Hindawi
Security and Communication Networks
Volume 2021, Article ID 5559552, 15 pages
https://doi.org/10.1155/2021/5559552

Review Article

Survey of CPU Cache-Based Side-Channel Attacks: Systematic Analysis, Security Models, and Countermeasures

Chao Su [1,2] and Qingkai Zeng [1,2]

[1]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[2]*Department of Computer Science and Technology, Nanjing 210023, China*

Correspondence should be addressed to Qingkai Zeng; zqk@nju.edu.cn

Received 25 February 2021; Revised 14 May 2021; Accepted 31 May 2021; Published 11 June 2021

Academic Editor: Petros Nicopolitidis

Privacy protection is an essential part of information security. The use of shared resources demands more privacy and security protection, especially in cloud computing environments. Side-channel attacks based on CPU cache utilize shared CPU caches within the same physical device to compromise the system's privacy (encryption keys, program status, etc.). Information is leaked through channels that are not intended to transmit information, jeopardizing system security. These attacks have the characteristics of both high concealment and high risk. Despite the improvement in architecture, which makes it more difficult to launch system intrusion and privacy leakage through traditional methods, side-channel attacks ignore those defenses because of the shared hardware. Difficult to be detected, they are much more dangerous in modern computer systems. Although some researchers focus on the survey of side-channel attacks, their study is limited to cryptographic modules such as Elliptic Curve Cryptosystems. All the discussions are based on real-world applications (e.g., Curve25519), and there is no systematic analysis for the related attack and security model. Firstly, this paper compares different types of cache-based side-channel attacks. Based on the comparison, a security model is proposed. The model describes the attacks from four key aspects, namely, vulnerability, cache type, pattern, and range. Through reviewing the corresponding defense methods, it reveals from which perspective defense strategies are effective for side-channel attacks. Finally, the challenges and research trends of CPU cache-based side-channel attacks in both attacking and defending are explored. The systematic analysis of CPU cache-based side-channel attacks highlights the fact that these attacks are more dangerous than expected. We believe our survey would draw developers' attention to side-channel attacks and help to reduce the attack surface in the future.

1. Introduction

With the development of modern computer systems, more and more attention is paid to the protection of security and privacy. However, CPU cache-based side-channel attacks have become a serious threat nowadays. Residing in the same hardware resources, the attacker is able to steal sensitive information from the victim program secretly. It collects temporal information about the behaviors in the victim program and then uses it to leak execution logic or important data.

Cloud computing services use virtualized resources to provide various services. The sharing hardware resources in the cloud make the program more vulnerable to side-channel attacks. For example, different services are run on the same physical CPU, making it possible to detect the CPU cache states. By observing the changes in CPU cache states, attackers infer the memory usage of the victim program and thus leak sensitive information. Since in CPU cache-based side-channel attacks attackers do not read the key information directly, traditional intrusion detections cannot respond effectively.

It has been decades since the side-channel attacks based on CPU cache were proposed [1]. Yet the research studies on this topic are still booming. According to the statistics from Web of Science, the number of research papers in recent years has quadrupled since 2010. Figure 1 shows the fact that every year over 600 papers are accepted or published by

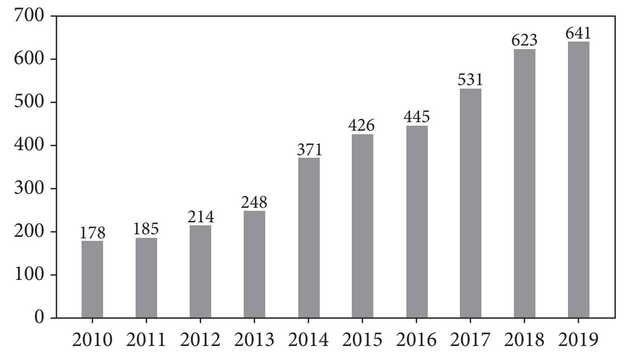

Figure 1: Number of research papers on side-channel attacks from 2012 to 2019 from Web of Science.

high-quality conferences and journals. Researchers propose new types of CPU cache-based side-channel attacks. By optimizing the interfering patterns between the attacker and the victim program, attackers can manipulate the cache in various ways. The emergence of these patterns (prime and probe, flush and reload, etc.) allows attackers and victims to access the cache in different orders, reducing the difficulty of launching an attack. Besides, attackers try to steal information across different domains. In the beginning, the attacker and the victim need to stay in the same process (as different threads) [2]. With the evolution of the attack, sensitive information can be stolen across VMs. Now attackers can even bypass Intel SGX and steal the secret from enclaves [3]. In addition, more common programs are suffering from the attack. CPU cache-based side-channel attacks originally targeted the RSA modules of OpenSSL. Researchers extend the application and find more programs vulnerable to the CPU cache-based side-channel attack. AES [4, 5], DSA [6], ECC [7], and other implementations of cryptographic modules in OpenSSL are now proved that they can be compromised by the attack. Attackers can even leak the kernel memory [8] and steal user activities [9].

Due to the significance of the side-channel attacks, there are various kinds of surveys focusing on them. However, these surveys lack a comprehensive understanding of the attack. Some research studies [10] only provided a taxonomy about the CPU cache-based side-channel attacks. Without building any models for the attacks, they missed the analysis of the attack procedure and thus lack understanding of them. They also listed the part of the defense strategies. Without thinking deeply into the conditions of the CPU cache-based side-channel attacks, they did not explain why these strategies could defend the attacks. Some research studies [11] pay more attention to attacks with other side channels, for instance, side channels based on power consumption. Furthermore, the target programs of these studies are limited to the implementation of cryptographic modules. In fact, CPU cache-based side-channel attacks can also be used in normal programs [12, 13].

To have a comprehensive understanding of CPU cache-based side-channel attacks and the current research status, this paper discusses the recent research studies of these attacks. By establishing an attack model with four factors, we extract the necessary conditions for generating security threats. We discuss different defense strategies and analyze how they mitigate the attacks. The main contributions of this paper are as follows:

(1) The paper concludes the general workflow of the attack by comparing existing attacks. We compare the methods and impacts of different side-channel attacks based on CPU cache and summarize the premises and general procedures of such attacks. It extends the attack scenarios and reveals how sensitive data is leaked in side-channel attacks.

(2) This paper establishes an attack model with four factors, namely, vulnerability, cache type, pattern, and range. This model can be used to describe various attacks. It indicates how these attacks evolve. Through this model, we extract the necessary conditions for generating security threats.

(3) It conducts a comprehensive analysis of different defense strategies. Through these strategies, we analyze from which aspect the defense strategies fail the attacks and provide better security. It verifies the necessary conditions and inspires a direction for future research studies on mitigating the influences of CPU cache-based side-channel attacks.

The remainder of this paper is organized as follows. Section 2 first introduces the CPU cache, followed by the current status of side-channel attacks based on CPU cache in Section 3. Section 4 establishes a model for such attacks. This model extracts factors and the necessary conditions for generating security threats. Then, in Section 5, we explore and discuss the characteristics of defending strategies by comparing the model with existing defenses. We discuss the challenges and solutions for future research, indicating trends of research studies on CPU cache-based attacks in Section 6. Finally, the conclusion of the paper is given in Section 7.

2. CPU Cache

A CPU cache is a type of storage device with a multilayer structure located between CPU and DRAM. Its capacity is smaller, while reading and writing speed is much faster than memory. CPU cache alleviates the contradiction between high-speed processors and low-speed memory. When the CPU is calling for data, it first queries whether the data is in the cache. If the content in the memory has been loaded in the cache, it can be read directly from the cache, which means a cache hit. Otherwise, the content needs to be loaded into cache before it is used by the CPU, meaning a cache miss. According to the principle of temporal and spatial locality, the content in this cache will be accessed again with a high probability, reducing the time the CPU waits for reading and writing data from the memory.

A modern CPU cache is designed with a hierarchy architecture. Usually, CPU has three layers of cache, named L1 cache, L2 cache, and L3 cache. There are differences in access speed and capacity between different layers. The higher-level cache (e.g., L1 cache) provides swifter access, while the lower-level cache (e.g., L3 cache) provides larger capacity. Table 1 shows part of the capacities and access latencies of different layers of cache from various CPUs. The access latency changes caused by various levels of cache are the basis of side-channel attacks. In the following sections, all side-channels refer to those based on CPU cache. Other side channels based on factors such as energy consumption or electromagnetic performance are out of our scope.

When the CPU tries to give access to data in the cache, it will first search in the L1 cache. If it cannot find it, it will switch to the L2 cache and so on. The hierarchy inheritance structure of the CPU cache has such a characteristic that if a certain data exists in the high-level cache (such as L1 cache), it must be found in the lower-level cache (such as L3 cache). The inclusiveness of the CPU cache is defined as follows: m denotes a piece of memory data. $L1$, $L2$, and $L3$ denote the contents in the L1 cache, L2 cache, and L3 cache. Then,

$$m \in \mathrm{L1} \longrightarrow m \in \mathrm{L2} \longrightarrow m \in \mathrm{L3}. \quad (1)$$

The inclusiveness of the CPU cache also ensures that the eviction in the L3 cache leads to the eviction in L2 and L1 cache, which means

$$m \notin \mathrm{L3} \longrightarrow m \notin \mathrm{L2} \longrightarrow m \notin \mathrm{L1}. \quad (2)$$

The inclusiveness ensures the consistency of the cache states in different levels.

3. Side-Channel Attacks

Kocher [1] first proposed the idea of timing attacks in 1996. By 2005, Colin Percival [14] first noticed the fragility of the CPU cache in the face of side channels against time. He used symmetric multithreading (SMT) and L1 data cache to attack the RSA in OpenSSL. It opened up the prologue of the CPU cache-based side-channel attack. Ristenpart [15] launched a cache-based side-channel attack across virtual machines, which brought a huge threat to cloud security. Irazoqui [4] and Zhang et al. [9] proposed cross-core cache side-channel attacks in 2014 and 2015, further expanding the scope of the attack. In 2016, researchers even completed a cache-based side-channel attack that works between different processors [16]. Even if the programs are not running on the same CPU, an attacker can leak sensitive information through cache because of cache coherency. Meltdown [13] and Spectre [12] shocked the security community and received widespread attention from the industry and academia. They come from flaws in the architecture design and are not caused by software or systems. Almost all modern CPUs have been affected. These two vulnerabilities and follow-up studies [17] rely on cache side-channels to help complete the attack. In recent years, Intel deployed SGX technology in their CPUs to help users to build a minimal trusted computing base. It allows users to put sensitive data and codes of the program in a secure container called enclave [18], providing confidentiality and integrity protection. Although SGX technology was once considered the future of trusted execution environments, it fails to solve cache side-channel attacks. Although enclave has advantages in defending against Flush + Reload cache side-channel attacks, Götzfried J [19] designed a Prime + Probe attack scheme to identify the memory location accessed by the enclave code, thereby leaking sensitive data.

The changing of CPU cache-based side-channel attacks reveals its evolution. The conditions of the attack are becoming looser. The scope of the attack is becoming wider. So the ability of the attack has been improved. Although the hardware vendors and software developers have raised the attention to security issues, the attack surface of cache side-channel attacks is becoming larger and larger, and the attack methods have become more complex.

3.1. Attack Workflow.

To cause information leakage, side-channel attacks need to complete the following four steps.

3.1.1. Define the Connection between the Victim Program and the Attacker Program.

The first step to launch a side-channel attack is to search for available channels. The connection between the victim and the attacker program can indicate the carrier of the channel. In CPU cache-based side-channel attacks, the carrier of the channel is the CPU cache, which means it is necessary to search for the correlation between the attacker program and the victim program in the cache. For example, the research by Eckert et al. [19] takes advantage of a shared library (OpenSSL 0.9.8n) that both attacker and victim call. In completely fair scheduling settings, they occupy the exact same cache. Large pages also introduce connections. It is common for virtualization applications such as VMware and Xen to deploy large pages to manage physical memory in guest virtual machines [20]. In this case, attackers utilize the large page mechanism to establish a connection between the virtual page and the physical page. This connection allows attacks to snoop other processes' piracy data through cache.

3.1.2. Collect the Activities in the Cache of the Attacker's Program While It Is Running.

According to the connection between the victim and the attack program, the attacker will use an appropriate memory read and write mode to detect its own cache states. At this stage, the attacker usually pre-set the state of the cache. For example, by continuous memory reading or writing, attackers ensure their target memory is loaded in cache. They also use `CLFLUSH` instruction or other methods to make sure the content is evicted out of cache. When the victim process is being executed, attackers will access memory again multiple times in a row. The state of the cache can be recorded by the access delay. This step is performed at the same time as the victim program.

3.1.3. Speculate on the Cache Changes of the Victim Program.

Due to the connection between the victim and the attacker program described in the first step, the cache behaviors of the victim program can be inferred from the cache states recorded by the attacker from step 2. There are usually two types of connections between cache states from the victim

Table 1: Cache capacity and latency.

Model	Microarchitecture	L1 size (K)	L1 latency	L2 size	L2 latency	L3 size (M)	L3 latency
i7-6900K	Bradwell	64	4	256	12	20	59
i7-6700	Skylake	64	4	256	12	8	42
i7-4770	Haswell	64	4	256	12	8	36
i7-3770	Ivy Bridge	64	4	256	12	8	29.9
E5-2699	Bradwell	64	4	256	12	55	65
i3-2120	Sandy Bridge	64	4	256	12	3	27.9

and attacker processes: consistency and exclusion. The consistency connection means victim and attacker processes share the same cache states (hit or miss), which widely exist in side-channel attacks based on shared libraries [5, 6, 16, 20–23]. It allows the content that is out of the cache to be loaded by its rival, and thus, it can spy on sensitive information. Exclusion refers to the exclusive use of the cache mutually by both victim and attacker processes. When one of them tries to occupy the cache, it evicts rival's content out of cache first, causing changes in the cache states.

3.1.4. Infer the Sensitive Information of the Victim's Program. There is a connection between the cache state of the victim program and sensitive information. In caching side-channel attacks, it is necessary to perform a priori analysis of the victim's program so that attackers are able to define the association between state changes and sensitive information of the victim.

For example, the basic operation unit in the encryption or decryption of RSA is $x = y^e \mathrm{mod} n$. To perform fast calculation, the RSA module of OpenSSL 0.9.7c uses the square-and-multiply algorithm. It squares $y \log_2 e$ times and then performs $\log_2 e/2$ additional multiplications by y. After that, the product is reduced by modulo n. OpenSSL optimizes the algorithm with siding windows' exponentiation. It calls the multiply function and square function in BIGNUM according to e and then uses Montgomery reduction. In this process, the numbers of square function calls and multiplication function calls are closely related to the sensitive information e, which is the key. Whether the functions are called or not indicates the bits in e, and the number of function calls indicates the number of all the bits.

When the attacker knows the status change of the victim's program, the attacker can speculate on its sensitive information and eventually cause the final information leakage. The complete attack process is explained in Figure 2.

In step 1, attackers check if the attacker program and victim program co-reside in the same system. The co-residence ensures that there is a connection between them, and thus, they can use the same cache.

In step 2, define s as the sensitive information in the victim program. When the victim program is executed, specific state changes (defined as p) displayed on the cache are related to the sensitive information s. That is, there is a mapping:

$$p = f(s). \tag{3}$$

In step 3, according to the correlation analysis of the victim program and the attacker program, define the relevance (marked as g) of the attacker's cache state (marked as q), which means

$$q = g(p). \tag{4}$$

In step 4, the process of restoring sensitive information in a side-channel attack can be expressed as

$$s = f^{-1}\left(g^{-1}(q)\right). \tag{5}$$

These 4 steps represent the workflow of revealing sensitive information s through cache-based side-channel attacks.

3.2. Threats of the Attacks. The harm of cache side-channel attacks is that the attacker uses this channel to secretly transmit sensitive information, posing threats to the system. The threat of cache side-channel attacks is mainly listed in the following aspects:

(a) Disclosure of sensitive information such as privacy
(b) Deliver the results of malicious code execution
(c) Denial of service

CPU cache-based side-channel attacks can launch attacks across different domains (different processes, different VMs, different platforms, different CPUs, etc.). Its most widely used application is to leak sensitive information of the target program. Sensitive information includes the key in the encryption and decryption algorithm, the crucial data calculated by the program, the user's behavior, or the memory layout. Due to the unpredictability of the cache state, this type of sensitive information leakage is difficult to capture through pattern or feature detection, so it is concealed.

Cache side-channel attacks often work to serve other malicious operations. Attackers pass the result of the previous malicious behavior through it. ExSpectre [17] is a typical example of using a side channel to deliver the results of malicious code execution. As shown in Figure 3, the attacker first trains the branch predictor through trigger codes. Next time, the program comes to the same branch; it will take the same branch in predicting execution. The red arrow is the execution of the misprediction. There are code gadgets of malicious behavior hidden in this branch.

When the CPU realizes the misprediction, it rolls back and executes the actual branch. However, the malicious instructions have been executed, and the execution result of the instruction is delivered to the attacker through a side-channel attack on a predefined array. The rollback does not eliminate the influence on the cache.

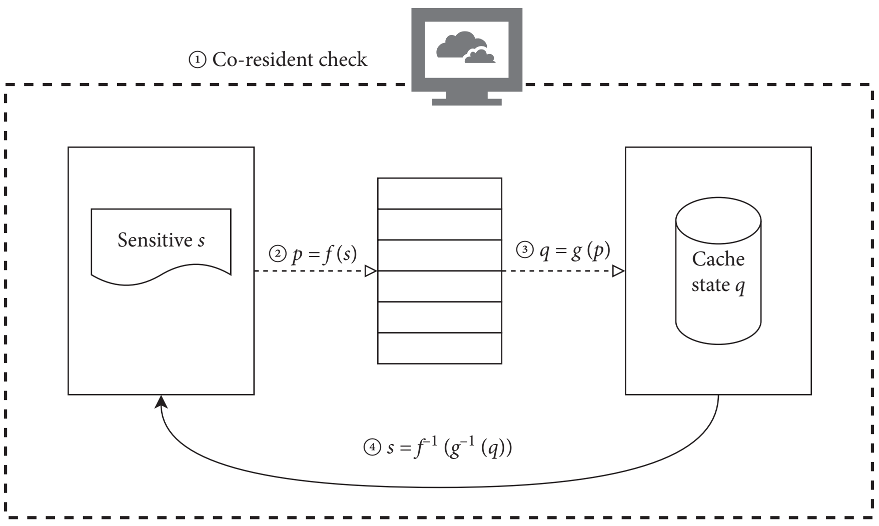

Figure 2: Workflow of cache-based side-channel attacks.

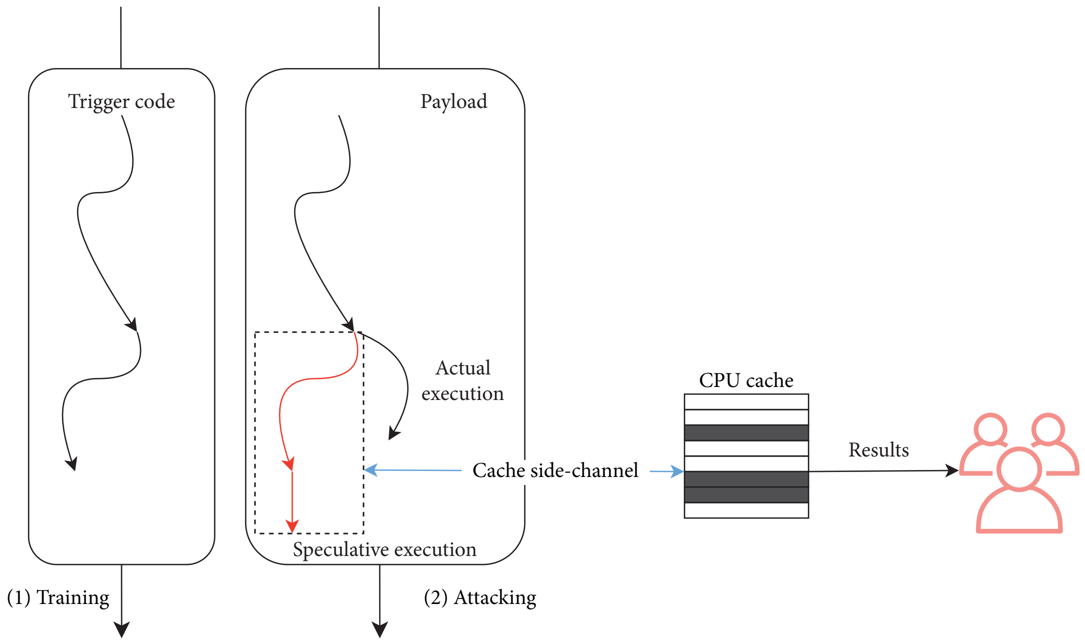

Figure 3: Leak malicious results of ExSpectre through cache-based side-channel attack.

Since predictive execution will roll back the result after finding that the wrong branch is selected, it is impossible to discover the malicious behavior being executed by code tracing or dynamic analysis. In this case, the threat of side-channel attacks lies in the delivery of malicious code running results.

Since low-level caches (such as LLC) are often shared by multiple cores, malicious programs can also use cache side-channel attacks to make a malicious program keep preempting the cache so that other programs on the core cannot use the cache and memory normally. Preempting the shared cache will reduce the performance of the victim's program, causing a denial of service. Research studies [24–26] show that this type of attack can cause a performance reduction by 95% and overhead increase by 7.9X.

4. Analysis Model of the Side-Channel Attacks

We define the model of cache side-channel attacks as

$$\begin{aligned}
&\text{CACHE\_SIDE\_CHANNEL\_ATK:}\\
&\langle \text{vulne}, \text{type}, \text{pattern}, \text{range}\rangle |\\
&\text{vulne} \in \text{LEAK\_CODE};\\
&\text{type} \in \text{CACHE\_TYPE};\\
&\text{pattern} \in \text{CACHE\_ACT};\\
&\text{range} \in \{\text{core}, \text{package}, \text{NUMA}, \text{system}\}.
\end{aligned}$$

Among them, *LEAK_CODE* refers to the vulnerable code inside the victim program that can be compromised. That is the basis for information leakage. For instance, CVE-2018-0737 [27], CVE-2017-5754 [28], and CVE-2016-2178

[29] expose this type of codes. Vulnerabilities may contain the leakage of executing times of different branches or memory access patterns, etc. CACHE_TYPE describes the type of cache used by side-channel attacks, which are the carrier of the channel. They can be L1-I, L1-D caches, LLC caches, or any available CPU cache. With the deepening of research, attackers tend to use more general cache than specialized cache, which widens the attack range and brings more and more threats. CACHE_ACT defines the connection between the state change caused by the attacker and the victim processes, including mutual exclusion and consistency, as mentioned above. According to CACHE_ACT, the attacker organizes the order and pattern for the victim and the attacker to use the cache and times the operations for probing. The model is illustrated in Figure 4.

According to the co-resident configuration of the attacker program and the victim program, a side-channel attack has a threat range, that is, the upper and lower bounds of the leaked confidence. range defines the range of sensitive information that is leaked, that is, the source and destination of the information leakage.

Table 2 lists some of the research studies about side-channel attacks indicating their development. Through comparison, the differences in cache side-channel attacks are mainly presented in the following aspects: the vulnerabilities the attack takes, the type of cache, the pattern of probing the cache state, and the collocation range that the attacker and the victim co-reside in. These are key elements that constitute the cache side-channel attack model.

4.1. Program Vulnerability. Vulnerabilities of the programs attacked by side-channel attacks result from the failure to cut the connection between sensitive information and cache state changes. In CVE-2018-0737 [27], OpenSSL calls `BN_mod_inverse()` function and `BN_mod_exp_mont()` function without setting the `BN_FLG_CONSTTIME` flag during Montgomery arithmetic setup and modular exponentiation. The resulting code path is not constant time and eventually leaks the critical GCD state and critical exponentiation state. The patch codes, shown in Figure 5, add `BN_FLG_CONSTTIME` flag, making it a constant execution time in the face of different inputs. This cuts off the connection between sensitive information and the cache state so that cache side-channel attacks cannot leak corresponding information through cache.

4.2. Cache Type. CACHE_TYPE defines the types of CPU cache used by the attacks. There are three main types of caches utilized in CPU cache-based side-channel attacks: L1-I, L2-D, and LLC, namely, CACHE_TYPE = $\{L1-I, L1-D, LLC\}$.

L1-I: the L1 cache is the fastest part of the cache. Its capacity and speed have a great impact on the performance of the CPU. Therefore, the cache is designed to be divided into the data cache and instruction cache. L1-I is the instruction cache, which is specially used to store instructions in the first-level cache. Whether the content (code) is in cache or not indicates whether it has been recently accessed. In side-channel attacks, the attackers usually use L1-I to infer the execution path. Research studies [6, 21] used L1-I to launch side-channel attacks on RSA for sensitive information.

L1-D: L1-D is the part of the first-level cache that stores data. Unlike L1-I, cache states of L1-D reflect the access to data in memory. Although it is unlikely to read the specific values of the data stored in these caches, the data structure of the victim's program and the access pattern of variables can also be utilized to infer sensitive data. Researches [2, 5, 14, 23] used L1-D cache as the carrier of side-channel attacks to compromise OpenSSL's RSA and AES algorithms and leak encryption and decryption keys successfully.

LLC: last-level cache is the part of the CPU cache close to memory. As shown in Table 1, compared to L1-I and L2-D caches, the LLC cache has a larger access latency. Therefore, cache side-channel attacks have better robustness when using LLC as their carrier. In modern CPUs with three-layer caches, the L3 cache is the LLC. In normal cases, the L3 cache can be shared and synchronized by the entire package, while L1 and L2 caches are shared by the core. These differences result in different ranges of attacks. Cache side-channel attacks targeting LLC can leak sensitive information to attackers who share the same package, while these attacks based on L1-D and L1-I cache can only leak to attackers who share the same core. This also explains why more side-channel attacks prefer to use L3 cache in their attacks [4, 8, 9, 12, 13, 17, 20, 22, 30–36, 38–41].

4.3. Attack Pattern. CACHE_ACT defines the communication mode between the attacker and the victim programs. There are 6 attack patterns.

4.3.1. Prime + Probe. In Prime + Probe mode, the attacker infers the behavior of the victim program by detecting which part of the cache from the attacker program has been evicted by the victim. The process of a single leak is shown in Figure 6. It has three steps.

First, in the prime stage, the attacker fills the cache with his own content (marked in yellow in the figure). Then, it lets the victim program run for some time. The time interval is related to the algorithm of the victim program and the sensitive information needed. During this time, the victim program will use the cache according to its own code logic. The data or code of the attacker's program in the corresponding cache is evicted because of cache conflict. These evicted cache lines are loaded with data from the victim, which is marked in blue in the figure. Then, in the probe stage, the attacker reads the data previously filled in the cache again and records the access time. If the read time exceeds the predefined threshold, it means the data in this cache line has been evicted by the victim program, and the corresponding data or code to the victim program was visited in the last time interval. Otherwise, the cache is not

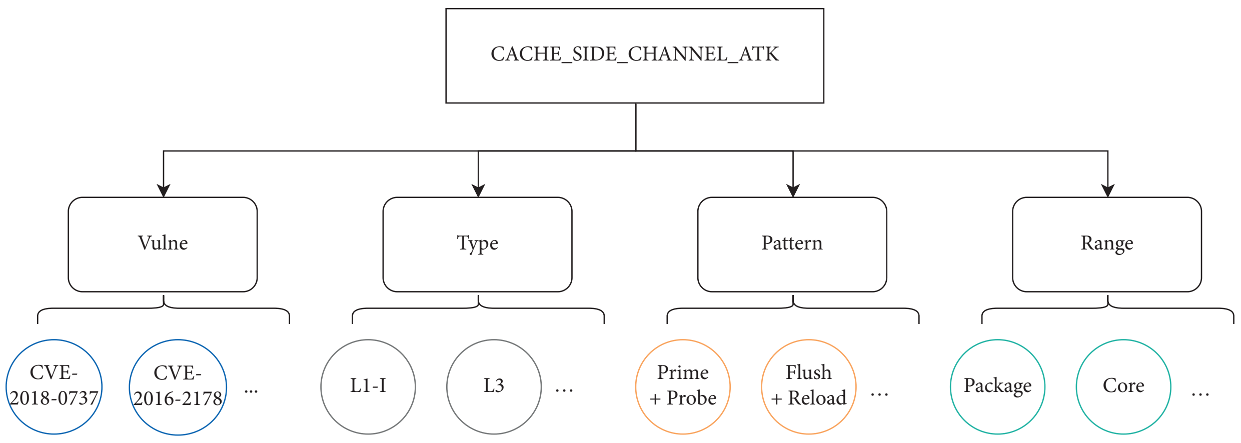

Figure 4: The model of the side-channel attacks.

Table 2: Research studies on cache-based side-channel attacks.

Attacks	Patterns	Cache types	Range	Target
[14]	Prime + Probe	L1-D	Core	OpenSSL(0.9.7c) RSA
[2]	Prime + Probe	L1-D	Core	AES
[5, 23]	Prime + Probe	L1-D	Core	OpenSSL(0.9.8) AES
[21]	Prime + Probe	L1-I	Core	OpenSSL(0.9.8d) RSA
[6]	Prime + Probe	L1-I	Core	OpenSSL(0.9.8e) RSA
[8]	Evict +time	LLC	CPU	Kernel memory
[20]	Flush + Reload	LLC	CPU	OpenSSL(0.9.8n) AES
[30]	Flush + Reload	LLC	CPU	GnuPG(1.4.13) RSA
[4]	Flush + Reload	LLC	CPU/VM	OpenSSL(1.0.1f) AES
[31–34]	Flush + Reload	LLC	CPU	OpenSSL(1.0.1e) ECDSA
[9]	Flush + Reload	LLC	CPU/VM	User activities
[35]	Flush + Reload	LLC	CPU	TLS andDTLS
[36]	Flush + Reload	LLC	CPU	Keyboard input
[37]	Prime + Probe	LLC	CPU	GnuPG(1.4.13/1.4.18) ElGamal
[22]	Flush + Reload	LLC	CPU	OpenSSL DSA
[38]	Prime + Probe	LLC	CPU	AES
[39]	Flush + Flush	LLC	Core	AES T-table
[16]	Invalidate + Transfer	—	System	Open SSL AES/libgcrypt ElGamal
[12, 13, 17]	Flush + reload/evict + time	LLC	CPU	Memory isolation
[40, 41]	Prime + Probe	LLC	CPU/enclave	RSA

```
diff --git a/crypto/rsa/rsa_gen.c b/crypto/rsa/rsa_gen.c
@@ –89, 6 +89, 8 @@ static int rsa_builtin_keygen (RSA *rsa, int bits, BIGNUM *e_value,
        if (BN_copy (rsa->e, e_value) == NULL)
             goto err;
+       BN_set_flags (rsa->p, BN_FLG_CONSTTIME);
+       BN_set_flags (rsa->q, BN_FLG_CONSTTIME);
        BN_set_flags (r2, BN_FLG_CONSTTIME);
        /* generate p and q */
        …
```

Figure 5: Patch code for CVE-2018-0737 vulnerability.

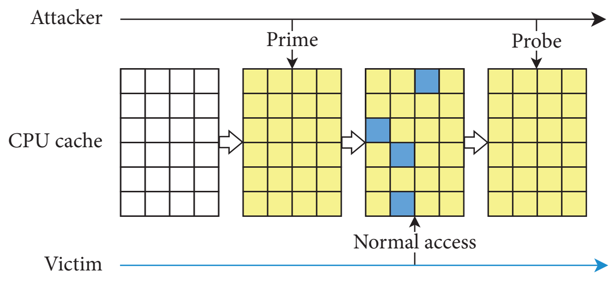

Figure 6: Workflow of Prime + Probe.

evicted, which means the victim did not give access to these cache lines. In the end, the complete sensitive information is leaked through multiple primings and probings.

4.3.2. Flush + Reload. Flush + Reload pattern is a variant of Prime + Probe. The multilayer inheritance structure of the CPU cache makes it inclusive. That is, the content in the high-level cache (such as L1) must also exist in the low-level cache (LLC). When content is expelled from the high-level cache, the low-level cache will also expel the corresponding cache line. Based on this fact, attacks utilize `CLFLUSH` instruction to empty the cache and launch the attack. The workflow of Flush + Reload is shown in Figure 7. The attacker first uses the CLFLUSH instruction to expel the content of the victim program from all caches. This makes sure that none of the victim's content is in the cache. Then, it lets the victim program run for a while. When the time slice is exhausted, the attacker program reloads the cached content and checks whether the content exists in the cache based on the reloading time. If the accessing time is short, it means the victim visited these contents in the previous phase. The Flush + Reload pattern requires shared memory to ensure that both the attacker and the victim can operate on the same memory content and the same cache line. If the attacker finds that these previously evicted contents appear in the cache, it means they were accessed by the victim program, which in turn reveals the running status and sensitive information of the victim program.

4.3.3. *Evict + Reload.*

Evict + Reload and Flush + Reload modes are similar. The main difference lies in the method of eviction. Usually in the Flush + Reload pattern, the attacker and victim programs have shared memory, so the corresponding cache line can be evicted directly through the CLFLUSH instruction. If these two do not have shared memory, attackers take the Evict + Reload pattern. The attack workflow is shown in Figure 8. It loads the memory that is competitive with the cache to evict cache lines (yellow blocks in the figure). That is, attackers find different memories that will be placed in the same cache line. In this way, when the attacker program accesses this memory, the backup of the victim program's corresponding memory in the cache will also be expelled from the cache line, thereby establishing the connection between them.

4.3.4. *Evict + Time.*

In Evict + Time, the attackers first let the victim program run normally. The program uses the cache normally, and they record the time to establish a baseline. Then, the attackers evict some cache lines and then let the victim program run again. By comparing the time and the baseline of the victim program, the attackers can determine whether the victim program uses the evicted cache line. Evict + Time requires multiple executions. If the victim program is only launched once, an accurate result cannot be obtained.

4.3.5. *Flush + Flush.*

The Flush + Flush pattern is based on the fact CLFLUSH instruction shows different execution times when facing different states of cache. If the target of CLFLUSH exists in the cache, it needs to be evicted from the multilevel cache during execution, so the execution time is longer. Otherwise, the execution time is shorter. The difference between the execution time allows the attacker's program to determine whether the cache line has been accessed by the victim through the execution time of the instruction.

The workflow of Flush + Flush is illustrated in Figure 9. After flushing all the cache lines, the normal access will load its contents into the cache. Then, the attacker will empty the cache again by CLFLUSH instruction and measure its

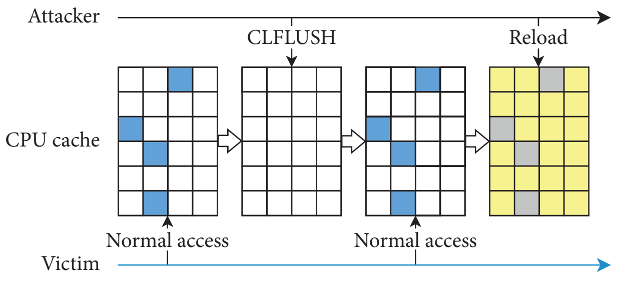

FIGURE 7: Workflow of Flush + Reload.

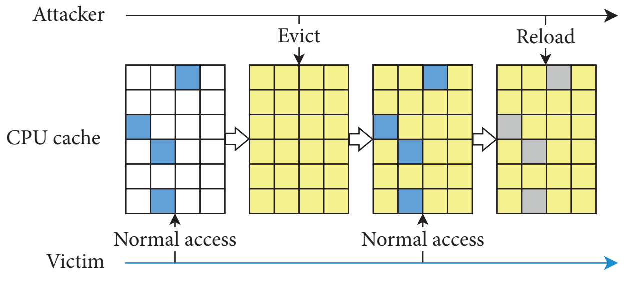

FIGURE 8: Workflow of Evict + Reload.

execution time. Different from other patterns, Flush + Flush utilizes the execution time of instructions rather than accessing memory, so it is prone to false positives and false negatives.

4.3.6. *Invalidate + Transfer.*

It seems impossible to leak sensitive information across processors in a platform with multiple processors because processes use separated CPU caches. However, Irazoqui [16] found that, to maintain cache consistency, the cache data of different CPUs are synchronized from different processors. When the memory that needs to be accessed is invalid in the cache of one CPU, it can be synchronized from the other one, which causes a time difference. In the Invalidate + Transfer pattern, the attacker program first evicts the shared memory from a CPU cache and then lets the victim program run. If the access time consumed when the victim program accesses the memory again is lower than the threshold, it means that the memory is re-accessed, and the previously evicted cache content is synchronized back to the cache across processors. In this way, the attacker can infer whether the target memory is accessed across different CPUs and then leak sensitive information.

4.4. *Range.*

The basis of cache side-channel attacks is cache, and different types of cache have their own scope of influence. L1 cache is core sharing, that is, only programs executed on the same physical core can access the same L1 cache at the same time, so the scope of the attack is also limited to the core range. We summarize the current attacks into four levels: package, core, NUMA, and system. The system sharing means resources that all processes in the system can access. NUMA sharing is smaller than the system sharing. It is based on the same memory controller. The

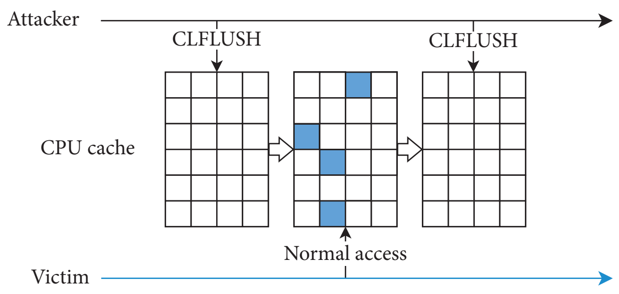

Figure 9: Workflow of Flush + Flush.

package range is shared between objects within the same package. Core sharing has the smallest range. When the range is defined, attacks can only leak sensitive information from victims in the same scope (same core or package).

4.5. Trend of the Attacks. There is a trend in the development of side-channel attacks:

(i) More common of the cache types

The cache utilized is transferring from the dedicated one to a common one. At first, the carrier of attacks was the L1 cache. The attack only occupied the data cache or instruction cache. The more dedicated the cache is, the more unique the cache state changes are and the simpler it is to infer the cause of the cache state change. For example, in the work [6], the authors utilized L1-I, which can only be shared over the core. That is to say, only when the target and the attacker programs are in the same domain (in this case, core), can the cache side-channel attack be carried out. This is a limitation because attackers have to use SMT or temporarily occupy the RSA process to share physical cores. Only these programs that can occupy physical cores at the same time as the target program can launch such attacks. The limitation is so strong that this side-channel attack cannot be widely used.

(ii) Wider range of the attack threats

The range of attack threats is getting wider. This is caused by the changes of the utilized cache type. Its attack capabilities range from the code shared by the local process to the memory shared between different local processes and then to code in the cloud environment (across different virtual machines). There are even attacks that leak sensitive information across different CPUs using data consistency. This means the attack surface becomes wider, which brings more challenges to privacy and security protection. More importantly, it also leads to the diversification of private information. Side-channel attacks originally target the keys involved in encryption and decryption algorithms such as AES and RSA. Nowadays, more higher-level sensitive information such as kernel address space layout or user behavior and keyboard strokes is being compromised as well.

(iii) Looser attack requirements

Requirements for attacks are getting looser. In the early days when cache side-channel attacks were first proposed, the conditions required to launch a complete attack were very harsh. For example, the access-driven side-channel attack designed by Michael et al. [2] requires the AES algorithm to occupy a separate lookup table in the last round. And, the attacker needs to preempt the AES process in a small interval. These conditions are unrealistic for ordinary programs, so the probability of a successful attack is small. However, the improved attacks are much more tolerant. The attack can succeed even in a system and platform with common settings. The attack proposed by Sinan Inci et al. [42] recovers the key in a cloud computing environment, only requires large pages to be enabled. In cloud computing platforms, the large page mechanism is usually configured for performance. This is especially true on some servers using oracle databases. These looser conditions greatly make it easier to launch side-channel attacks.

(iv) More stable channels

Channels are getting more and more stable. The stability of the channel has been improved. The instability of the attacks results from the instability of the cache. However, the new attack reduces the instability through new patterns and verification mechanisms. Additionally, attacks have evolved from the initial multiple runs to a single execution.

4.6. Attack Conditions. To threaten the system, the CPU cache-based side-channel attack needs to meet the following three conditions:

*T*1. There is a corresponding mapping between the state change of the cache and the sensitive information in the program

*T*2. In the range where the corresponding cache is shared, other programs are allowed to co-reside

*T*3. The co-resident program can infer the target cache status change through its own cache status

*T*1 condition is the basis of CPU cache-based side-channel attack. Since it utilizes the changes of cache states to leak sensitive information of the target program, different sensitive information definitely cause differences between cache states during the attack. Otherwise, the attacker cannot infer sensitive information by observing the changes in the cache states.

*T*2 condition ensures that the attack program and the victim program reside in the same hardware device. If *T*2 is not satisfied, they are deployed on different caches. The separation of the hardware usage makes it unavailable of the changes in the cache state, and thus, it blocks the inference of the sensitive information.

Even if *T*2 condition is met, attackers may not necessarily be able to use the cache to obtain sensitive information in the victim's program. The attackers also need to ensure that when the victim's cache status changes, they can speculate on these changes through their own caches. *T*3 condition demands that these caches are not only physically related but also logically inferred from each other.

Since it needs to meet these attack conditions in order to leak the sensitive information of the target program and bring threats to the system, we can also destroy these conditions in the attack scenario to invalidate the attack. As a matter of fact, all defense strategies against cache side-channel attacks are based on these perspectives to reduce the threat of attacks and protect the system. They will be discussed in the next section.

5. Analysis of the Defenses

By studying the reasons why defenses could take effect, we divide them into the different strategy types.

5.1. Information Independency. *T*1 requires a correlation between the sensitive information and the cache states during the operation. Information independency (constant time, for example) ruins this condition. It ensures that the behavior of the target program is completely independent of sensitive data, that is, cache access, branch selection, etc., are independent of sensitive information. In this case, even if the order of the cache or even the program execution is revealed, sensitive information can stay safe. Patches for OpenSSL vulnerabilities CVE-2018-0737 [27], CVE-2018-0734 [43], CVE-2018-12438 [44], CVE-2014-0076 [45], and CVE-2016-2178 [29] modify codes in different branches, making the program execution sequence unrelated to the key, thereby preventing the corresponding cache side-channel attack. Brickell [46] suggested not to perform sensitive information-dependent access operations on the memory that exceeds the granularity of the cache line, destroying the connection between sensitive information and memory access patterns. NaCl library [47] and libfixedtimefixpoint [48] turn the execution time of different branches in OpenSSL into constants that do not depend on inputs, which can defend against side-channel attacks.

5.2. Time Blinding. Modifying the time (CPU cycles) read by the attacker can affect the judgment of the cache state. The time blinding strategy is mainly divided into two major methods:

(i) Virtual time

The virtual time hides the real access time. When the attacker requests to read CPU cycles, the system returns a time that is constructed. Vattikonda et al. [49] designed a cloud computing environment with Xen that can provide virtual time. In this cloud environment, when an attacker requests to read CPU cycles, he will get a virtual clock cycle given by the system instead of the real time. Therefore, it is impossible to infer the change of the cache state of the victim program, which is effective for defending side-channel attacks.

(ii) Time black box

Cock et al. and Zhang et al. [50, 51] used time black box to mitigate side-channel attacks. Unlike the virtual time that modifies all the interfaces for detecting time, the black box treats the entire system as a whole and then takes control over the time of events that can be tested externally. In other words, the execution of the program is a black box, and the execution time of each part cannot be obtained. The time the attacker obtains from the outside is the time of the entire black box, so it is impossible to infer the execution status in the black box, let alone the connection between sensitive information and state changes.

Time blinding strategies ruin *T*3 to make their defenses effective.

5.3. Time Sharing. Regardless of the attack patterns mentioned in Section 4.3, the victim and attacker programs use the cache in turn. Through the time-sharing utilization of the cache, the threat of cache side-channel attacks can be reduced. Godfrey et al. [52] proposed an idea that clears all cache contents when different VMs are switched. However, it brings huge performance overhead to the system. In the experiment of Benjamin et al. [53], clearing the contents of the L1 cache will cause a 17% performance cost. If the LLC is cleared out of the cache every time the VM is switched, the performance overhead will be even greater. Time sharing strategy realizes the defense by ruining *T*3.

5.4. Resource Isolation. Different from the time-sharing strategy in which shared resources are provided to the victim and attacker programs, the resource division strategy divides the resources for use by different programs. That is, different resources are no longer shared by multiple programs. They are now exclusive. The basis of the resource isolation strategy is to reduce the shared area. For example, turning off the hyperthreading in hardware reduces the resource sharing caused by SMT and prevents different threads from accessing the cache. In fact, most cloud service providers, such as Microsoft's Azure [54], turned off SMT to gain higher security guarantees. VMWare [53] suggests turning off the page sharing feature in the configuration to resist cross-virtual machine side-channel attacks.

Resource isolation strategies are applied to CPU caches as well, which are mainly divided into hardware isolation and software isolation. Hardware isolation aims to use a hardware mechanism to ensure that the cache line is exclusively occupied by the victim program, so the attacker program cannot use its own cache performance to infer the changes in the victim program. It ruins *T*3 to counter side-channel attacks. Cache Allocation Technology (CAT) provided by Intel can prevent specific cache lines in the LLC from being replaced. CATalyst [55] manages the use of cache through CAT. Its experiments prove that it shows effective defense against LLC-based side-channel attacks in cloud computing environments. Partition-locked cache (PLcache)

[56] adds a locking attribute to the use of the cache, which isolates the use of cache lines. It destroys *T*3 by reducing the conflict between the victim and the attacker on the cache line.

The software isolation of resources mainly uses the method of cache coloring. The cache coloring algorithm was originally proposed to improve the cache replacement efficiency, reducing conflicts and increasing the hit rate for higher system performance. Cache coloring divides the cache into different color blocks and indexes the color blocks that should be stored according to the coloring bits. When the cache group is mapped, the high part of the index is used as the coloring bits to determine the corresponding cache line. If the physical address (PA) is used as the index of the cache, the coloring bits are not only the high part of the set index but also the low part of the frame number during memory addressing. This ensures that if the memory belongs to different pages, it will be allocated to different color blocks.

With cache coloring, Kim et al. [57] proposed a function similar to PLcache, which allows the target program to monopolize the cache line of a certain color block so that the attacker and the victim cannot share the same cache line in side-channel attacks. It also ruins *T*3.

5.5. Anti-Co-Resident Detection. *T*2 shows that the attacker can only carry out a complete side-channel attack when it is co-resident in the same environment with the victim. This is especially essential in cloud computing platforms because the attacker needs to check whether it is co-resident with the target victim through co-resident detection. The attacker can launch co-resident detection based on network information, resource interference, and other information.

The authors of [15, 58] proposed a method of co-resident information exchange across VMM using a covert channel. However, it requires both the attacker and the victim to be under complete control, so it can only be used for proof of concept, which means it will not be exploited in actual threats. Attackers can perform co-resident detection through the occupation of hardware resources. When the attacker and the victim program are in a co-resident state, resource preemption and scheduling will inevitably occur, which in turn gives the attacker the ability to search for attackable objects under the platform. In the work by Adam et al. [59], the network is used to determine whether two virtual machines are under the same physical host. The attacker keeps sending network packets to occupy the physical network card, which makes the network communication delay between it and the target virtual machine increase. Then, it infers if the target is on the same physical host as it. In this way, the co-resident detection is completed. Zhang et al. [60], on the contrary, used statistical cache load to infer whether there are other virtual machines in the corresponding environment. Si Yu et al. [61] later further abstracted the modeling of this method, which improved the accuracy.

Attackers can also find co-resident targets through network information. The host platform often assigns the same or similar IP addresses to the virtual machines. Based on the IP information, Ristenpart et al. [15] successfully performed co-resident detection on the Amazon Elastic Compute Cloud server. The co-resident detection based on network information can be repaired by software updates. The current Amazon EC2 server has configured its network, and network information can no longer be used for co-resident detection. However, the idea of using network information to detect other virtual machines existing in the same physical machine is the most commonly used method. It is simple but effective.

Co-resident detection is an important step in cross-VM side-channel attacks. By optimizing virtual machine isolation and hardware resource management on cloud services, attackers are prevented from inferring co-resident information through changes in hardware performance. In addition, by modifying the network configuration, the co-resident information is hidden in the cloud services. This destroys *T*2 and makes cross-VM side-channel attacks impossible to proceed normally.

However, anti-co-resident detection is mostly used for defenses in cloud servers. In a local side-channel attack, the attacker already has strong prior knowledge of the victim's program and confirms that the victim's program has been executed. The attacker may have the source code or binary code or even the ability to execute the victim's program multiple times. In the face of this situation, the anti-co-resident detection strategies may fail.

5.6. Channel Interference. A stable and usable channel is the practical basis for side-channel attacks. Injecting noise into the cache can interfere with channel usage. Hu [62] designed fuzzy time to inject noise into various events. Similarly, Brickell et al. [63] compress, reload, and randomize the lookup table in the encryption and decryption of AES to defend cache side-channel attacks. For normal encryption and decryption algorithms, the cache changes caused by these operations are noises. The injected noise interferes with the side channel so that the attacker program cannot use the cache to guess how the victim program uses the cache, which ruins *T*3.

There are many techniques to interfere with the channel. RPcache [56] (random permutation cache) uses a cache random indexing method to bring in extra entropy, which prevents attacker programs from preempting and speculating on the cache. Vattikonda [49] uses the hypervisor to insert noise in the time measurement of the event. In addition, Zhang et al. [64] set up a bystander virtual machine to inject noise into the L2 cache of the entire virtual machine platform. Experiments show that the noise the bystander virtual machine brings in can effectively interfere with the channel.

However, channel interference is not perfect. For high-security requirements, cracking channel interference is only a game of complexity and time. In order to strengthen the channel, the attacker proposed different countermeasures: use the error correction code or the SSH protocol to enhance the robustness of the buffer channel [65]. These enhancement measures can bypass channel interference defenses such as injected noise.

With the deepening of the research on side-channel attacks, more forms and variants of attacks have been developed. However, all the new attack methods proposed are still within the model mentioned in Section 3. Therefore, the defense used to respond has not been separated away from *T*1, *T*2, and *T*3.

6. Challenges and Trends

Side-channel attacks are a common security problem but not easy to be noticed. Cache-based side-channel attacks are still a huge threat for programs in both the cloud and local environment because of the challenges:

(i) Different from common vulnerabilities, it links sensitive information and state changes, which is unexpected for developers, resulting in compromising of privacy. Side-channel attacks use CPU cache as a carrier, which is hardly paid attention in developing. However, CPU caches are so important in modern computer architecture that it is impossible to break away from the performance improvement brought by them, so the CPU cache-based side-channel attack has a wide range of application environments.

(ii) Considering efficiency and performance, environments (especially cloud computing service platforms) are unwilling to completely isolate hardware resources. The future of cloud computing is sharing and reuse. It is impossible to deploy massive dedicated resources.

(iii) Artificial Intelligence helps to recognize the pattern of cache activities. AI algorithms reduce the difficulty of connecting sensitive information and cache states and thus make the channel much more usable.

According to the challenges, defense strategies should be explored from the following perspectives in the future.

(i) Strengthen security awareness of programmers, and create more effective code review policies in software engineering

Developers should avoid connecting sensitive information with cache states during the execution of the programs. Good programing habits contribute to more security against side-channel attacks. Besides, secure codes such as libraries should be developed and reused. Using reviewed codes can reduce the risk of introducing side-channel attack vulnerabilities.

(ii) Deploy more efficient resource isolation

It is essential to design an efficient and secure resources isolation mechanism against side-channel attacks. More software and hardware features should be developed and utilized to balance security and performance, which is a hot research spot to be studied and quantified in the future.

(iii) The diversification of obfuscation and identification of the cache

Research studies trying to hide the access mode of memory are meaningful. ORAM (Oblivious Random Access Machine), for example, is one of the potential schemes. It encrypts information such as the sequence, time, and frequency of access during IO operations, thereby protecting private data. Although the additional storage and performance overhead brought by ORAM is still a problem for the time being, it cannot be denied its significance in combating attacks involving cache states. Similarly, obfuscation technologies can be deployed to hide access modes and other information that can be utilized in side-channel attacks.

7. Conclusion

This paper focuses on side-channel attacks that are based on CPU caches. By comparing different types of attacks, we summarize the general workflow of the attack and propose a model that distinguishes attacks from four perspectives, namely, vulnerability, cache type, pattern, and range. Discussion on the model helps to conclude necessary conditions of introducing security threats in side-channel attacks. Meanwhile, defense strategies are classified into 6 types, according to how they take effect in the actual world.

We conduct a systematic analysis that provides a comprehensive understanding of the CPU cache-based side-channel attacks and their defense strategies. The proposed analysis model suggests that the attacks can be analyzed from the code vulnerability, cache type, attack pattern, and range. We expect that further research studies could create new attacks by changing these factors. In addition, the attack conditions indicate the basis of the defense strategies. Interesting directions to follow will be invalidating these conditions and proposing more effective defenses. CPU cache-based side-channel attacks are more dangerous than expected. We hope this survey can attract more attention and promote the development of research studies on them.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partly supported by the National NSF of China under Grant nos. 61772266 and 61431008.

References

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Proceedings of the Advances in Cryptology—CRYPTO'96, 16th Annual International Cryptology Conference, Volume 1109 of Lecture Notes in Computer Science*, pp. 104–113, Springer, Santa Barbara, CA, USA, August 1996.

[2] M. Neve and J.-P. Seifert, "Advances on access-driven cache attacks on AES," in *Proceedings of the Selected Areas in*

Cryptography, 13th International Workshop, SAC 2006, Volume 4356 of Lecture Notes in Computer Science, pp. 147–162, Springer,, Montreal, Canada, August 2006.

[3] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T.-H. Lai, “Sgxpectre: stealing intel secrets from SGX enclaves via speculative execution,” *IEEE Security & Privacy*, vol. 18, no. 3, pp. 28–37, 2020.

[4] G. I. Apecechea, M. Sinan Inci, T. Eisenbarth, and B. Sunar, “Wait a minute! A fast, cross-vm attack on AES,” in *Proceedings of the Research in Attacks, Intrusions and Defenses—17th International Symposium, RAID 2014, Volume 8688 of Lecture Notes in Computer Science*, pp. 299–319, Springer, Gothenburg, Sweden, September 2014.

[5] D. A. Osvik, A. Shamir, and E. Tromer, “Cache attacks and countermeasures: the case of AES,” in *Proceedings of the Topics in Cryptology—CT-RSA 2006, The Cryptographers’ Track at the RSA Conference 2006, Volume 3860 of Lecture Notes in Computer Science*, pp. 1–20, Springer, San Jose, CA, USA, February 2006.

[6] O. Aciiçmez and W. Schindler, “A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on openssl,” in *Proceedings of the Topics in Cryptology—CT-RSA 2008, the Cryptographers’ Track at the RSA Conference 2008, Volume 4964 of Lecture Notes in Computer Science*, pp. 256–273, Springer, San Francisco, CA, USA, April 2008.

[7] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, “State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures,” in *Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) HOST 2010*, pp. 76–87, Anaheim Convention Center IEEE Computer Society, Anaheim, CA, USA, June 2010.

[8] R. Hund, C. Willems, and T. Holz, “Practical timing side channel attacks against kernel space ASLR,” in *Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS 2013*, The Internet Society, San Diego, California, USA, February 2013.

[9] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-tenant side-channel attacks in paas clouds,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 990–1003, ACM, Scottsdale, AZ, USA, November, 2014.

[10] G. Qian, Y. Yarom, D. Cock, and G. Heiser, “A survey of microarchitectural timing attacks and countermeasures on contemporary hardware,” *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, 2018.

[11] M. Devi and A. Majumder, “Side-channel attack in internet of things: a survey,” in *Applications of Internet of Things*, pp. 213–222, Springer, Berlin, Germany, 2021.

[12] K. Paul, J. Horn, F. Anders et al., “Spectre attacks: exploiting speculative execution,” in *Proceedings of the 2019 IEEE Symposium on Security and Privacy, SP 2019*, pp. 1–19, IEEE, San Francisco, CA, USA, May 2019.

[13] M. Lipp, M. Schwarz, D. Gruss et al., “Meltdown: reading kernel memory from user space,” in *Proceedings of the 27th USENIX Security Symposium, USENIX Security 2018*, pp. 973–990, USENIX Association, Baltimore, MD, USA, August 2018.

[14] C. Percival, “Cache missing for fun and profit,” in *Proceedings of the BSDCan 2005*, Ottawa, Canada, March 2021, http://www.daemonology.net/hyperthreading-considered-harmful/+-.

[15] T. Ristenpart, E. Tromer, H. Shacham, S. Savage, and Y. Hey, “Get off of my cloud: exploring information leakage in third-party compute clouds,” in *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*, pp. 199–212, ACM, Chicago, IL, USA, November 2009.

[16] G. Irazoqui, T. Eisenbarth, and B. Sunar, “Cross processor cache attacks,” in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016*, pp. 353–364, ACM, Xi’an, China, May 2016.

[17] J. Wampler, Ian Martiny, and E. Wustrow, “Exspectre: hiding malware in speculative execution,” in *Proceedings of the 26th Annual Network and Distributed System Security Symposium, NDSS 2019*, The Internet Society, San Diego, CA, USA, February 2019.

[18] S. Chakrabarti, T. Knauth, D. Kuvaiskii, M. Steiner, and M. Vij, “Trusted execution environment with intel sgx,” *Responsible Genomic Data Sharing*, Elsevier, Amsterdam, Netherlands, pp. 161–190, 2020.

[19] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, “Cache attacks on intel SGX,” in *Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017*, pp. 2:1–2:6, ACM, Belgrade, Serbia, April 2017.

[20] D. Gullasch, E. Bangerter, and S. Krenn, “Cache games—bringing access-based cache attacks on AES to practice,” in *Proceedings of the 32nd IEEE Symposium on Security and Privacy, S&P 2011*, pp. 490–505, IEEE Computer Society, Berkeley, CA, USA, May 2011.

[21] O. Aciiçmez, “Yet another microarchitectural attack: exploiting i-cache,” in *Proceedings of the 2007 ACM workshop on Computer Security Architecture, CSAW 2007*, pp. 11–18, ACM, Fairfax, VA, USA, November 2007.

[22] C. P. García, B. B. Brumley, and Y. Yarom, “Make sure DSA signing exponentiations really are constant-time,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1639–1650, ACM, Vienna, Austria, October 2016.

[23] E. Tromer, D. A. Osvik, D. A. Osvik, and A. Shamir, “Efficient cache attacks on aes, and countermeasures,” *Journal of Cryptology*, vol. 23, no. 1, pp. 37–71, 2010.

[24] Y. Wang and G. Edward Suh, “Efficient timing channel protection for on-chip networks,” in *Proceedings of the 2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, pp. 142–151, IEEE Computer Society, Copenhagen, Denmark, May 2012.

[25] D. H. Woo and H. H. Lee, “Analyzing performance vulnerability due to resource denial of service attack on chip multiprocessors,” in *Proceedings of the Workshop on Chip Multiprocessor Memory Systems and Interconnects*, Phoenix, AZ, USA, February 2007.

[26] T. Zhang, Y. Zhang, and B. Ruby, “Memory dos attacks in multi-tenant clouds: severity and mitigation,” 2016, https://arxiv.org/abs/1603.03404.

[27] National Vulnerability Database, “The openssl Rsa Key generation cache timing side channel attack vulnerabilities, cve-2018-0737,” 2018, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-0737.

[28] National Vulnerability Database, “Cpu side-channel information disclosure vulnerabilities, Cve-2017-5754,” March 2021, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5754.

[29] National Vulnerability Database, “Dsa private key side-channel attack vulnerabilities, Cve-2016-2178,” March 2021, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2178.

[30] Y. Yarom and K. Falkner, “FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack,” in

Proceedings of the 23rd USENIX Security Symposium, pp. 719–732, USENIX Association, San Diego, CA, USA, August 2014.

[31] T. Allan, B. B. Brumley, and E. Katrina, "Falkner, Joop van de Pol, and Yuval Yarom. Amplifying side channels through performance degradation," in *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016*, pp. 422–435, ACM, Los Angeles, CA, USA, December 2016.

[32] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom, ""Ooh aah ... just a little bit": a small amount of side channel can go a long way . . . just a little bit": a small amount of side channel can go a long way," in *Proceedings of the Cryptographic Hardware and Embedded Systems: CHES 2014—16th International Workshop, Volume 8731 of Lecture Notes in Computer Science, Advanced Information Systems Engineering*, pp. 75–92, Springer, Busan, South Korea, September 2014.

[33] Joop van de Pol and P. Nigel, "Smart, and Yuval Yarom. Just a little bit more," in *Proceedings of the Topics in Cryptology—CT-RSA 2015, the Cryptographer's Track at the RSA Conference 2015, Volume 9048 of Lecture Notes in Computer Science*, pp. 3–21, Springer, San Francisco, CA, USA, April 2015.

[34] Y. Yarom and N. Benger, "Recovering openssl ECDSA nonces using the FLUSH+RELOAD cache side-channel attack," *IACR Cryptology ePrint Archive*, vol. 140, p. 2014, 2014.

[35] G. I. Apecechea, M. Sinan Inci, T. Eisenbarth, and B. Sunar, "Lucky 13 strikes back," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS'15*, pp. 85–96, ACM, Singapore, April 2015.

[36] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: automating attacks on inclusive last-level caches," in *Proceedings of the 24th USENIX Security Symposium, USENIX Security 15*, pp. 897–912, USENIX Association, Washington, DC, USA, August 2015.

[37] F. Liu, Y. Yarom, G. Qian, G. Heiser, B. Ruby, and Lee, "Last-level cache side-channel attacks are practical," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP 2015*, pp. 605–622, IEEE Computer Society, San Jose, CA, USA, May 2015.

[38] M. Kayaalp, B. Nael, Abu-Ghazaleh, D. V. Ponomarev, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016*, pp. 72:1–72:6, ACM, Austin, TX, USA, June 5-9, 2016.

[39] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: a fast and stealthy cache attack," in *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment—13th International Conference, DIMVA 2016, Volume 9721 of Lecture Notes in Computer Science*, pp. 279–299, Springer, San Sebastián, Spain, July 2016.

[40] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proceedings of the 11th USENIX Workshop on Offensive Technologies, WOOT 2017*, USENIX Association, Vancouver Canada, August 2017.

[41] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: using SGX to conceal cache attacks," in *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment—14th International Conference, DIMVA 2017, volume 10327 of Lecture Notes in Computer Science*, pp. 3–24, Springer, Bonn, Germany, July 2017.

[42] M. Sinan Inci, B. Gülmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *Proceedings of the Cryptographic Hardware and Embedded Systems: CHES 2016—18th International Conference, Volume 9813 of Lecture Notes in Computer Science*, pp. 368–388, Springer, Santa Barbara, CA, USA, August 2016.

[43] National Vulnerability Database, "The openssl Dsa signature algorithm timing side channel attack vulnerabilities, cve-2018-0734," 2018, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-0734.

[44] National Vulnerability Database, "The elliptic curve cryptography library memory-cache side-channel attack vulnerabilities, cve-2018-12438," March 2021, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12438.

[45] National Vulnerability Database, "The openssl montgomery ladder implementation flush+reload cache side-channel attack vulnerabilities, Cve-2014-0076," March 2021, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0076.

[46] E. Brickell, "Technologies to improve platform security," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*, vol. 11, Nara, Japan, September-October 2011.

[47] D. J. Bernstein, T. Lange, and S. Peter, "The security impact of a new cryptographic library," in *Proceedings of the Progress in Cryptology—LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Volume 7533 of Lecture Notes in Computer Science*, pp. 159–176, Springer, Santiago, Chile, October 2012.

[48] M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham, "On subnormal floating point and abnormal timing," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP 2015*, pp. 623–639, IEEE Computer Society, San Jose, CA, USA, May 2015.

[49] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in xen," in *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011*, pp. 41–46, ACM, Chicago, IL, USA, October 2011.

[50] D. Cock, G. Qian, T. C. Murray, and G. Heiser, "The last mile: an empirical study of timing channels on sel4," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 570–581, ACM, Scottsdale, AZ, USA, November 2014.

[51] D. Zhang, A. Aslan, and A. C. Myers, "Predictive mitigation of timing channels in interactive systems," in *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011*, pp. 563–574, ACM, Chicago, IL, USA, October 2011.

[52] M. M. Godfrey and M. Zulkernine, "A server-side solution to cache-based side-channel attacks in the cloud," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, pp. 163–170, IEEE Computer Society, Santa Clara, CA, USA, June-July 2013.

[53] S. Benjamin, K. Markantonakis, R. N. Akram, and K. Jan, "Leaky controller: cross-vm memory controller covert channel on multi-core systems," in *Proceedings of the ICT Systems Security and Privacy Protection—35th IFIP TC 11 International Conference, SEC 2020, Volume 580 of IFIP Advances in Information and Communication Technology*, pp. 3–16, Springer, Maribor, Slovenia, September 2020.

[54] M. Corp, "Microsoft azure," March 2021, https://azure.microsoft.com/.

[55] F. Liu, G. Qian, Y. Yarom et al., "Catalyst: defeating last-level cache side channel attacks in cloud computing," in *Proceedings of the 2016 IEEE International Symposium on High Performance*

Computer Architecture, HPCA 2016, pp. 406–418, IEEE Computer Society, Barcelona, Spain, March 2016.

[56] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 34th International Symposium on Computer Architecture (ISCA 2007), ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 494–505, ACM, San Diego, CA, USA, June 2007.

[57] T. Kim, P. Marcus, and G. Mainar-Ruiz, "STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud," in *Proceedings of the 21th USENIX Security Symposium*, pp. 189–204, USENIX Association, Bellevue, WA, USA, August 2012.

[58] M. Sinan Inci, B. Gülmezoglu, G. I. Apecechea, T. Eisenbarth, and B. Sunar, "Seriously, get off my cloud! cross-vm RSA key recovery in a public cloud," *IACR Cryptology ePrint Archive*, vol. 898, p. 2015, 2015.

[59] B. Adam, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and R. B. Kevin, "On detecting co-resident cloud instances using network flow watermarking techniques," *International Journal of Information Security*, vol. 13, no. 2, pp. 171–189, 2014.

[60] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "Homealone: Co-residency detection in the cloud via side-channel analysis," in *Proceedings of the 32nd IEEE Symposium on Security and Privacy, S&P 2011*, pp. 313–328, IEEE Computer Society, Berkeley, CA, USA, May 2011.

[61] S. Yu, X. Gui, X. Zhang, J. Lin, and J. Wang, "Co-residency detection scheme based on shared cache in the cloud," *Journal of Computer Research and Development*, vol. 50, no. 12, pp. 2651–2660, 2013.

[62] W.-M. Hu, "Reducing timing channels with fuzzy time," in *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pp. 8–20, IEEE Computer Society, Oakland, CA, USA, May 1991.

[63] E. Brickell, G. Gary, M. Neve, and J.-P. Seifert, "Software mitigations to hedge AES against cache-based software side channel vulnerabilities," *IACR Cryptology ePrint Archive*, vol. 52, p. 2006, 2006.

[64] R. Zhang, X. Su, J. Wang et al., "On mitigating the risk of cross-vm covert channels in a public cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 8, pp. 2327–2339, 2015.

[65] C. Maurice, M. Weber, M. Schwarz et al., "Hello from the other side: SSH over robust cache covert channels in the cloud," in *Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS 2017*, The Internet Society, San Diego, CA, USA, February 26 - March 2017.