Secure TLBs

Shuwen Deng, Wenjie Xiong, Jakub Szefer
{shuwen.deng,wenjie.xiong,jakub.szefer}@yale.edu
Yale University

ABSTRACT

This paper introduces a new attack vector in modern processors: the timing-based channel attacks due to the Translation Look-aside Buffers (TLBs). This paper first presents a three-step modeling approach to exhaustively enumerate all possible TLB timing-based vulnerabilities and automatically-generated micro security benchmarks that test for the TLB vulnerabilities. After showing the insecurity of standard TLBs, two new secure TLB designs are implemented under RISC-V Rocket Core processor architecture: a Static-Partition (SP) TLB and a Random-Fill (RF) TLB. The proposed secure TLBs are shown to be able to defend against all attacks with a small performance and area overhead.

KEYWORDS

timing side and covert channel attacks, timing attack defenses, TLBs

1 INTRODUCTION

Research on timing-based attacks (and defenses) in processors has a long history. To date, researchers have focused mainly on the memory subsystem when showing the different timing-based attacks, and have, for example, demonstrated a plethora of timing-based channels in caches, e.g. [1, 7]. All the attacks have shown the possibilities to extract sensitive information via the timing-based channels, and often the focus is on extracting cryptographic keys.

Timing-based channels in TLBs, which is the focus of this work, are distinct from caches in that they are triggered by memory translation requests, not by direct accesses to data. They also have a different granularity (pages vs. cache lines for data or instruction caches), and, in commercial processors, TLBs have more complicated logic, compared to caches, due to support for various memory page sizes. Further, defending cache attacks does not protect against TLB attacks [6]. Moreover, there has not been a systematic security analysis of the TLB vulnerabilities, nor concrete proposals for secure TLB design. This paper provides both.

This work starts by providing a novel three-step modeling approach to enumerate all possible TLB timing-based vulnerabilities exhaustively. Rather than modeling software attacks, the three-step approach analyzes all possible victim or attacker behaviors that affect the TLB state. In total, 24 possible vulnerabilities were found, of which only 8 map to existing attacks [6, 8]. We believe that the other 16 are new attack types not previously considered. Based on the three-step model, micro security benchmarks are then semi-automatically generated.

Armed with the three-step model and the security benchmarks, the security of different typical configurations of TLBs are tested using Rocket Core implementation of the RISC-V processor architecture. *Standard TLBs*, i.e. Fully-Associative (FA) or Set-Associative (SA) TLBs, which include process IDs, e.g. ASID in RISC-V architecture, are shown to be vulnerable to many of the attacks. Consequently, this work presents new defenses. Especially, we present two new secure TLB designs in hardware: a Static-Partition (SP) TLB and a Random-Fill (RF) TLB, latter of which is more complex but can defend all of the attacks. These are the first hardware defenses to TLB attacks. To help understand the impact of the new secure TLBs on the system performance, a RISC-V Rocket Core based processor with the new secure TLBs are synthesized on the Zynq ZC706 and ZedBoard FPGAs.

, ,
© 2023
ACM ISBN . . . $

2 BACKGROUND

This section reviews existing work on caches (most closely related to TLBs) and the few existing works on TLBs.

2.1 Timing-Based Attacks and Caches

In modern processor caches, there are timing differences between cache hits (fast) and cache misses (slow), and these variations in timing have been exploited to leak sensitive information. Especially, a large number of different cache timing-based side-channel and covert-channel attacks have been presented in the literature [7, 14]. And, there are many secure hardware cache designs that aim to prevent these different attacks [18, 22] However, even if the cache-based attacks are mitigated, TLB-based attacks are the next attack vector that malicious attackers might use – and hence are the focus of this work.

2.2 Timing-Based Channels in TLBs

Compared with caches, there are two published TLB-based timing attacks [1]. TLBleed attack [6] uses timing-based channels combined with machine learning to create an attack which is able to leak bits of secret keys from the RSA algorithm (they also show attack for the EdDSA algorithm). They leverage the Prime + Probe [13] attack strategy previously applied in processor caches.

Prior to TLBleed, the Double Page Fault attack [8] leverages the Cache Collision [1] attack strategy previously applied in processor caches. It requires the victim to access some kernel memory pages twice, and uses the fact that access to previously allocated kernel virtual pages will bring in TLB entries, even if a page fault is generated and accesses permission checks failed. The timing of the second access thus reveals information on whether an inherent TLB hit happened.

[1]The Leaky Cauldron [17] attack is also related to TLB and targets Intel SGX. However, it does not depend on hits and misses in the TLB, but instead, it relies on the assumption that the attacker can evict the enclave entries in the TLB, so an enclave's memory access will trigger a page table walk, and the malicious OS can get the page access pattern trace.

Beyond these individual attacks, there are neither exhaustive categorizations nor models of possible TLB timing-based attacks – as are proposed in this work.

2.3 Existing Approaches to Securing TLBs

Currently, we are only aware of five approaches (mostly software-based) that can help mitigate some TLB attacks, but are not as effective as our hardware-secure TLBs.

First, today's Linux system makes use of virtual addresses and process identifiers, e.g., ASID on RISC-V, to identify different processes in the SA TLBs to do the partition. Second, in the Sanctum [2] secure processor design, the per-core SA TLBs are flushed by a *security monitor* software whenever a core switches between enclave and non-enclave code. Third, Intel SGX also flushes SA TLBs during switching between enclave and non-enclave code [9]. Fourth, the InvisiSpec [21] work proposes to prevent observable changes to D-TLBs only for speculative attacks. Fifth, some processors employ FA TLBs, which by design do not have different TLB sets (there is only one set).

Unlike all the existing work, this work presents two new hardware secure TLB designs, including the RF TLB which can prevent all types of timing-based attacks according to the three-step model, and has about the same performance as a SA TLB.

3 FRAMEWORK

Our paper's key idea focused on the new attack vector in modern processors: the timing-based channel attacks due to the TLBs. We provide systematic approach to analyze the full set of the vulnerabilities and provide corresponding hardware defenses to mitigate them.

3.1 Modeling TLB Timing-Based Vulnerabilities

We first presented a novel three-step modeling approach that was used to exhaustively enumerate all possible TLB timing-based vulnerabilities.

3.1.1 Threat Model and Assumptions. A TLB timing-based attack involves an attacker and a victim. In many cases, they are executing on the same processor core, a set of cores, or a set of hyper-threads which share the same physical TLB, but this is not required for all types of attacks. In this paper, we use A and V to denote the attacker and the victim with different process IDs. For the attacks where the attacker and the victim are in the same address space, the attacker is able to trigger some known address memory operation as if it were the victim, e.g. states V_a and $V_{a^{alias}}$ in Table 1 can be actually attackers.

We assume, in hardware, all memory operations are identified by the virtual memory address, *vaddr* (including null address in case of certain TLB flush-related operations) and the process ID (including null process ID in case of certain TLB flush-related operations), e.g., ASID in RISC-V.

The victim is assumed to have some security-critical memory range, x, within which the access pattern depends on the secret the attacker wants to learn. An example of a security-critical region is the set of page entries accessed during execution of the RSA functions of *libgcrypt*, where the value of the key bit (either 0 or 1) determines which specific memory pages are accessed. The timing of the accesses to the security-critical memory range is affected by the timing of TLB-related operations, and it can reveal information such as cryptographic keys.

The attacker is assumed to know the victim software, e.g., what implementation of a cryptographic algorithm it uses, but not the secret cryptographic keys. He or she is assumed to know the size, *ssize*, and the location, *sbase* (in virtual memory) of the security-critical memory range x. The attacker can measure the timing of its own memory operations or operations of the victim; but cannot access the actual sensitive data being processed by the victim.

3.1.2 Introduction of the Three-Step Model. One observation we make is that all existing TLB timing-based attacks take three steps. In *Step* 1, a memory operation is performed, placing the TLB block (also called TLB slot or TLB entry) in a known initial state (e.g. a new translation is put into the block or block is invalidated). Then, in *Step* 2, a second memory operation alters the state of the TLB block from the initial state. Finally, in *Step* 3, a final memory operation is performed, and the timing of the final operation reveals some information about the relationship among the addresses from

Table 1: The 10 possible states for a single TLB block in our three-step vulnerability modeling procedure.

States	Description
V_u	The TLB block contains translation for a memory address u, translation which is placed in the TLB block due to a memory access by the victim. Attacker does not know u, but u is from a range x of memory locations, range which is known to the attacker. The address u may have same page index as A_a or V_a and thus conflict with them in the TLB block. The goal of the attacker is to learn the page address or index of V_u.
A_a or V_a	The TLB block contains translation for a memory address a. The translation is placed in the TLB block due to a memory access by the attacker, A_a, or the victim, V_a. The attacker knows the address a, independent of whether the access was by the victim or the attacker themselves. The address a is from within the range of sensitive locations x. The address a may or may not be the same as the address u.
$A_{a^{alias}}$ or $V_{a^{alias}}$	The TLB block contains translation for a memory address a^{alias}. The translation is placed in the TLB block due to a memory access by the attacker, $A_{a^{alias}}$, or the victim, $V_{a^{alias}}$. The address a^{alias} is within the range x. It is not the same as a, but it has same page index and can map to the same TLB block, i.e. it "aliases" to the same block.
A_{inv} or V_{inv}	The TLB block previously containing translation for a memory address is now invalid. The translation is "removed" from the TLB block by the attacker A_{inv} or the victim V_{inv} as the result of TLB block being invalidated, e.g. due to synchronization updates to in-memory memory-management data structures or due to context switch between processes which causes OS to flush per-core TLB entries.
A_d or V_d	The TLB block contains translation for a memory address d. The translation is placed in the TLB block due to a memory access by the attacker, A_d, or the victim, V_d. The address d is not within the range x.
$\star$	Any data, or no data, can be in the TLB block. The attacker has no knowledge of page address in this TLB block.

Table 2: The table shows all the timing-based TLB vulnerabilities. *Attack Strategy* column gives our common name for each set of one or more specific vulnerabilities that would be exploited in an attack in a similar manner (many of the names are borrowed from cache timing-based attacks in literature). *Vulnerability Type* column gives the three steps that define each vulnerability. For *Step* 3, *fast* indicates a TLB hit must be observed, while *slow* indicates a TLB miss must be observed. *Macro Type* column proposes the categorization the vulnerability belongs to. "E" is for external interference vulnerabilities. "I" is for internal interference vulnerabilities. "M" is for miss-based vulnerabilities. "H" is for hit-based vulnerabilities. *Attack* column shows if a type of vulnerability has been previously presented in literature.

Attack Strategy	Vulnerability Type			Macro Type	Attack
	Step 1	*Step* 2	*Step* 3		
TLB Internal Collision	A_{inv}	V_u	V_a (fast)	IH	(1)
	V_{inv}	V_u	V_a (fast)	IH	(1)
	A_d	V_u	V_a (fast)	IH	(1)
	V_d	V_u	V_a (fast)	IH	(1)
	$A_{a_{alias}}$	V_u	V_a (fast)	IH	(1)
	$V_{a_{alias}}$	V_u	V_a (fast)	IH	(1)
TLB Flush + Reload	A_{inv}	V_u	A_a (fast)	EH	**new**
	V_{inv}	V_u	A_a (fast)	EH	**new**
	A_d	V_u	A_a (fast)	EH	**new**
	V_d	V_u	A_a (fast)	EH	**new**
	$A_{a_{alias}}$	V_u	A_a (fast)	EH	**new**
	$V_{a_{alias}}$	V_u	A_a (fast)	EH	**new**
TLB Evict + Time	V_u	A_d	V_u (slow)	EM	**new**
	V_u	A_a	V_u (slow)	EM	**new**
TLB Prime + Probe	A_d	V_u	A_d (slow)	EM	(2)
	A_a	V_u	A_a (slow)	EM	(2)
TLB version of Bernstein's Attack	V_u	V_a	V_u (slow)	IM	**new**
	V_u	V_d	V_u (slow)	IM	**new**
	V_d	V_u	V_d (slow)	IM	**new**
	V_a	V_u	V_a (slow)	IM	**new**
TLB Evict + Probe	V_d	V_u	A_d (slow)	EM	**new**
	V_a	V_u	A_a (slow)	EM	**new**
TLB Prime + Time	A_d	V_u	V_d (slow)	IM	**new**
	A_a	V_u	V_a (slow)	IM	**new**

(1) Double Page Fault attack [8].
(2) TLBleed attack [6].

Step 1, *Step* 2 and *Step* 3. Attacks with more than three steps can be reduced to a three-step attack (details are in our paper [3]). Table 1 lists all the 10 possible states of the TLB block for each step of our three-step model. Each step in the model represents a state of a TLB block.

3.1.3 Derivation of All TLB Vulnerabilities. Based on the states possible in each step there are in total $10 * 10 * 10 = 1000$ combinations of possible three-steps. We developed an algorithm that can process the list of all the three-steps, and eliminates ones which cannot lead to an attack based on a list of derived rules (details mentioned in our paper [3]).

After applying the script which implements our simplification algorithm, 34 three-step access patterns remain as candidates for possible timing-based TLB attacks. These 34 access patterns are further manually reduced to a list of 24 types of timing-based TLB vulnerabilities, listed in Table 2. Due to space limitation, details on why the 10 patterns cannot form vulnerabilities are not included in the paper.

To summarize all the vulnerability types, Table 2 shows the list of all the 24 vulnerability types, along with a more coarse-grained attack strategies, which cover one or more vulnerability types. The list of vulnerability types can be further collected into four simple macro types: internal interference miss-based (IM), internal interference hit-based (IH), external interference miss-based (EM), external interference hit-based (EH). Most of the vulnerability types have not been explored before, except some mapping to existing Double Page Fault attack [8] and the TLBleed attack [6].

3.1.4 Key Contribution. The *key contribution* of this paper was to show the first systematic modeling approach that can be used to reason about all timing-based attacks on TLBs. Our work developed a novel model of the attacker and the victim behavior in relation to the TLB states. Rather than modeling software attacks, the three-step approach analyzed all possible victim or attacker behaviors that affected the TLB states. All possible combinations of the attacker and victim behaviors were evaluated, and systematically reduced to only three-step behaviors that can result in timing-based attacks. In total, 24 possible vulnerabilities were found, including 16 new attack types not previously considered.

3.2 Micro Security benchmarks

Building on the three-step model, our paper then showed how to automatically generate micro security benchmarks to test TLBs to check if they are vulnerable to each of the attack types. To generate the micro security benchmarks, we leverage a Python script that follows a three-step template to generate assembly code of all the types of vulnerabilities showed in Table 2.

We use channel capacity [5] to quantify the amount of information about the secret address translation that the attack gains from a specific timing-based attack.

3.3 Secure TLB Designs

After showing the insecurity of standard TLBs, our work also proposed the first hardware defenses for TLB attacks: the new SP TLB and the new RF TLB, and realize them in a Rocket Core implementation of a RISC-V processor. Especially, RF TLB was more complex in logic but could defend all of the attacks comparing to SP TLB.

The first type of TLB, SP TLB is a SA TLB where certain ways are assigned to a victim process and other ways are assigned to all remaining processes, which by default are assumed to be potential attacker processes. The process ID, e.g. ASID in RISC-V, is used to differentiate the victim and the attacker. The number of ways assigned to each is set at design time, but could be further extended to be dynamic at run time.

To protect all the vulnerabilities, we propose Random-Fill TLB, which is able to de-correlate the requested memory access from actual TLB entries that are brought into the TLB, making the attacker's observations non-deterministic. For TLB hits, the behavior is the same as the SA TLB. For TLB misses, depending on the memory address region, a random address translation will be fetched into the TLB ("random fill"), while the originally requested address is directly sent back to the CPU without filling the TLB ("no fill").

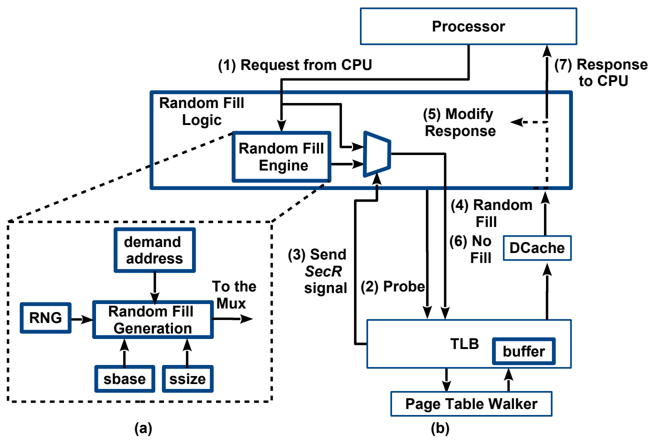

Figure 1: (a) Random Fill Engine, (b) RF TLB block diagram.

The RF TLB also introduces the *Sec* bit which is used to identify certain memory translation entries are belonging to secure data.

RF TLB block diagram is shown in Figure 1b. All the bold lines and blocks are the added hardware and logic extension. In the TLB array, an extra field (a secure bit *Sec*, either 0 or 1) is added to each of the TLB entries to indicate whether it contains an address translation within the secure region. In addition, the existing process ID field (e.g., ASID in RISC-V) in each TLB entry is used to differentiate the victim and the attacker process. By default, we set specific process ID 1 for the victim program and all other ASIDs to be attackers.

An extra set of registers is added to store the process ID of the victim process and the start address, *sbase*, and the size, *ssize* of the secure region (the base and size are defined in terms of pages, usually 4KiB). The registers can be managed by a trusted OS to change the victim process ID and secure regions when different victim programs need protection.

An extra *buffer* is added which stores equivalent of one TLB entry. It is used as temporary storage for translation data that is returned to the CPU, but which should not be placed in the TLB. It will be cleaned up after the address is returned.

The Random Fill Engine (RFE), shown in Figure 1a is used to generate addresses which should be used for TLB updates[2]. In Figure 1b, (1-2), the "no fill" *fill_type* will first be sent to TLB. On a TLB miss, the TLB will *probe* the page address without filling TLB entries to see if the chosen entry has a valid secure page address translation. Then, (3) the Sec_R bit is set and sent back. Next, (4) if it is a request to the secure region or the Sec_R bit is one, a random fill request will be triggered. If the original request is in the secure region, a random virtual page address is derived from RFE within the secure region [*sbase*, *sbase* + *ssize*], and a translation will be put into the TLB entry. If the original request comes from the non-secure region, most of the higher bits of the requested address are remain the same while the bits that correspond to the TLB set index[3] will be randomized to make the eviction indeterministic. Next, (5) the *Random Fill Logic* will modify the response and prevent the random fill result from being sent to the processor. Then, (6) the original page address is finally requested, and "no fill" *fill_type* will be sent to the TLB to obtain the translation. Finally, (7) this address will be stored in the *buffer*, without modifying TLB entries, and be sent back to the processor.

Table 3: Comparison of SA TLB, SP TLB and RF TLB simulation and theoretical results. C^* and C represent mutual information based on simulation and theoretical calculation, respectively. Bold C^* and C are the ones with value 0 or about 0, indicating that this TLB is able to prevent the corresponding vulnerability. Small numbers are rounded up. Some vulnerability types are not shown to save space.

		SA TLB		SP TLB		RF TLB	
Attack Category	Vulnerability Type	C^*	C	C^*	C	C^*	C
TLB Evict+Probe	$V_d \rightsquigarrow V_u \rightsquigarrow A_d$ (slow)	**0**	**0**	**0**	**0**	**0**	**0**
TLB Prime+Time	$A_d \rightsquigarrow V_u \rightsquigarrow V_d$ (slow)	**0**	**0**	**0**	**0**	**0**	**0**
TLB Flush+ Reload	$A_d \rightsquigarrow V_u \rightsquigarrow A_a$ (fast)	**0**	**0**	**0**	**0**	**0**	**0**
TLB Prime+Probe	$A_d \rightsquigarrow V_u \rightsquigarrow A_d$ (slow)	0.99	1	**0.02**	**0**	**0.01**	**0**
TLB Evict+Time	$V_u \rightsquigarrow A_d \rightsquigarrow V_u$ (slow)	1	1	**0.03**	**0**	**0**	**0**
TLB Internal Collision	$A_d \rightsquigarrow V_u \rightsquigarrow V_a$ (fast)	1	1	0.98	1	**0.01**	**0**
TLB Bernstein's Attack	$V_u \rightsquigarrow V_a \rightsquigarrow V_u$ (slow)	0.99	1	0.99	1	**0.01**	**0**

3.4 Security Evaluation

The three-step model and the security benchmarks were used to analyze the security of the new designs in simulation. Based on the analysis, we showed the proposed secure TLBs could defend not only against the previously publicized attacks, but also against other new timing-based attacks in TLBs found using our new three-step modeling approach.

As demonstrated in Table 3, while evaluating the security of the standard and secure TLBs using the micro security benchmarks running on RISC-V simulation, we showed that results matched with theoretical mutual information calculation. For the specific security effectiveness of different TLBs, our security evaluation showed that standard SA TLBs could defend 10 types of external hit-based related attacks. For the new hardware defenses, our evaluation showed that SP TLB was able to further prevent 4 more external miss-based vulnerabilities, in total defending 14 out of 24 vulnerabilities. Meanwhile, the RF TLB was able to prevent all of the 24 possible timing-based vulnerabilities in TLBs. The proposed secure TLBs could defend not only against the previously publicized attacks, but also other possible timing-based attacks in TLBs found using our new three-step modeling approach.

3.5 Performance Evaluation

Finally, we were able to maintain the performance overhead to a small number while protecting the fully security. We tested performance by synthesizing the hardware on FPGAs, and running RSA decryption tests alongside SPEC 2006 benchmarks under Linux on the FPGAs. To help understand the impact of the new secure TLBs on the system performance, a RISC-V Rocket Core based processor

[2]We assume the OS has pre-generated page table entries that may correspond to the random virtual address generated by the RFE, which may not be actually used by the original program, to prevent OS or software-based timing attacks due to page faults when a page entry for a random address is looked up by the TLB.

[3]The TLB set index to be randomized has bit size $S_n = log_2[min(ssize, nsets)]$, where $nsets$ is the number of sets in TLB. A random set index will be generated within the region $[sbase[S_n - 1, 0], sbase[S_n - 1, 0] + min(ssize, nsets)]$ for random fill.

with the new secure TLBs were synthesized on the Zynq ZC706 and ZedBoard FPGAs. This allowed for running security software alongside SPEC 2006 benchmarks and a full Linux system. Based on our evaluation, for example, the SP TLB had 3x misses per kilo-instructions (MPKI) compared to the standard SA TLB, while the RF TLB had 9% more MPKI than the standard TLB. For the RF TLB, the hardware cost of the defenses was about 8% more logic.

4 DISCUSSION AND FUTURE WORK

Research on timing-based attacks (and defenses) in processors has a long history. Most mitigations of timing-based attacks in the memory subsystem have focused on the design of secure caches. Meanwhile, our work focused on preventing timing-based attacks due to TLBs and presented the first hardware defenses for TLBs. In our securing TLBs work, both the systematic approach of finding all possible TLB timing side-channel vulnerabilities and the hardware defenses of TLBs regarding timing side-channel vulnerabilities were presented as a promising methodology and solutions, respectively, to examine and tackle TLB timing side channels. Especially, our hardware defenses were tested in real hardware (FPGA), while much of the existing work is evaluated in simulation only. Therefore, our work can give more confidence and possibly be more easily adapted to be used in industrial and commercial products. In addition, although it has been only two years since this work's publication, it has already been cited by 14 different research groups and projects as well as exploited ideas introduced in further papers, such as [4, 12, 16, 19, 20, 23]. The methodology and solution presented in our work have thus already influenced other research projects on timing side-channels and TLBs.

Firstly, the three-step modeling method for TLBs inspired our prior work [4] which focused on timing-based vulnerabilities in caches. Thus the three-step modeling method demonstrated in TLBs (and previously in caches) is emerging as a promising method for analyzing cache-like structures in processors. Further, the TLB (and caches) work checked the soundness of the three-step model, and show that it is possibly a generic approach to other timing-channel analyses.

Our hardware defenses for TLBs have further been shown to be effective even with newly-developed attacks. For example, PThammer [23] developed new attacks focusing on eviction-based cache and TLB attacks. It exploited the fact that cache and TLB were shared between sensitive data and non-sensitive data. It also confirmed our hardware defenses are effective and can readily mitigate PThammer by partitioning or randomizing the TLB. Without the use of our secure TLBs, the PThammer could easily compromise TLBs and lead to information leaks in real systems.

Meanwhile, our TLB work has also been shown in work [20] to be able to prevent transient execution attacks. The recent Spectre [10] and Meltdown [11] attacks have shown that timing-based channels are more dangerous than previously thought. Whether by themselves or in combination with speculative execution such as the Spectre and Meltdown attacks, the timing-based channels in microarchitecture pose threats to system security, and should be mitigated. Our work proposed hardware defenses to mitigate timing-based channels in TLBs, therefore, it is effective even for the new transient execution attacks which utilize TLB microarchitecture timing-based channel.

Our hardware defense ideas have also been expanded by others. SMT-COP [16] extended the processor defenses for Simultaneous MultiThreading (SMT). SMT-COP provided a system that eliminated all known side-channels through shared execution logic, including ports and functional units, on SMT processors. This work helped resolve execution logic side-channels in SMT architectures and extends the TLB timing side-channel defenses more focusing on the SMT side.

Research based on our work has clearly acknowledged the effectiveness of our modeling approach, and the hardware defense methods for the TLB timing side-channel vulnerabilities [12, 15, 19]. We believe that our work on securing the TLBs from the timing-based channels has already created impact and will continue to serve as a catalyst for higher security, especially in future computer architectures.

5 CONCLUSION

This paper proposed a novel three-step modeling approach that exhaustively enumerates all possible TLB timing-based vulnerabilities. It showed how to automatically generate micro security benchmarks that test for the TLB vulnerabilities. It gave details of two new hardware secure TLB designs: a Static-Partition (SP) TLB and a Random-Fill (RF) TLB. The simulations confirmed the theoretical channel capacity calculations and full system performance on FPGA showed that the new secure TLBs are as good as regular TLBs, while protecting against the various attacks. The proposed secure TLBs can defend not only against the previously publicized attacks, but also other possible timing-based attacks in TLBs found using our new three-step modeling approach.

REFERENCES

[1] Joseph Bonneau and Ilya Mironov. 2006. Cache-Collision Timing Attacks against AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 201–215.

[2] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*. 857–874.

[3] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2019. Secure tlbs. In *Proceedings of the 46th International Symposium on Computer Architecture*. 346–359.

[4] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2020. 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. https://doi.org/10.1145/3373376.3378510

[5] Andrea J Goldsmith and Pravin P Varaiya. 1997. Capacity of Fading Channels with Channel Side Information. *IEEE Transactions on Information Theory* 43, 6 (1997), 1986–1992.

[6] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks. In *USENIX Security Symposium*. USENIX, 955–972.

[7] David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache Games–Bringing Access-Based Cache Attacks on AES to Practice. In *IEEE Symposium on Security and Privacy*. IEEE, 490–505.

[8] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical Timing Side Channel Attacks Against Kernel Space ASLR. In *IEEE Symposium on Security and Privacy*. IEEE, 191–205.

[9] Intel Intel. 64. IA-32 Architectures Software Developer's Manual. *Volume 3A: System Programming Guide, Part* 1, 64 (64), 64.

[10] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *ArXiv e-prints* (Jan. 2018). arXiv:1801.01203.

[11] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *ArXiv e-prints* (Jan. 2018). arXiv:1801.01207

[12] Ajay Nayak, Vinod Ganapathy, and Arkaprava Basu. 2021. (Mis) managed: A Novel TLB-based Covert Channel on GPUs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 872–885.

[13] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: the Case of AES. In *Cryptographers' Track at the RSA Conference*. Springer, 1–20.

[14] Colin Percival. 2005. Cache Missing for Fun and Profit.

[15] Dimitrios Skarlatos, Zirui Neil Zhao, Riccardo Paccagnella, Christopher W Fletcher, and Josep Torrellas. 2021. Jamais vu: thwarting microarchitectural replay attacks. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 1061–1076.

[16] Daniel Townley and Dmitry Ponomarev. 2019. SMT-COP: Defeating Side-Channel Attacks on Execution Units in SMT Processors. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE. https://doi.org/10.1109/pact.2019.00012

[17] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *Conference on Computer and Communications Security*. ACM, 2421–2434.

[18] Zhenghong Wang and Ruby B Lee. 2007. New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks. In *ACM SIGARCH Computer Architecture News*, Vol. 35. ACM, 494–505.

[19] Xiaohui Wu, Yeping He, Qiming Zhou, Hengtai Ma, Liang He, Wenhao Wang, and Liheng Chen. 2020. Partial-SMT: Core-scheduling Protection Against SMT Contention-based Attacks. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 378–385.

[20] Wenjie Xiong and Jakub Szefer. 2020. Survey of Transient Execution Attacks. arXiv:arXiv:2005.13435

[21] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher W Fletcher, and Josep Torrellas. 2018. InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy. In *International Symposium on Microarchitecture (MICRO)*. IEEE, 428–441.

[22] Danfeng Zhang, Yao Wang, G Edward Suh, and Andrew C Myers. 2015. A Hardware Design Language for Timing-Sensitive Information-Flow Security. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 503–516.

[23] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. 2020. PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses. arXiv:arXiv:2007.08707