entry. Since the tagless table and the tagged table are independent of each other, we also extend them with a new $mispredict'$ state to avoid state confusion in the first and third steps using different tables. Finally, we derive 34 attack patterns against TAGE in this modeling approach.

Another modeling approach is to model each tagged table as a unit, adding 16 variants of $A_{pc}/V_{pc}/A_{his}/V_{his}$ operations and 4 variants of $mispredict$ states are added for each tagged table. We then extend the states, operations, and state update constraints of the three-step simulator to support this modeling approach. Compared to abstracting four tagged tables as a whole, this modeling method can capture more accurate state changes in each tagged table, and we derive 106 attack patterns in this modeling approach.

D. Case Study II: Proof-of-Concept for Derived Attacks

To demonstrate the feasibility of our discovered variants, we implement proof-of-concept for a novel V_{pc}-based attack and a novel V_{his}-based attack. For V_{pc}-based attacks, we perform the same branch in the victim's address space and mislead the PHT to $taken$. Then, we trigger the secret-dependent branch V_{val} twice in the victim's address space and observe the timing of the second branch to infer the secret information. For V_{his}-based attacks, we perform the same branch with different outcomes in the victim's address space to activate two-level branch prediction. We then execute several branches with the same source and destination addresses in the victim's address space to poison the branch history. Finally, we trigger the target branch V_{val} four times in the victim's address space and observe the timing to infer the secret information.

TABLE V
EVALUATION OF V_{pc}-BASED AND V_{his}-BASED PHT ATTACK

Variant	Configuration	Resolution (Cycles)	Capacity (Kbps)
V_{pc}-based PHT Attack ($V_{pc} \to V_{val} \to V_{val}$)	Intel Core i5-1135G7 (TigerLake, WSL2)	92 vs 108	865.711
	Intel Core i7-12700 (AlderLake, Arch)	69 vs 83	925.482
V_{his}-based PHT Attack ($V_{his} \to V_{val} \to V_{val}$)	Intel Core i5-1135G7 (TigerLake, WSL2)	91 vs 114	690.745
	Intel Core i7-12700 (AlderLake, Arch)	67 vs 83	734.140

We prove the viability of these two attacks by assessing their channel capacity on two Intel CPUs through the transmission of random "0" and "1" bits repeated 1,000,000 times. As depicted in Table V, the results indicate that both variants can effectively exploit vulnerabilities, enabling the leakage of sensitive information with a substantial bandwidth (865+ Kbps for the V_{pc}-based PHT attack and 690+ Kbps for the V_{his}-based PHT attack).

E. Case Study III: Attacking OpenSSL with Novel Variants

Previous study [11] shows that $EVP\_EncryptUpdate()$ function in $libcrypto$ library of OpenSSL 1.1.1b is vulnerable to branch predictor attacks. We then demonstrate the practicality of a novel timing-based BTB attack variant ($\{V_{val} \to A_{pc} \to V_{val}\}$) to recover the LSB of the first bytes exploiting the same vulnerability.

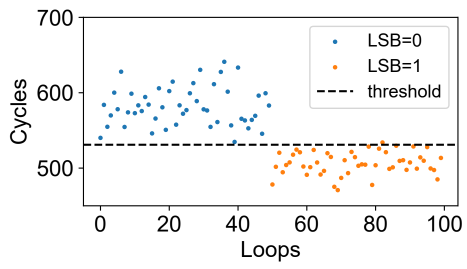


Fig. 4. Recovering LSB in OpenSSL with a novel timing-based BTB attack.

We implement a proof-of-concept attack on Intel Core i7-12700 consisting of three steps. First, the attacker measures the threshold execution timing of V_{val} operation according to branch predictor hit and miss. Then, the attacker conducts the branch target injection with A_{pc} operation to mislead the indirect branch in the $libcrypto$ library, which will result in branch interference in V_{val} (hit for value "1" and miss for value "0"). Finally, the attacker triggers the victim to execute V_{val} and measures the execution timing to infer the target LSB value. We repeat the attack 100 times and the evaluation result shown in Figure 4 indicates that the attacker can recover the correct LSB value with this variant.

V. AUTOMATED SECURITY ANALYSIS OF EXISTING SECURE BRANCH PREDICTORS

In this section, we first apply our symbolic execution-based security analysis framework to modeling 8 existing secure branch predictors [12], [16], [43], [52], [56], [64]–[66] and conducting an automated security analysis of these designs. Then, we also evaluate their effectiveness in mitigating speculative execution attacks on branch predictors by comparing them with four typical hardware-based speculative execution countermeasures.

A. Modeling Existing Secure Branch Predictors

Since all the potential three-step attack patterns we derive do not contain $A_{val}/A_{\star}/V_{\star}$ operations, we only need to consider the remaining 38 possible operations in the following analysis. We also assume that RSB vulnerabilities are protected by RSB refilling mechanism [42], which flushes the RSB entries during the context switch. Then, we model the operations of the attacker and victim in the three-step model for each design.

We analyze all operations of the attacker and victim in the three-step model for each secure branch predictor and reduce the operations that their secure design can prevent with the following reduction rules:

- A_{inv}/V_{inv}: If the attacker cannot evict the target entry from the branch predictor using branch instructions in the attacker/victim's address space, then we can reduce the A_{inv}/V_{inv} operations.
- A_{pc}/V_{pc}: If the attacker cannot pollute the entry with the same index as the victim in the attacker/victim's address space, then we can reduce the A_{pc}/V_{pc} operations.
- A_{his}/V_{his}: If the attacker cannot poison the branch history related to the target entry using branch instructions in the attacker/victim's address space, then we can reduce the A_{his}/V_{his} operations.
- A_{alias}/V_{alias}: If the attacker cannot pollute the entry with the alias index in the attacker/victim's address space, then we can reduce the A_{alias}/V_{alias} operations.

Lock-Based BTB [52]. This technique uses a locking mechanism to protect the secret-related entries of the BTB, and the locked entries cannot be updated and replaced by other processes. Therefore, the attacker cannot perform the $A_{inv}/A_{pc}/A_{his}/A_{alias}$ operation in its own address space to pollute the state of the branch predictor entry associated with the security-critical branch. However, the attacker can still control the victim to change the state of the locked entry and infer the secret information through the timing-based attack.

MI6 [12] and BRB [56]. These two solutions both flush the state of the PHT during context switches, thus eliminating the A_{pc}/A_{his} operation for the PHT in the three-step model. Similarly, this technique allows the attacker to infer the secret information through the timing-based attack by poisoning the PHT state with the victim's operations.

Two-Level Encryption [43] and Noisy-XOR-BP [65]. These two schemes use encryption to protect the indexing and data of the branch predictor entries. The attacker cannot construct the same index in its address space to pollute the PHT entry corresponding to the security-critical branch, nor can it use the same or an alias index to poison the BTB entry corresponding to the security-critical branch. Therefore, these two techniques remove the A_{pc} operation for the PHT and the $A_{pc}/A_{alias}/V_{pc}/V_{alias}$ operation for the BTB in the three-step model. However, the attacker can still poison the branch history associated with the target PHT or BTB entry to interfere with the victim's branch prediction and steal secret information through the timing-based attack.

Probabilistic Saturating Counter [64]. The basic principle of this approach is to make the saturation counter in the PHT entry update with a certain probability after each branch instruction. Although this design imposes a performance penalty on the branch predictor, if the updating probability is set to a very low value, the attacker cannot force the PHT into a deterministic state in this case, i.e., the attacker cannot know what state of the PHT entry is in a finite state machine. Therefore, the $A_{pc}/A_{his}/V_{pc}/V_{his}$ operation in the three-step model for the PHT can all be eliminated.

LS-BP [16]. This approach uses the PID and the entire branch PC value to index the PHT and BTB entries, making it infeasible for the attacker to index the same entry as the victim in its own address space or through the alias index in the victim's address space. Therefore, the A_{pc} operation for the PHT and the $A_{pc}/A_{alias}/V_{pc}/V_{alias}$ operation for the BTB are removed in the three-step model. However, the attacker can still invalidate the BTB entry, trigger the victim to execute the branch instruction and poison the branch histories to interfere with the victim's branch prediction.

HyBP [66]. This hybrid design combines physical isolation and randomization to protect the security-critical entries of the branch predictor, preventing the attacker from constructing an eviction set for the BTB and poisoning the branch target of the victim entry. This approach eliminates the A_{pc} operation for the PHT and the $A_{inv}/A_{pc}/A_{alias}/V_{inv}/V_{pc}/V_{alias}$ operation for the BTB in the three-step combinations. However, the attacker can still pollute the branch history associated with the target PHT or BTB entry to mistrain the victim's branches.

B. Evaluation of Secure Branch Predictors

After modeling 8 existing typical secure branch predictor designs, we evaluate the security of these secure designs using the three-step branch predictor simulator. Table VI shows the effectiveness of different secure branch predictors in protecting against security vulnerabilities for each branch predictor unit and the total number of vulnerabilities that still exist. Among them, the value to the left of "/" indicates the number of vulnerabilities that exist for that branch predictor unit under that secure design, and the value to the right of "/" indicates the total number of vulnerabilities previously derived for that branch predictor unit.

The results show that PSC [64] can be very effective in protecting against PHT security vulnerabilities under ideal circumstances since this mechanism makes it infeasible for an attacker to mistrain the branch predictor deterministically. Then, HyBP [66] can effectively protect against three-step attack combinations against BTB and is also the most effective design for all security vulnerability protection, which can mitigate 123 out of 156 three-step attack combinations.

TABLE VI
SECURE BRANCH PREDICTOR EVALUATION FOR ALL VULNERABILITIES

Defense Strategy	PHT	BTB (ind)	BTB (call)	BTB (ret)	RSB	Total
Lock-Based BTB [52]	28/28	19/56	11/30	11/30	5/12	74/156
MI6 [12]	10/28	56/56	30/30	30/30	5/12	131/156
BRB [56]	10/28	56/56	30/30	30/30	5/12	131/156
Two-Level Encryption [43]	18/28	12/56	2/30	2/30	5/12	39/156
Noisy-XOR-BP [65]	18/28	12/56	2/30	2/30	5/12	39/156
PSC [64]	0/28	56/56	30/30	30/30	5/12	121/156
LS-BP [16]	18/28	12/56	2/30	2/30	5/12	39/156
HyBP [66]	18/28	10/56	0/30	0/30	5/12	33/156

We then also analyze the effectiveness of protection against four categories of attacks, previously classified as hit-based internal interference attacks (IH), hit-based external interference attacks (EH), missing internal interference attacks (IM), and missing external interference attacks (EM). As listed in Table VII, we can see that HyBP [66] can provide the most effective protection against all four attack categories, which can mitigate 21 out of 28 IH attacks, 14 out of 20 EH attacks, 41 out of 54 IM attacks, and 47 out of 54 EM attacks. Then, Two-Level Encryption [43], Noisy-XOR-BP [65], and LS-BP [16] can also provide the best protection against EH attacks, IM attacks, and EM attacks like HyBP [66].

TABLE VII
SECURE BRANCH PREDICTOR EVALUATION FOR IH/EH/IM/EM ATTACKS

Defense Strategy	IH Attack	EH Attack	IM Attack	EM Attack
Lock-Based BTB [52]	16/28	12/20	36/54	10/54
MI6 [12]	24/28	17/20	49/54	41/54
BRB [56]	24/28	17/20	49/54	41/54
Two-Level Encryption [43]	13/28	6/20	13/54	7/54
Noisy-XOR-BP [65]	13/28	6/20	13/54	7/54
PSC [64]	22/28	15/20	43/54	41/54
LS-BP [16]	13/28	6/20	13/54	7/54
HyBP [66]	7/28	6/20	13/54	7/54

Next, we evaluate the coverage of these secure designs against 67 three-step attack patterns corresponding to the attacks found in previous research work and 89 newly derived attack patterns that we derive using the branch predictor simulator. The experimental results listed in Table VIII show that HyBP [66] provides the best protection against known and newly derived attacks in these secure designs, which can mitigate 58 out of 67 known attacks and 65 out of 89 newly derived attacks. Then, Two-Level Encryption [43], Noisy-XOR-BP [65], and LS-BP [16] have better protection coverage for both known and newly derived attacks. In addition, Lock-Based BTB [52] has better protection against known attacks, but this design has significant omissions for newly derived attacks. Furthermore, MI6 [12] and BRB [56] do not adequately protect against known and newly derived attacks.

TABLE VIII
SECURE BRANCH PREDICTOR EVALUATION FOR KNOWN/NEW ATTACKS

Defense Strategy	**PHT (known)**	**BTB (known)**	**RSB (known)**	**PHT (new)**	**BTB (new)**	**RSB (new)**
Lock-Based BTB [52]	12/12	6/50	0/5	16/16	35/66	5/7
MI6 [12]	2/12	50/50	0/5	8/16	66/66	5/7
BRB [56]	2/12	50/50	0/5	8/16	66/66	5/7
Two-Level Encryption [43]	5/12	7/50	0/5	9/16	35/66	5/7
Noisy-XOR-BP [65]	5/12	7/50	0/5	9/16	35/66	5/7
PSC [64]	0/12	50/50	0/5	0/16	66/66	5/7
LS-BP [16]	5/12	7/50	0/5	13/16	9/66	5/7
HyBP [66]	5/12	4/50	0/5	13/16	6/66	5/7

Although these experimental results show that the existing secure branch predictor designs are not able to resist all microarchitecture side-channel attacks, even the best-performing HyBP [66] can only shield about 79% of the attack patterns, while the worst-performing MI6 [12] and BRB [56] can only cover about 16% of the attack patterns, we can also observe from the evaluation results that various branch predictor designs have better security coverage for previously unknown attack patterns. Therefore, the secure branch predictor design has some positive significance for enhancing the security of branch predictors and computer systems.

C. Comparison with other Hardware-Based Defenses

Apart from secure branch predictor designs, researchers have also proposed many hardware defense strategies against speculative execution attacks, and we then compare the performance of secure branch predictors with these approaches against speculative execution attacks in the following. According to the classification of defense strategies proposed by Hu et al. [36], we select four representative hardware-based defenses for our evaluation that introduce a relatively low-performance overhead in the existing defenses.

DAWG (No Setup) [39]. This method partitions the cache lines into different domains and prevents the attacker from accessing the cache lines in the victim's domain, which can prevent the attacker from performing the A_{cc} operation on the cache in the three-step model. However, the attacker can still perform the A_{cc} operation through other covert channels or when the attacker and the victim are in the same domain.

CSF-LFENCE (No Access Without Authorization) [53]. This hardware-based defense prevents the transient execution after a security-critical branch by inserting a fence instruction between that branch and the secret-dependent memory access, thus eliminating the transient execution of the V_{val} operation in the second step for the PHT in the three-step model.

Speculative Taint Tracking (No Use Without Authorization) [62]. This approach leverages taint-tracking to track sensitive instructions and information flow in the program, then blocks the execution of the security-critical branch until the authorization finishes. Therefore, this method can eliminate the V_{val} operation for the PHT in the three-step model.

InvisiSpec (No Send Without Authorization) [60]. This hardware-based mitigation prevents the attacker from performing the A_{cc} operation on the cache in the three-step model by extending a speculative buffer to store speculatively accessed data and not updating the cache state until the speculation is validated. However, the attacker can still perform the A_{cc} operation through other covert channels.

Then, we evaluate the security vulnerability coverage of these four hardware-based defenses against speculative execution attacks, as shown in Table IX. The result demonstrates that these hardware-based defenses can only mitigate a limited number of speculative execution attacks or only mitigate specific cache covert channels. In contrast, the secure branch predictor designs can mitigate more speculative execution attacks, such as Two-Level Encryption [43], Noisy-XOR-BP [65], LS-BP [16], and HyBP [66]. Therefore, secure branch predictor designs are an effective and viable solution to mitigate the security vulnerabilities against branch predictors compared with hardware-based defenses.

TABLE IX
EVALUATION OF SECURE BRANCH PREDICTORS AND HARDWARE-BASED COUNTERMEASURES AGAINST SPECULATIVE EXECUTION ATTACKS

Defense Strategy	**Defense Type**	**TEA (cache)**	**TEA (other)**
Lock-Based BTB [52]	Secure BP	12/20	12/20
MI6 [12]	Secure BP	17/20	17/20
BRB [56]	Secure BP	17/20	17/20
Two-Level Encryption [43]	Secure BP	6/20	6/20
Noisy-XOR-BP [65]	Secure BP	6/20	6/20
PSC [64]	Secure BP	15/20	15/20
LS-BP [16]	Secure BP	6/20	6/20
HyBP [66]	Secure BP	6/20	6/20
DAWG [39]	No Setup	17/20	19/20
CSF-LFENCE [53]	No Access	15/20	15/20
STT [62]	No Use	15/20	15/20
InvisiSpec [60]	No Send	15/20	19/20

VI. CONCLUSION

This paper proposes a comprehensive and automated security analysis framework for verifying security guarantees of branch predictors against microarchitectural attacks at the design stage. We first present a three-step attack model targeting branch predictors in modern processors by abstractly modeling 19 branch predictor states and 53 operations that could affect these states. Based on this attack model, we develop a symbolic execution-based framework and derive 156 valid attack patterns against branch predictors, including 89 never identified in previous work. Finally, we use this framework to inspect the security vulnerabilities of 8 existing secure branch predictors and four typical hardware-based speculative execution defenses. The result shows that the secure branch predictor design is a promising solution for preserving the confidentiality and integrity of computer systems.

Acknowledgment

This work was supported by National Key R&D Program of China under Grant No. 2022YFB3103800 and National Natural Science Foundation of China under Grant No. 61972295. We would also like to thank the anonymous reviewers for their constructive and insightful comments.

References

[1] O. Acıiçmez, S. Gueron, and J.-P. Seifert, "New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures," in *Proceedings of the 11th IMA International Conference on Cryptography and Coding*. Springer, 2007, pp. 185–203.

[2] O. Aciiçmez, Ç. K. Koç, and J.-P. Seifert, "On the Power of Simple Branch Prediction Analysis," in *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*. ACM, 2007, pp. 312–320.

[3] O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert, "Predicting Secret Keys via Branch Prediction," in *Proceedings of the 7th Cryptographers' track at the RSA conference on Topics in Cryptology*. Springer, 2007, pp. 225–242.

[4] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM Side-Channel(s): Attacks and Assessment Methodologies," in *4th International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 29–45.

[5] 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)*. USENIX Association, 2022, pp. 971–988.

[6] G. Barthe, S. Cauligi, B. Grégoire, A. Koutsos, K. Liao, T. Oliveira, S. Priya, T. Rezk, and P. Schwabe, "High-Assurance Cryptography in the Spectre Era," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1884–1901.

[7] M. Behnia, P. Sahu, R. Paccagnella, J. Yu, Z. N. Zhao, X. Zou, T. Unterluggauer, J. Torrellas, C. Rozas, A. Morrison *et al.*, "Speculative Interference Attacks: Breaking Invisible Speculation Schemes," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2021, pp. 1046–1060.

[8] S. Bhattacharya, C. Maurice, S. Bhasin, and D. Mukhopadhyay, "Template Attack on Blinded Scalar Multiplication with Asynchronous perf-ioctl Calls," *Cryptology ePrint Archive*, 2017.

[9] S. Bhattacharya, C. Maurice, S. Bhasin, and D. Mukhopadhyay, "Branch Prediction Attack on Blinded Scalar Multiplication," *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 633–648, 2019.

[10] A. Bhattacharyya, A. S. Marin, E. M. Koruyeh, N. Abu-Ghazaleh, C. Song, and M. Payer, "SpecROP: Speculative Exploitation of ROP Chains," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. USENIX Association, 2020, pp. 1–16.

[11] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer, and A. Kurmus, "SMoTherSpectre: Exploiting Speculative Execution through Port Contention," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019, pp. 785–800.

[12] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, and S. Devadas, "MI6: Secure Enclaves in a Speculative Out-of-Order Processor," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2019, pp. 42–56.

[13] Y. Bulygin, "CPU side-channels vs. virtualization malware: The good, the bad or the ugly," *ToorCon*, 2008.

[14] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A Systematic Evaluation of Transient Execution Attacks and Defenses," in *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019, pp. 249–266.

[15] K. Cheang, C. Rasmussen, S. Seshia, and P. Subramanyan, "A Formal Approach to Secure Speculation," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019, pp. 288–303.

[16] C. Chen, C. Shen, and J. Zhang, "Lightweight and Secure Branch Predictors against Spectre Attacks," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 25–30.

[17] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[18] M. H. I. Chowdhuryy and F. Yao, "Leaking Secrets through Modern Branch Predictor in the Speculative World," *IEEE Transactions on Computers*, vol. 71, no. 9, pp. 2059–2072, 2021.

[19] L.-A. Daniel, S. Bardin, and T. Rezk, "Hunting the Haunter — Efficient Relational Symbolic Execution for Spectre with Haunted RelSE," in *Network and Distributed Systems Security (NDSS) Symposium 2021*. ISOC, 2021.

[20] S. Deng, W. Xiong, and J. Szefer, "Analysis of Secure Caches Using a Three-Step Model for Timing-Based Attacks," *Journal of Hardware and Systems Security*, vol. 3, no. 4, pp. 397–425, 2019.

[21] S. Deng, W. Xiong, and J. Szefer, "Secure TLBs," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2019, pp. 346–359.

[22] S. Deng, W. Xiong, and J. Szefer, "A Benchmark Suite for Evaluating Caches' Vulnerability to Timing Attacks," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2020, pp. 683–697.

[23] P. W. Deutsch, W. T. Na, T. Bourgeat, J. S. Emer, and M. Yan, "Metior: A Comprehensive Model to Evaluate Obfuscating Side-Channel Defense Schemes," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. ACM, 2023, pp. 1–16.

[24] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Covert channels through branch predictors: a feasibility study," in *Proceedings of the 4th Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2015, pp. 1–8.

[25] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Jump over ASLR: Attacking branch predictors to bypass ASLR," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[26] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Understanding and Mitigating Covert Channels Through Branch Predictors," *ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 1, pp. 1–23, 2016.

[27] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev, "BranchScope: A New Side-Channel Attack on Directional Branch Predictor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 693–707.

[28] X. Fabian, M. Guarnieri, and M. Patrignani, "Automatic Detection of Speculative Execution Combinations," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 965–978.

[29] J. Fustos, M. Bechtel, and H. Yun, "SpectreRewind: Leaking Secrets to Past Instructions," in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*. ACM, 2020, pp. 117–126.

[30] R. Guanciale, M. Balliu, and M. Dam, "InSpectre: Breaking and Fixing Microarchitectural Vulnerabilities by Formal Analysis," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 1853–1869.

[31] M. Guarnieri, B. Köpf, J. F. Morales, J. Reineke, and A. Sánchez, "SPECTECTOR: Principled Detection of Speculative Information Flows," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1–19.

[32] M. Guarnieri, B. Köpf, J. Reineke, and P. Vila, "Hardware-Software Contracts for Secure Speculation," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1868–1883.

[33] S. Guo, Y. Chen, P. Li, Y. Cheng, H. Wang, M. Wu, and Z. Zuo, "SpecuSym: Speculative symbolic execution for cache timing leak detection," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. IEEE, 2020, pp. 1235–1247.

[34] Z. He, G. Hu, and R. Lee, "New Models for Understanding and Reasoning about Speculative Execution Attacks," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 40–53.

[35] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Elsevier, 2011.

[36] G. Hu, Z. He, and R. B. Lee, "SoK: Hardware Defenses Against Speculative Execution Attacks," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2021, pp. 108–120.

[37] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari, "Understanding contention-based channels and using them for defense," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 639–650.

[38] T. Huo, X. Meng, W. Wang, C. Hao, P. Zhao, J. Zhai, and M. Li, "Bluethunder: A 2-level Directional Predictor Based Side-Channel Attack against SGX," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 1, pp. 321–347, 2020.

[39] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 974–987.

[40] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre Attacks: Exploiting Speculative Execution," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[41] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. Springer, 1999, pp. 388–397.

[42] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Spectre Returns! Speculation Attacks using the Return Stack Buffer," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018, pp. 1–12.

[43] J. Lee, Y. Ishii, and D. Sunwoo, "Securing Branch Predictors with Two-Level Encryption," *ACM Transactions on Architecture and Code Optimization*, vol. 17, no. 3, pp. 1–25, 2020.

[44] X. Lou, T. Zhang, J. Jiang, and Y. Zhang, "A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–37, 2021.

[45] G. Maisuradze and C. Rossow, "ret2spec: Speculative Execution Using Return Stack Buffers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2109–2122.

[46] N. Mosier, H. Lachnitt, H. Nemati, and C. Trippel, "Axiomatic hardware-software contracts for security," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ACM, 2022, pp. 72–86.

[47] H. Ponce-de León and J. Kinder, "Cats vs. Spectre: An Axiomatic Approach to Modeling Speculative Execution Attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 235–248.

[48] J. Ravichandran, W. T. Na, J. Lang, and M. Yan, "PACMAN: attacking ARM pointer authentication with speculative execution," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ACM, 2022, pp. 685–698.

[49] M. Schwarz, M. Schwarzl, M. Lipp, J. Masters, and D. Gruss, "NetSpectre: Read Arbitrary Memory over Network," in *24th European Symposium on Research in Computer Security*. Springer, 2019, pp. 279–299.

[50] A. Seznec, "A 256 kbits l-tage branch predictor," *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)*, vol. 9, pp. 1–6, 2007.

[51] J. Szefer, "Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses," *Journal of Hardware and Systems Security*, vol. 3, no. 3, pp. 219–234, 2019.

[52] Y. Tan, J. Wei, and W. Guo, "The Micro-architectural Support Countermeasures against the Branch Prediction Analysis Attack," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2014, pp. 276–283.

[53] M. Taram, A. Venkat, and D. Tullsen, "Context-Sensitive Fencing: Securing Speculative Execution via Microcode Customization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, pp. 395–410.

[54] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 681–698.

[55] C. Trippel, D. Lustig, and M. Martonosi, "MeltdownPrime and SpectrePrime: Automatically-Synthesized Attacks Exploiting Invalidation-Based Coherence Protocols," *arXiv preprint arXiv:1802.03802*, 2018.

[56] I. Vougioukas, N. Nikoleris, A. Sandberg, S. Diestelhorst, B. M. Al-Hashimi, and G. V. Merrett, "BRB: Mitigating Branch Predictor Side-Channels," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 466–477.

[57] G. Wang, S. Chattopadhyay, A. K. Biswas, T. Mitra, and A. Roychoudhury, "KLEESpectre: Detecting Information Leakage through Speculative Cache Attacks via Symbolic Execution," *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 3, pp. 1–31, 2020.

[58] J. Wikner and K. Razavi, "RETBLEED: Arbitrary Speculative Code Execution with Return Instructions," in *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 2022, pp. 3825–3842.

[59] W. Xiong and J. Szefer, "Survey of Transient Execution Attacks and Their Mitigations," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–36, 2021.

[60] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. Fletcher, and J. Torrellas, "InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 428–441.

[61] Y. Yang, T. Bourgeat, S. Lau, and M. Yan, "Pensieve: Microarchitectural Modeling for Security Evaluation," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*. ACM, 2023, pp. 1–15.

[62] J. Yu, M. Yan, A. Khyzha, A. Morrison, J. Torrellas, and C. W. Fletcher, "Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2019, pp. 954–968.

[63] T. Zhang, K. Koltermann, and D. Evtyushkin, "Exploring Branch Predictors for Constructing Transient Execution Trojans," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2020, pp. 667–682.

[64] L.-T. Zhao, R. Hou, K. Wang, Y.-L. Su, P.-N. Li, and D. Meng, "A Novel Probabilistic Saturating Counter Design for Secure Branch Predictor," *Journal of Computer Science and Technology*, vol. 36, pp. 1022–1036, 2021.

[65] L. Zhao, P. Li, R. Hou, M. C. Huang, J. Li, L. Zhang, X. Qian, and D. Meng, "A Lightweight Isolation Mechanism for Secure Branch Predictors," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1267–1272.

[66] L. Zhao, P. Li, R. Hou, M. C. Huang, X. Qian, L. Zhang, and D. Meng, "HyBP: Hybrid Isolation-Randomization Secure Branch Predictor," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 346–359.

Appendix

A. Abstract

Our artifact contains a modeling methodology for branch predictor states and a three-step symbolic execution simulator for deriving vulnerable three-step attack patterns. The inputs of the simulator can be configured to model different branch predictor designs and speculative execution attack countermeasures. The output of the simulator includes valid attack patterns vulnerable to the given design and the number of different attack categories. Researchers can analyze their branch predictor designs by customizing the code in `src/simulator.rs` to add additional branch predictor states, operations, and update logic for the simulator.

B. Artifact check-list (meta-information)

- **Algorithm:** Three-step branch predictor simulator based on symbolic execution
- **Run-time environment:** Rust 2021
- **Output:** Terminal outputs including expected results

- **Experiments:** Derivation of valid attack patterns and evaluation of existing defenses
- **How much disk space required (approximately)?:** 10 MB
- **How much time is needed to complete experiments (approximately)?:** 5 minutes
- **Publicly available?:** Yes, available at https://doi.org/10.5281/zenodo.10297402
- **Code licenses (if publicly available)?:** Apache License 2.0
- **Archived (provide DOI)?:** 10.5281/zenodo.10297402

C. Description

1) How to access: Our branch predictor modeling approach and simulator are available at https://doi.org/10.5281/zenodo.10297402. The newest version of the simulator is available at https://github.com/iamywang/bp-security-framework.

D. Installation

1. Install Rust:

`$ sudo apt install rustc` (Ubuntu 18.04 or later)

`$ sudo pacman -S rust` (Arch Linux)

2. Build the simulator:

```
$ cd bp-sec-sim
$ cargo build --release
$ cp ./target/release/bp-sec-sim ./
```

E. Evaluation and expected results

Derivation of All 156 Three-Step Vulnerabilities. This experiment is used to reproduce valid attack patterns listed in Table IV, including attack categories, attack patterns, and the execution timing of secret-dependent branches.

Running the following command will generate terminal output expected to be similar to *expected_res/exp1_derivation.out*, which contains all the 156 attack patterns derived by our simulator.

```
$ ./bp-sec-sim exp1_derivation
```

Security Analysis of 8 Secure Branch Predictors. This experiment is used to reproduce the results listed in Table VI, Table VII, and Table VIII.

Running the command below will generate terminal output expected to be similar to *expected_res/exp2_rsb_refilling.out*, which contains the number of vulnerabilities that cannot be covered by RSB refilling.

```
$ ./bp-sec-sim exp2_rsb_refilling
```

Running the following command will generate terminal output expected to be similar to *expected_res/exp2_secure_bp.out*, which contains the number of vulnerabilities for each secure branch predictor.

```
$ ./bp-sec-sim exp2_secure_bp
```

Security Analysis of 4 Hardware-Based Speculative Attack Countermeasures. This experiment is used to reproduce the results listed in Table IX.

Running the command below will generate terminal output expected to be similar to *expected_res/exp3_baseline_bp.out*, which contains the number of speculative attacks for the baseline branch predictor.

```
$ ./bp-sec-sim exp3_baseline_bp
```

Running the following command will generate terminal output expected to be similar to *expected_res/exp3_secure_bp.out*, which contains the number of speculative attacks for each secure branch predictor.

```
$ ./bp-sec-sim exp3_secure_bp
```

Running the command below will generate terminal output expected to be similar to *expected_res/exp3_hw_defenses.out*, which contains the number of speculative attacks for each hardware-based countermeasure.

```
$ ./bp-sec-sim exp3_hw_defenses
```

Security Analysis of Two Modeling Approaches for TAGE Branch Predictor. This experiment is used to reproduce the results of Case Study I in Section IV-C.

Running the following command will generate terminal output expected to be similar to *expected_res/exp4_tage.out*, which contains the number of vulnerabilities for the TAGE branch predictor.

```
$ ./bp-sec-sim exp4_tage
```

F. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html