Pitfall: Uncovering and Exploiting the Store Forwarding Predictor on Intel CPUs

Chang Liu[1 [0000−0002−1834−4958]], Xin Zhang[2 (✉) [0000−0003−4185−7214]], Dapeng Ju[1 [0009−0005−6495−2275]], Yongqiang Lyu[1 [0000−0003−2573−963X]], and Dongsheng Wang[1 [0000−0001−5779−9026]]

[1] Tsinghua University, Beijing, China
chang-li17@tsinghua.org.cn, {judapeng,luyq,wds}@tsinghua.edu.cn
[2] Shandong University, Qingdao, China
zhangxin00sdu@gmail.com

Abstract. To bridge the widening gap between CPU execution speed and memory access latency, modern CPUs incorporate a variety of predictors in the memory subsystem to enable high-accuracy data-flow speculation. While these predictors improve performance, they also introduce new security risks.

In this paper, we identify a Store Forwarding Predictor (SFP) in recent Intel desktop and server CPUs. The SFP predicts whether a store can forward its data to a dependent load before the load address becomes available, thereby increasing the parallelism of loads and subsequent data-dependent instructions.

To evaluate the security implications of the SFP, we first reverse-engineer its design, including its state machine, indexing mechanism, and flushing behavior. Based on this analysis, we propose Pitfall, the first side-channel attack exploiting the Intel SFP. Pitfall-v1 exploits SFP mispredictions to trigger in-place transient execution attacks. Pitfall-v2 exploits state sharing across different store-load pairs in the SFP to enable stealthier out-of-place transient execution attacks. Pitfall-v3 leverages SFP state flushing during context switches to detect interrupts, achieving up to 97.6% top-5 accuracy in inferring website visits in Google Chrome. The source code of Pitfall is available at https://github.com/cliu2025/pitfall.

Keywords: Store Forwarding Predictor · Reverse Engineering · CPU Security · Microarchitectural Vulnerability · Side-channel Attack.

1 Introduction

To improve performance, modern CPUs employ speculative execution to increase instruction-level parallelism. Beyond the well-known control-flow speculation based on branch predictors, modern CPUs further introduce data-flow speculation in the backend to accelerate memory operations. For example, CPUs from Intel, AMD, and Arm deploy Memory Dependence Predictors (MDPs) to predict data dependencies between loads and stores whose addresses are not yet

resolved [11,12,13,14]. In addition, Apple CPUs introduce a Load Address Predictor (LAP) [8] and a Load Value Predictor (LVP) [7], which predict the load address or value directly when the load address is unavailable.

However, aggressive data-flow speculation also introduces significant security risks. Incorrect speculation in the MDP, LAP, and LVP can be exploited to construct transient execution attacks [7,8,13]. Moreover, the MDP state updates can be used to build cross-domain side-channel attacks [11]. The security implications of these predictors have been widely studied.

Nevertheless, beyond the MDP, LAP, and LVP, modern CPUs may include additional predictors to support more advanced data-flow speculation. For example, Mascot [15] extends the MDP by further predicting whether a store can directly forward its data to a load, effectively combining the function of LVP. In this paper, we refer to such predictors as Store Forwarding Predictors (SFPs).

Existing work on SFPs remains limited. Prior studies only identify their presence on AMD Zen 3 CPUs [13]. For other architectures, whether an SFP is deployed remains unclear. This leads to the following questions:

Is an SFP implemented in CPUs beyond AMD Zen 3? If so, do these SFP designs introduce vulnerabilities?

This paper answers the above questions by identifying and characterizing the SFP on recent Intel CPUs. By constructing microbenchmarks, we identify an SFP, whose design remains undocumented, on seven Intel 11th-generation and later Core CPUs, as well as one Xeon CPU. We further characterize the activation behavior of the SFP and its cross-address state sharing property. We also analyze the security properties of the SFP and find that its state is flushed during context switches, which provides strong inter-process isolation.

Based on these findings, we identify three vulnerabilities in the Intel SFP. First, misprediction of the SFP will cause speculative execution, enabling a new variant of transient execution attacks. Second, the SFP allows out-of-place training within the same process. Third, the flush behavior of the SFP during context switches provides a deterministic signal indicating whether a process has been interrupted, which can be leveraged to build an interrupt side-channel attack [21,23,24].

To exploit these vulnerabilities, we propose Pitfall, the first side-channel attack that exploits the Intel SFP. Pitfall includes three variants: Pitfall-v1, which performs an in-place transient execution attack; Pitfall-v2, which performs an out-of-place transient execution attack; and Pitfall-v3, which implements an interrupt side channel. On P-cores, Pitfall-v1 and Pitfall-v2 achieve near-100% accuracy and a throughput above 2600 Bps. Pitfall-v3 reaches a top-5 accuracy of up to 96.6% in a website fingerprinting attack over 100 webpages. Finally, we discuss possible defenses against Pitfall and evaluate the performance overhead of disabling the SFP, which can be as high as 12.4%.

In summary, this paper makes the following contributions:

- We identify the existence of an SFP on seven Intel CPUs for the first time.
- We conduct an in-depth analysis of the Intel SFP, characterizing its activation conditions, state sharing, and flush behavior during context switches.

– We propose Pitfall, the first side-channel attack that exploits the Intel SFP. We implement three variants of Pitfall and evaluate their performance. We also discuss the defenses and analyze the performance overhead of disabling the SFP.

2 Background and Related Work

2.1 Speculative Execution of Load Instructions

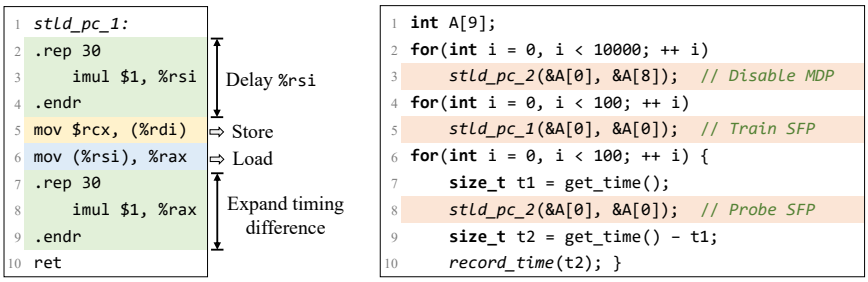
Speculative execution is a common technique used on modern CPUs to improve performance. Some CPUs employ load speculation before the load address is ready, or when there exist preceding stores with unresolved addresses [7,8,12,13]. For example, speculative store forwarding predicts whether a store can directly forward its data to a load when either the store or load address is not ready.

Modern CPUs typically rely on dedicated predictors to support these mechanisms. In particular, the SFP predicts whether speculative store forwarding should be performed [15]. The SFP usually records store-load pairs that access the same address (i.e., they are dependent), and speculatively forwards the store data when the same pair is executed again. As a result, the SFP is typically indexed by the instruction addresses (i.e., Program Counters, PCs) of the store and load. Similar to branch predictors, the SFP maintains a counter to represent prediction confidence, and enables speculative forwarding only after sufficient training [15]. In this paper, we analyze both the indexing and the activation conditions of the SFP on recent Intel CPUs.

2.2 Transient Execution Attacks

Transient execution attacks are a class of side-channel attacks that exploit misspeculation and microarchitectural behaviors. Speculatively executed instructions are not committed, and therefore do not modify architectural state such as registers or memory. However, they leave observable traces in the microarchitectural state. For example, a load executed speculatively may still issue a memory request and bring data into the cache. As a result, a common attack pattern is to trigger misspeculation in victim code, perform out-of-bounds memory accesses using speculative instructions, and use a cache side channel to record the accessed data [2].

Transient execution attacks that rely on misprediction can be categorized into *in-place* and *out-of-place* variants. In an in-place attack, both predictor training and misprediction are triggered using the same instruction at the same address. In contrast, an out-of-place attack allows training and misprediction to occur at different addresses. Out-of-place attacks significantly reduce the number of required victim invocations, which improves stealth and helps bypass defenses based on call frequency or monitoring [3,9]. In this paper, we show that the SFP on Intel CPUs supports both in-place and out-of-place attacks.



(a) Microbenchmark (b) Usage of the microbenchmarks

Fig. 1: Microbenchmarks used for SFP characterization.

3 Characterization and Security Analysis of the Intel SFP

In this section, we first identify the SFP on Intel CPUs in Section 3.1. We then characterize the Intel SFP in Section 3.2, including its activation conditions and cross-address sharing. We analyze its security implications in Section 3.3. All experiments are conducted on the ten Intel CPUs listed in Table 1.

3.1 SFP Identification

We adapt prior microbenchmark designs [13] to enable the analysis of the SFP, as shown in Fig. 1a. To trigger the SFP, we introduce latency in load address generation by inserting 30 multiplication instructions. Since the load address is not ready, the MDP is not activated [11]. In contrast, the store data is ready. Therefore, if the SFP is present, the store data may be forwarded to the load speculatively, allowing the subsequent 30 multiplications to execute earlier. While delaying the store address, as done in [13], can also activate the SFP, it simultaneously triggers the MDP, which greatly complicates the analysis. Therefore, we specifically choose to delay the load address to characterize the SFP.

Depending on whether the SFP prediction is triggered and whether the prediction is correct, the microbenchmark exhibits three possible execution times. First, if the SFP is absent or predicts independence, the load is stalled, leading to a longer execution time T_1. Second, if the SFP predicts dependence and the prediction is correct, forwarding is triggered, resulting in the shortest execution time T_2. Finally, if the SFP predicts dependence but is incorrect, the CPU rolls back and re-executes the load, leading to the longest execution time T_3.

To identify the existence of the SFP, we aim to distinguish between T_1 and T_2. Specifically, we first set the store and load to different addresses and execute the microbenchmark 50 times. We then set the store and load to the same address and execute another 50 runs. If an SFP exists, the first 50 executions train the SFP toward independence, resulting in execution time T_1. In the following 50 executions, the data dependence causes the SFP to adjust its prediction at some point, leading to a transition in execution time from T_1 to T_2.

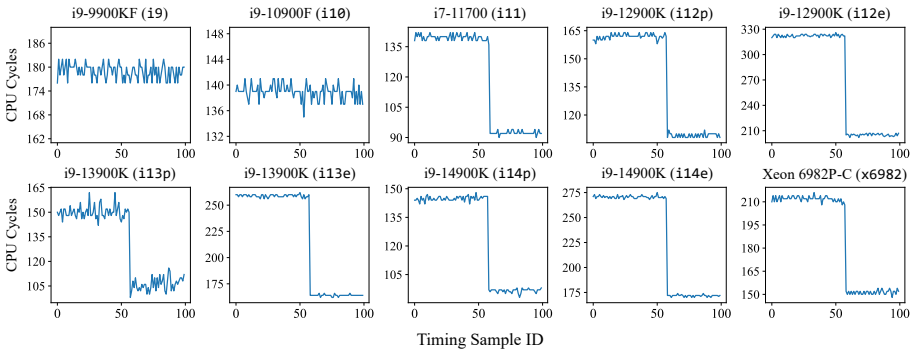


Fig. 2: Existence test of the SFP across ten Intel CPUs.

To mitigate any potential effects from LVP and LAP in case they exist on the target CPUs, we randomize all load addresses and values, as these predictors could otherwise be triggered during periods of unresolved load addresses.

We conduct the above experiment on ten Intel CPUs, and the results are shown in Fig. 2. The older CPUs such as `i9` and `i10` show no step change in execution time, indicating that the SFP is not present. In contrast, on 11th-generation and later Core and Xeon CPUs, a clear transition in execution time appears after approximately ten executions of dependent store-load pairs. This behavior indicates that the SFP is deployed on these CPUs.

3.2 SFP Characterization

In Section 3.1, we observe that the activation threshold of the SFP is typically below 10 executions. We further validate this observation and find that at most eight executions of dependent store-load pairs are sufficient to activate the SFP. Based on this result, we infer that the Intel SFP uses a 3-bit saturating counter to track prediction confidence. When a dependent store-load pair is observed for the first time, the counter is initialized to zero. Each subsequent dependent execution increments the counter, and once the counter reaches saturation, the predictor outputs a dependence prediction.

We further extend the microbenchmark to study the cross-address sharing of the SFP, as shown in Fig. 1b. We use two microbenchmarks with identical instruction sequences, but vary the PCs of the stores and loads. We then align several least significant bits (LSBs) of the store and load PCs and observe whether the SFP state is shared. We conduct experiments on eight CPUs that deploy an SFP, as shown in Fig. 3. The results indicate that when the 16 LSBs of the load PC are aligned, the SFP prediction is consistently triggered. When the 16 LSBs of the store PC are aligned, the SFP prediction is also consistently triggered on some Efficiency Cores (E-cores). These observations suggest that although the SFP is indexed by both store and load PCs, it only uses their lower bits for indexing. As a result, ensuring that the lower 16 bits of the store and

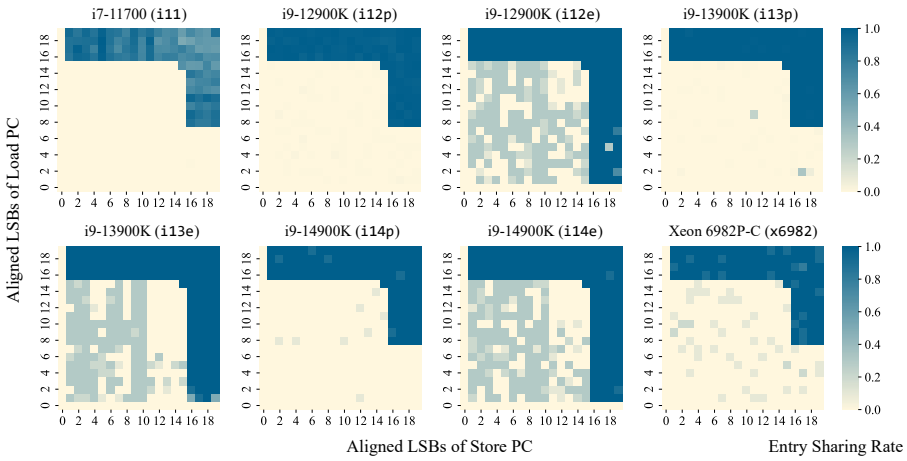


Fig. 3: Indexing test of the SFP across eight Intel CPUs.

load PCs match is sufficient to make two store-load pairs at different addresses share the same SFP entry on both Performance Cores (P-cores) and E-cores.

3.3 SFP Security Analysis

We finally analyze the isolation of the SFP across different security domains. We first run identical store-load pairs at the same instruction addresses in two processes and observe that the prediction state is not shared, regardless of whether the processes execute on the same core. In particular, when the two processes run on the same core, each context switch resets the SFP state. This result indicates that the Intel SFP is private to each core and enforces inter-process isolation.

To further examine when the SFP state is reset, we trigger a system call after training the SFP and probe its state in kernel space. We observe that the SFP state is already cleared. This suggests that the SFP is flushed during context switches caused by privilege-level transitions. Although this mechanism prevents cross-process training, it also introduces a side effect: a process can infer whether a context switch has occurred by observing changes in the SFP state. This behavior enables an interrupt side-channel attack. In Section 4.4, we exploit this mechanism to construct such an attack.

Table 1 summarizes our identification and analysis of the SFP. We identify the SFP on eight recent Intel CPUs and characterize its activation conditions, cross-address sharing, and flush behavior during context switches.

4 Pitfall: Exploiting the Intel SFP to Leak Secrets

In this section, based on the characterization and security analysis of the SFP, we propose Pitfall, the first side-channel attack that exploits the Intel SFP to leak

Table 1: Summary of SFP characterization across ten Intel CPUs

Device*		**Design Characteristics**			**Isolation**	
CPU	OS	Existence	Store PC**	Load PC	Intra-process	Inter-process
`i9`	ubuntu 22.04	✗	–	–	–	–
`i10`	ubuntu 20.04	✗	–	–	–	–
`i11`	ubuntu 22.04	✔	16	16	✔	✗
`i12e`	ubuntu 22.04	✔	16	16	✔	✗
`i12p`	ubuntu 22.04	✔	16	16	✔	✗
`i13e`	ubuntu 20.04	✔	16	16	✔	✗
`i13p`	ubuntu 20.04	✔	16	16	✔	✗
`i14e`	ubuntu 22.04	✔	16	16	✔	✗
`i14p`	ubuntu 22.04	✔	16	16	✔	✗
`x6982`	ubuntu 24.04	✔	16	16	✔	✗

* `i9`: Intel Core i9-9900KF. `i10`: Intel Core i9-10900F. `i11`: Intel Core i11-11700. `i12e/i12p`: Intel Core i9-12900K efficiency/performance cores. `i13e/i13p`: Intel Core i9-13900K efficiency/performance cores. `i14e/i14p`: Intel Core i9-14900K efficiency/performance cores. `x6982`: Intel Xeon 6982P-C.
** Number of aligned least significant bits of store PC to ensure an SFP entry sharing.

secrets from user space. The workflows of the three variants of Pitfall are shown in Fig. 4. In Section 4.2, we present Pitfall-v1, an in-place transient execution attack that exploits the misprediction of the SFP. Next, in Section 4.3, we present Pitfall-v2, an out-of-place transient execution attack based on entry sharing of the SFP among different instruction PCs. This method significantly reduces the number of victim function invocations. Finally, in Section 4.4, we present Pitfall-v3, which leverages the flush behavior of the SFP during context switches to launch an interrupt side-channel attack.

4.1 Threat Model

In this work, we adopt a common local user-space threat model. We assume that the attacker has user-level privileges, can execute unprivileged code, and uses `rdtsc` for fine-grained timing.

For transient execution attacks, including Pitfall-v1 and Pitfall-v2, we assume the attacker and the victim reside within the same process. This aligns with standard threat models established in prior literature [7,8,13,24], where the attacker cannot directly access secrets protected by sandboxing or similar isolation mechanisms. Under this model, the attacker either possesses the capability to inject malicious code or relies on exploitable gadgets already present in the victim's code. Discovering such gadgets within victim software is an orthogonal research challenge [10,11,16], and thus we defer systematic gadget discovery to future work. By exploiting transient execution, the attacker can speculatively leak data from arbitrary addresses within the victim's domain, enabling cross-domain data leakage in web browsers (e.g., leveraging JIT compilation to exfiltrate data from one webpage to another [7,8,13]).

For Pitfall-v3, we adopt a cross-process/cross-core threat model widely used in prior website fingerprinting works [4,21,24]. Under this model, the attacker

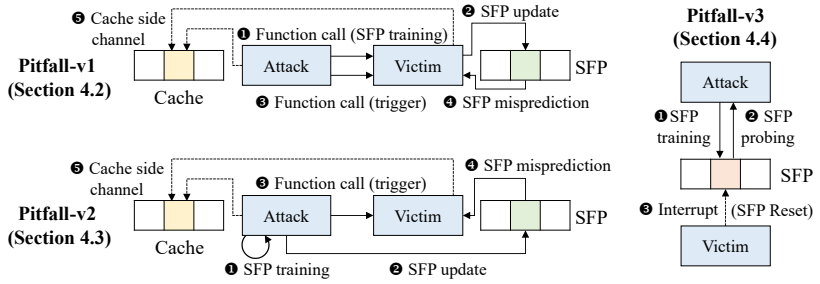


Fig. 4: Attack workflows of Pitfall-v1, Pitfall-v2, and Pitfall-v3.

```c
uint8_t array1[16];
uint64_t array2[256 * 512];
void victim_function(int st_idx, int fwd_val) {
    array2[st_idx * 512] = fwd_val; // store
    uint64_t v = array2[0]; // load-1
    uint8_t  s = array1[v]; // load-2
    uint64_t temp;
    temp = array2[s * 512]; // load-3
    return;
}
```

(a) Victim code in C

```
// rdi: &st_idx, r8: array1, r9: array2
...
mov    (%rdi),%rax
shl    $0xc,%rax
add    %r9,%rax                          Forwarding
mov    %rsi,(%rax)          // store
mov    (%r9),%rdx           // load-1
movzbq (%r8,%rdx,1),%rdx    // load-2
shl    $0xc,%rdx
mov    (%r9,%rdx,1),%rax    // load-3
```

(b) Victim code in Assembly

Fig. 5: Victim code in C and Assembly used in Pitfall-v1 and Pitfall-v2.

and the victim execute in separate processes or even on different cores, and the victim uses a stock, unmodified web browser. The attacker probes the SFP to observe the interrupts triggered by the victim process, and then infers which webpage the victim is accessing.

4.2 Pitfall-v1: In-place Transient Execution Attack

Pitfall-v1 exploits mispredictions in the SFP to perform a transient execution attack. As shown in Fig. 4, the attacker first calls a victim function in the victim address space (❶). This function contains a store-load pair with address latency. The attacker sets the store and load to the same address and repeatedly executes the victim function to train the SFP toward a dependence prediction (❷). Next, the attacker calls the victim function again (❸), but sets the store and load to different addresses, which triggers a misprediction in the SFP (❹). Under the misprediction, the store wrongly forwards its data to the load. Similar to Spectre-v1, the attacker uses the mispredicted load value to perform an out-of-bounds memory access, which leaks a secret byte from the victim. The attacker then leverages a cache side channel to record and recover the secret byte (❺).

Victim function. The victim function required in Pitfall-v1 is shown in Fig. 5. The data dependence between the store and load-1 is controlled by `st_idx`. During the training phase, the attacker sets `st_idx` to 0, so the store and load-1

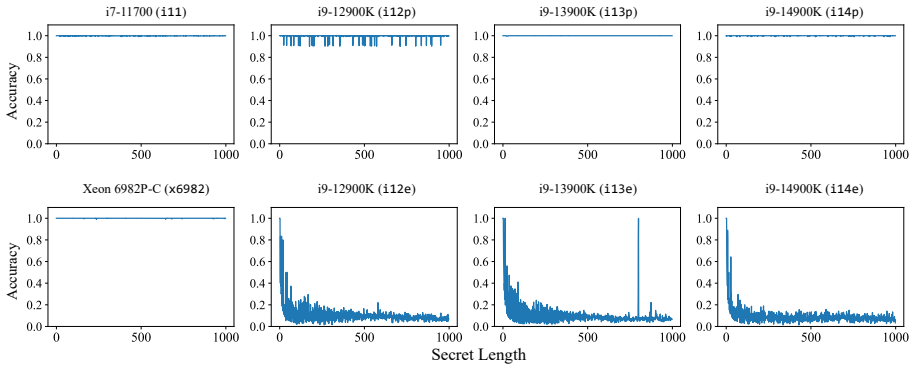


Fig. 6: Accuracy of Pitfall-v1 under different secret lengths.

access the same address and are dependent. At the same time, the attacker controls `fwd_val` to ensure that load-1 does not perform an out-of-bounds access. Next, the attacker flushes `st_idx` to introduce latency in store address generation. The attacker repeats the execution eight times to train the SFP to predict dependence and enable speculative forwarding.

During the attack phase, the attacker updates `st_idx` to 10. Under incorrect forwarding by the SFP, `fwd_val` is wrongly forwarded to `v`. The attacker carefully chooses `fwd_val` so that `array1[v]` triggers an out-of-bounds access and retrieves the secret byte `s`. Finally, the attacker uses load-3 to encode `s` into the cache state. In practice, we repeat the attack 100 times to reduce noise. We also introduce an early-stop mechanism. Once the confidence for a leaked byte exceeds a threshold, the attack terminates early.

It is worth noting that delaying either the store or the load can successfully activate the SFP. In Fig. 5, we show the store-delayed variant to provide a clearer illustration of the respective roles played by the three loads during the attack sequence. In a practical attack scenario, delaying the execution of the first load yields an identical effect.

Evaluation. We evaluate the byte recovery accuracy for different leakage lengths on CPUs summarized in Table 1 that support the SFP. The results are illustrated in Fig. 6. On the P-cores such as `i11`, `x6982`, `i12p` and others, the leakage accuracy remains close to 100%.

In contrast, on the E-cores of `i12e`, `i13e`, and `i14e`, the attack only recovers the earliest bytes, typically no more than 10 bytes. This behavior suggests the presence of a watchdog on E-cores: frequent SFP mispredictions temporarily disable the SFP. Further analysis shows that this watchdog is not affected by context switches, but it resets after a certain period. As a result, leaking long secrets on E-cores becomes time-consuming. Therefore, in the following evaluation, we focus only on P-core performance.

We further increase the length and randomness of the secret data and evaluate Pitfall-v1 on P-cores. The results are shown in Table 2. On P-cores, Pitfall-v1 achieves an accuracy above 99%. In addition, due to the early-stop mechanism, the leakage throughput reaches at least 2600 Bps.

Table 2: Evaluation of Pitfall-v1 and Pitfall-v2 across five CPUs

CPU	Pitfall-v1			Pitfall-v2		
	Accuracy	Speed/Bps	Function Call*	Accuracy	Speed/Bps	Function Call
i11	1.00	6194.82	108.14	1.00	645.77	38.38
i12p	1.00	3227.50	93.80	1.00	3596.86	3.25
i13p	1.00	3547.86	83.27	1.00	3575.69	3.35
i14p	1.00	3513.85	85.35	1.00	3515.25	3.20
x6982	1.00	2630.29	82.05	1.00	2591.55	3.14

* Number of victim function calls to leak one byte.

4.3 Pitfall-v2: Out-of-place Transient Execution Attack

Pitfall-v2 improves upon Pitfall-v1 by exploiting the out-of-place SFP training. Frequent calls of the victim function can be detected by hardware performance counters or software sampling interfaces [9], which may expose the presence of the attack. To improve stealth, as shown in Fig. 4, Pitfall-v2 replaces the first two steps of the attack. Specifically, it exploits the fact that SFP entries can be shared. The attacker constructs a store-load pair in their own address space that maps to the same SFP entry as the victim (❷), and uses this pair for training (❸). In our implementation, we align the 16 LSBs of the attacker's store and load with the store (line 6) and load (line 7) in Fig. 5b, respectively.

Evaluation. The performance of Pitfall-v2 on P-cores is shown in Table 2. On all P-cores except `i11`, Pitfall-v2 achieves performance comparable to Pitfall-v1, while reducing the number of victim function invocations by more than 20×. On the `i11`, however, Pitfall-v2 shows lower performance. This is because the attacker's training affects the cache state, which increases noise in the cache side channel and prevents the early-stop mechanism from taking effect promptly.

4.4 Pitfall-v3: Website Fingerprinting Attack

In this section, we apply Pitfall-v3 to mount a representative interrupt side-channel attack, i.e., website fingerprinting. Our key insight is that, when accessing a website, distinctive network packets will be sent and received, generating distinctive network interrupts. A co-located attacker can use Pitfall-v3 to capture these temporal patterns and use a machine learning (ML) classifier to associate an observed interrupt trace with the corresponding website, resulting in an ML-based side-channel attack [4,6,22,23,24].

Attack design. Pitfall-v3 exploits the following two key properties. First, the SFP state is completely flushed upon context switches. As illustrated in Fig. 7, the attacker deploys an interrupt-probing process that repeatedly executes a loop at fixed intervals (e.g., every 1 second). This loop comprises three sequential phases: *prime*, *wait*, and *probe*. During the *prime* phase, the attacker executes a sufficient number of dependent store-load pairs to fully train a specific entry within the SFP prediction table, saturating its internal counter to 7. In the *wait* phase, the attacker remains quiescent by executing a small sequence of `nop`s.

```
for(int i = 0; i < sample_size; ++ i) {
    while(time_interval_less_than_one_second()) { // sample the number of interrupts within 1s
        sfp_init_to_7(); // Initialize one of the SFP prediction table entries to 7
        wait(); // wait for a few microseconds, during which a context switch may happen
        int sfp_counter_val = sfp_probe(); // Probe the SFP state
        if (sfp_counter_val < 7) { // Detect a flush of SFP state
            number_of_context_switches += 1;
        }
    }
}
```

Fig. 7: C implementation of the Pitfall-v3 attack

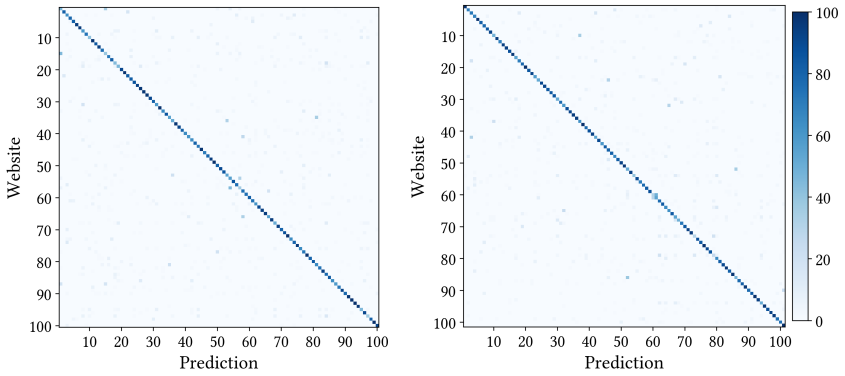
Finally, in the *probe* phase, the attacker evaluates the current SFP state. If the SFP prediction table has been flushed, the attacker can reliably infer that an interrupt-induced context switch transpired on the resident core during the preceding *wait* phase.

Second, hardware interrupts in Linux systems are inherently distributed across multiple cores [24]. Consequently, when a browser triggers an interrupt during webpage rendering or complex JavaScript execution, this interrupt may be routed to a core hosting the attacker's interrupt-probing process. In the current attack setup, the probing process is pinned to a single core. While this configuration may omit several cross-core interrupts, experimental results demonstrate that it remains sufficient for robust website fingerprinting. The performance of this side channel can be further improved by deploying concurrent interrupt-probing threads across all available CPU cores.

Our attack consists of an offline phase and an online phase. In the offline phase, the attacker collects interrupt traces generated during website loading and uses them to train a prediction model. In the online phase, the trained model takes a newly observed interrupt trace as input and outputs the most likely website visited. In our evaluation, we directly feed raw Pitfall-v3 traces into a 32-unit LSTM model, following Cook et al. [4]. We perform 10-fold partitioning over the collected traces, reserving one fold for evaluation and splitting the remaining nine folds into a training set and a validation set.

Experimental setup. We evaluate website fingerprinting in both closed-world and open-world settings. In the closed-world case, the attacker is assumed to have prior knowledge of the entire set of websites that the victim may access. The inference task is then to identify which one of 100 candidate websites was visited. Exactly aligned with prior work [4,20], we select the top 100 websites from Alexa [1] and collect 100 interrupt traces for each website. Although the Alexa Top-100 list is no longer maintained, the websites used in our evaluation remain active and accessible. This website set has also been used in recent website fingerprinting studies [5,6,20], making our setup comparable to prior work.

In the open-world case, we augment the monitored set with a single class representing websites outside the top 100. For this purpose, we collect one trace from each of 4000 additional websites drawn from the Alexa Top 1 Million list, excluding the monitored websites. This configuration follows the standard setup used in prior website fingerprinting work [4,17,20]. We acknowledge that the



(a) Closed-world Setting (b) Open-world Setting

Fig. 8: Confusion matrices of our website fingerprinting attack using Pitfall-v3.

resulting class imbalance may affect aggregate metrics. However, a systematic analysis of this effect is broadly relevant to open-world website fingerprinting and side-channel evaluations, rather than being specific to Pitfall. We therefore leave such an analysis to future work.

Experimental results. Figure 8 reports confusion matrices of our attack. Each cell indicates the probability that a trace generated by a website on the y-axis is classified as the website on the x-axis. The pronounced diagonal pattern demonstrates the high classification accuracy of our attack. To evaluate end-to-end attack performance, we use top-n accuracy as the evaluation metric, which measures whether the ground-truth website appears among the n candidate websites predicted by the classifier. In the closed-world setting, random guessing over 100 candidate websites yields a top-1 accuracy of only 1%. In contrast, our attack achieves a top-1 accuracy of 78.9% and a top-5 accuracy of 96.6%. In the open-world setting, the attack achieves a top-1 accuracy of 84.5% and a top-5 accuracy of 97.6%. The higher accuracy in the open-world setting is mainly due to the classifier's strong ability to distinguish the non-monitored class from the monitored top-100 websites.

5 Mitigation

The straightforward defense against Pitfall is to disable the SFP. Due to Spectre attacks, Intel introduces Speculative Store Bypass Disable (SSBD) to disable load speculation [13], which also disables the SFP. When SSBD is enabled, the SFP neither performs prediction nor exposes its state through timing. However, enabling SSBD leads to noticeable performance degradation. We evaluate the performance overhead of SSBD on four processors (`i11`, `i12p`, `i13p`, and `i14p`) using SPEC 2017. The results are shown in Table 3. In the `intspeed` benchmark, SSBD introduces an overhead ranging from 6.1% to 12.4%.

Beyond disabling the SFP through SSBD, defenses against Pitfall-v1 and Pitfall-v2 can also rely on detecting and hardening exploitable code patterns. For

Table 3: Performance overhead of SSBD on four Intel CPUs

Configuration	`i11`	`i12p`	`i13p`	`i14p`
SSBD Off*	8.26	10.9	11.8	12.9
SSBD On**	7.78	10.1	10.5	11.8
Overhead	6.1%	7.9%	12.4%	9.3%

* Geometric mean score of SPEC `SPECspeed 2017 Integer` for SSBD Off configurations.
** Geometric mean score of SPEC `SPECspeed 2017 Integer` for SSBD On configurations.

example, a compiler can identify store-load pairs that may trigger the SFP [10,11] and are potentially exploitable, and then insert fences or enforce explicit data dependencies to prevent speculative execution on those loads. However, such approaches do not mitigate Pitfall-v3, which exploits the flush behavior of the SFP during context switches. To defend against Pitfall-v3, the security design of the SFP must be improved. For example, partitioning or index randomization can be applied to enforce isolation across security domains. These solutions, however, require hardware modifications, making them difficult to deploy immediately.

6 Future Work

In this paper, we confirm the existence of the SFP on Intel CPUs for the first time, characterize its functional properties, and demonstrate three distinct variants of exploitation. Beyond these findings, several points around the SFP remain to be explored in future research.

First, a more detailed and finer-grained reverse-engineering of the SFP can be conducted to unveil its structure. Specifically, future work could investigate the total capacity of the SFP prediction table, its associativity, and its index mapping and replacement policies. Characterizing these internal design parameters would provide deeper insights into expanding the SFP attack surface or enhancing the efficiency of existing exploits.

Second, the SFP can be exploited to construct a broader range of practical end-to-end attacks, such as executing cross-domain data leakage within web browsers, and performing transient execution attacks against the Linux kernel. For kernel-level exploitation, static or symbolic analysis techniques need to be employed to systematically search for exploitable gadgets within the kernel space [18]. Furthermore, since the SFP is automatically flushed upon context switches, specialized techniques must be devised to ensure that both the predictor training and the speculative execution phases are contained entirely within the kernel context [19].

Finally, the study of SFPs can be generalized to other CPU architectures, such as AMD and Arm. To achieve this, researchers could build upon the concepts of automated benchmarking frameworks like SSBench [12] to develop a cross-architecture SFP automated analysis tool, which automatically synthesizes microbenchmarks to detect and characterize SFP parameters across diverse hardware platforms.

7 Conclusion

In this work, we identify the presence of the SFP on Intel CPUs for the first time, and analyze its activation conditions, cross-address sharing behavior, and security properties. Based on this analysis, we propose Pitfall, the first side-channel attack that exploits the SFP. Pitfall-v1 and Pitfall-v2 achieve near 100% accuracy with a throughput exceeding 2600 Bps. Pitfall-v3 achieves up to 96.6% top-5 accuracy in a website fingerprinting attack over 100 webpages. Finally, we discuss possible defenses against Pitfall and evaluate the performance overhead of disabling the SFP, which can reach up to 12.4% on some CPUs.

Acknowledgments. This work was supported by the National Natural Science Foundation of China (Grant No. U24A6009). The technical solution and experimental design presented in this paper were independently completed by the authors. AI tools were used solely for language polishing and formatting optimization, and did not participate in the development of the research ideas or core content.

References

1. Alexa: The top 1 million sites on the web (2023), https://www.alexa.com/topsites
2. Canella, C., Van Bulck, J., Schwarz, M., Lipp, M., Von Berg, B., Ortner, P., Piessens, F., Evtyushkin, D., Gruss, D.: A Systematic Evaluation of Transient Execution Attacks and Defenses. In: USENIX Security Symposium. pp. 249–266 (2019)
3. Constable, S., Van Bulck, J., Cheng, X., Xiao, Y., Xing, C., Alexandrovich, I., Kim, T., Piessens, F., Vij, M., Silberstein, M.: AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves. In: USENIX Security Symposium. pp. 4051–4068 (2023)
4. Cook, J., Drean, J., Behrens, J., Yan, M.: There's Always a Bigger Fish: A Clarifying Analysis of a Machine-Learning-Assisted Side-Channel Attack. In: International Symposium on Computer Architecture (ISCA). pp. 204–217 (2022)
5. Feng, Y., O'Connel, S., Zhang, X., Chuengsatiansup, C., Genkin, D., Yarom, Y., Zhang, Y., Zhang, Z.: Fish and Chips: on the Root Causes of Co-located Website-Fingerprinting Attacks. IEEE Transactions on Dependable and Secure Computing (2025)
6. Feng, Y., Zhang, X., O'Connel, S., Qiu, L., Chuengsatiansup, C., Genkin, D., Yarom, Y., Zhang, Y., Zhang, Z.: TimeGaps Channels: Exploiting CPU Halted Time for Fun and Profit. In: International Symposium on Computer Architecture (ISCA) (2026)
7. Kim, J., Chuang, J., Genkin, D., Yarom, Y.: FLOP: Breaking the Apple M3 CPU via False Load Output Predictions. In: USENIX Security Symposium. pp. 2595–2614 (2025)
8. Kim, J., Genkin, D., Yarom, Y.: SLAP: Data Speculation Attacks via Load Address Prediction on Apple Silicon. In: IEEE Symposium on Security and Privacy (S&P). pp. 3549–3566 (2025)
9. Liu, C., Feng, S., Cui, J., Zheng, H., Cui, A., Dong, J., Li, Y., Carlson, T.E., Wang, D.: Thrend: Mitigating Counting Thread-based Fine-grained Timing on Arm and Apple CPUs. In: Design Automation Conference (DAC) (2026)

10. Liu, C., Feng, S., Li, Y., Wang, D., Carlson, T.E.: HoBBy: Hardening Unbalanced Branches against Control Flow Attacks on Intel SGX and AMD SEV. In: Design Automation Conference (DAC) (2025)
11. Liu, C., Feng, S., Li, Y., Wang, D., He, W., Lyu, Y., Carlson, T.E.: MDPeek: Breaking Balanced Branches in SGX with Memory Disambiguation Unit Side Channels. In: Architectural Support for Programming Languages and Operating Systems (ASPLOS). pp. 622–638 (2025)
12. Liu, C., Jin, Y., Fan, Y., Xiao, T., Yin, L., Carlson, T.E., Deng, S., Wang, D.: SSBench: Automated Characterization of Memory Dependence Predictors on Modern CPUs. In: International Symposium on Computer Architecture (ISCA) (2026)
13. Liu, C., Wang, D., Lyu, Y., Qiu, P., Jin, Y., Lu, Z., Zhang, Y., Qu, G.: Uncovering and Exploiting AMD Speculative Memory Access Predictors for Fun and Profit. In: IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 31–45 (2024)
14. Liu, C., Zheng, H., Zhang, X., Ju, D., Wang, D., Zhang, Y., Carlson, T.E.: SSBleed: Non-Speculative Side-Channel Attacks via Speculative Store Bypass on Armv9 CPUs. In: IEEE International Symposium on High-Performance Computer Architecture (HPCA) (2026)
15. Mose, K.H., Kim, S.S., Ros, A., Jones, T.M., Mullins, R.D.: Mascot: Predicting Memory Dependencies and Opportunities for Speculative Memory Bypassing. In: IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 59–71 (2025)
16. Oleksenko, O., Trach, B., Silberstein, M., Fetzer, C.: SpecFuzz: Bringing Spectre-type vulnerabilities to the surface. In: USENIX Security Symposium. pp. 1481–1498 (2020)
17. Rauscher, F., Gruss, D.: Cross-Core Interrupt Detection: Exploiting User and Virtualized IPIs. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 94–108 (2024)
18. Wiebing, S., de Faveri Tron, A., Bos, H., Giuffrida, C.: InSpectre Gadget: Inspecting the Residual Attack Surface of Cross-privilege Spectre v2. In: USENIX Security Symposium. pp. 577–594 (2024)
19. Wiebing, S., Giuffrida, C.: Training Solo: On the Limitations of Domain Isolation Against Spectre-v2 Attacks. In: IEEE Symposium on Security and Privacy (S&P). pp. 3599–3616 (2025)
20. Zhang, X., Liu, C., Zou, J., Yang, Y., Shen, Q., Zhang, Z., Carlson, T.E.: Towards Practical Interrupt Side Channel Attacks on macOS for Apple Silicon. In: International Symposium on Computer Architecture (ISCA) (2026)
21. Zhang, X., Shen, Q., Zhang, Z., Gao, Y., Zou, J., Yang, Y., Wu, Z.: Fantastic Interrupts and Where to Find Them: Exploiting Non-movable Interrupts on x86. IEEE Transactions on Information Forensics and Security (2025)
22. Zhang, X., Yang, Y., Zou, J., Shen, Q., Zhang, Z., Gao, Y., Wu, Z., Carlson, T.: AmpereBleed: Exploiting On-chip Current Sensors for Circuit-Free Attacks on ARM-FPGA SoCs. In: Design Automation Conference (DAC) (2025)
23. Zhang, X., Zhang, Z., Shen, Q., Wang, W., Gao, Y., Yang, Z., Wu, Z.: ThermalScope: A Practical Interrupt Side Channel Attack Based on Thermal Event Interrupts. In: Design Automation Conference (DAC) (2024)
24. Zhang, X., Zhang, Z., Shen, Q., Wang, W., Gao, Y., Yang, Z., Zhang, J.: SegScope: Probing Fine-grained Interrupts via Architectural Footprints. In: IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 424–438 (2024)