Flush + Reload Side-Channel Attack

Samuel Youssef
Department of Electrical Engineering
The City College of the City University of New York
New York, US

Ricky Valdez
Department of Electrical Engineering
The City College of the City University of New York
New York, US

***Abstract*— This paper will be on the Flush + Reload side channel attack. First, we'll discuss the cache hierarchy and page sharing, and how page sharing can present vulnerabilities. Then, we'll go into details of the flush + reload attack and how the attacker can gain information by using this vulnerability. Next, we'll talk about the applications of the attack, first by discussing a paper that demonstrates how RSA encryption private key can be compromised by using this type of attack. Then, we'll talk about another paper detailing how the attack can be used on a victim using a web browser to see which web pages the victim has been visiting.**

Keywords—Flush + Reload, Last Level Cache, RSA, probe, Levenshtein Distance Algorithm, Side-Channel.

I. INTRODUCTION

Side-channel attacks (SCAs) aim at extracting secrets from a chip or a system, through measurement and analysis of physical parameters. Examples of such parameters include supply current, execution time, and electromagnetic emission. These attacks pose a serious threat to modules that integrate cryptographic systems, as many side-channel analysis techniques have proven successful in breaking an algorithmically robust cryptographic operation (for example, encryption) and extracting the secret key. Side-channel attacks work by monitoring the emissions produced by electronic circuits when the victim's computer is being used. In addition to exploiting information about power consumption and electromagnetic fields, an attacker may listen to the sounds a central processing unit (CPU) produces and use that information to reverse engineer what the computer is doing. This type of side-channel attack is called an acoustic cryptanalysis attack. Other types of side-channel attacks include: Cache attacks that exploit how and when cache is accessed in a physical system, Differential fault analysis attacks that seek to glean information from a system by introducing faults into the system's computations, Timing attacks that track the movement of data to and from a system's CPU and memory, Thermal-imaging attacks that use infrared images to observe the surface of a CPU chip and collect executed code, and Optical side-channel attacks that collect information about hard disk activity by using an audio/visual recorder, such as a video camera. In this paper, we are going to explore Cache attacks, specifically Flush + Reload attacks that monitor Last Level Cache (LLC) to extract information about a victim's process through measuring the access time to shared data between the attacker and the victim.

II. PAGE SHARING

Page sharing allows separate processes to access the same page stored in the cache. It is used for efficiently utilizing cache and memory space by avoiding having separate processes create duplicates of the same block within the cache or memory. Since unrelated processes are sharing the content stored in the cache, it leaves a process vulnerable to an attacker modifying that shared content from an external process. To prevent an attacker from modifying shared content, hardware can be implemented to enforce copy-on-write semantics. Copy-on-write only allow processes to read from shared content; if a process tries to write into a shared page, it instead gets interrupted. During the interrupt a copy of the shared content is created, and the writing process gets mapped to the copy before it resumes the write operation. Although copy-on-write prevents attackers from modifying shared content, the delay that is introduced from modifying a shared page can be detected by external processes, potentially allowing data to be leaked.

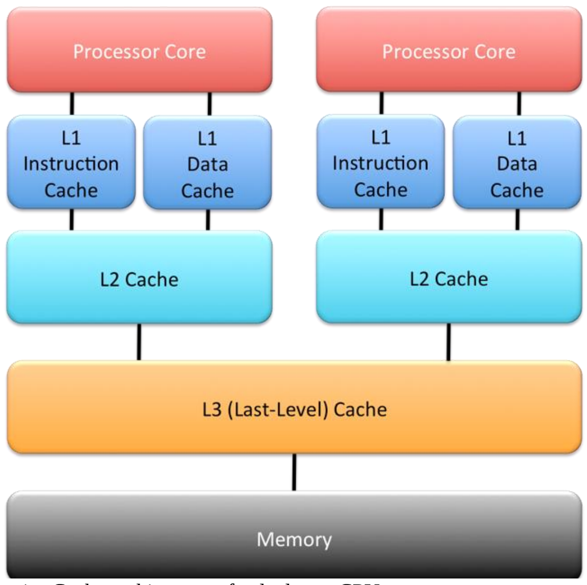


Fig. 1. Cache architecture of a dual core CPU.

In figure 1, we have the cache architecture of an x86 processor with two cores. For the Flush + Reload attack we assume that the cache is an inclusive cache, which is the case in most modern Intel processors. Inclusive caches create a copy of all data in lower level caches into the last level cache and whenever a page is removed from the cache, it removes all copies of that page from every level of the cache. Unlike L1 and L2 cache which are in each individual core of the processor, the LLC (L3) is located outside the cores of the CPU allowing all cores to access it.

III. FLUSH + RELOAD ATTACK

A. *Phases of the Attack*

The Flush + Reload attack targets the LLC, this allows the attacker to spy on a victim program that is located on a separate core through unrestricted use of the *clflush* instruction. The *clflush* instruction takes in a memory line address and clears the data referenced by that address from all cache levels. The Flush + Reload attack can be divided into three stages:

1) *FLUSH:* In the flush phase, the spy program calls the clflush command which clears the shared data from the cache completely, that includes any copies in the L1 or L2 cache of every core.
2) *WAIT:* In the wait stage, the spy program waits idly for some time, the purpose being that the attacker wants to give the victim program enough time to perform a cache miss and access the shared data from main memory.
3) *RELOAD:* In the reload stage, the spy program tries to read the shared data and it records the time it takes to perform the load.

Depending on how long it takes to load, the attacker can learn whether the victim has recently accessed the shared data, since it takes significantly longer to read from main memory than to read from the cache. If the reload time is short, then that means that the victim accessed the shared data recently and placed a new copy in the cache. Otherwise, if the load took long, that means that the victim hasn't accessed the page recently, so there wasn't a copy in the cache.

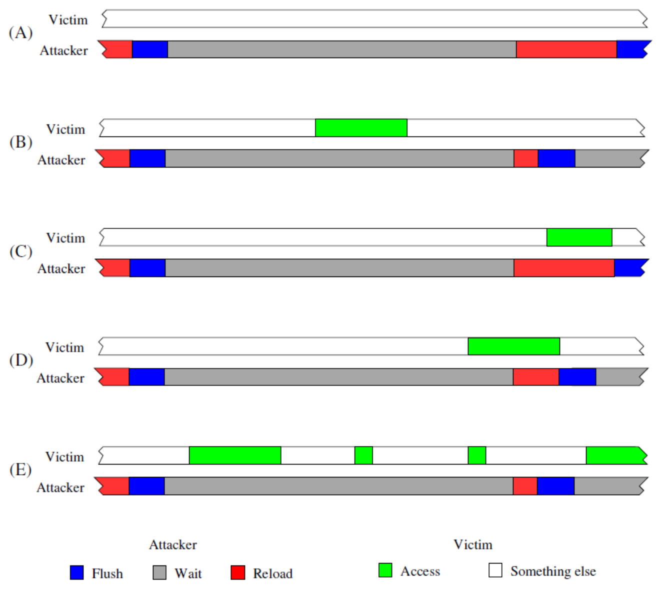


Fig. 2. Timing of FLUSH+RELOAD. (A) No Victim Access (B) With Victim Access (C) Victim Access Overlap (D) Partial Overlap (E) Multiple Victim Accesses [1].

Figure 2 shows the timing for all the possible cases for the flush + reload attack. Cases A and B are the ideal cases in which the victim either accesses or doesn't access at all the shared data during the wait phase, allowing the attacker to successfully determine whether the victim has accessed the shared data. The attack isn't always successful and cases C, D, E show how the attacker can miss an access to memory. In case C, the victim accesses the shared data while the attacker is in the reload phase and in case D the attacker reloads while the victim is still accessing memory. In case E the attacker waits too long, giving the victim time to access the shared cache multiple times, but the attacker only thinks the victim only accessed the shared data once. In order to minimize the probability of missing an access, the attacker must wait longer enough to reduce the odds of the access overlapping with the reload, but short enough that multiple accesses don't occur during the wait.

IV. RSA

In the paper by Yuval Yarom and Katrina Falkner [1], they demonstrated an application of the flush + reload attack in cryptography. The paper investigates RSA encryption of GnuPG, which uses a series of exponentiations and modulations to encrypt data. RSA is a method for encrypting and decrypting a message using public and private keys described in a paper by R. L. Rivest, A. Shamir, and L. Adleman [2].

```
1  function exponent($b$, $e$, $m$)
2  begin
3    $x \leftarrow 1$
4    for $i \leftarrow |e| - 1$ downto 0 do
5      $x \leftarrow x^2$
6      $x \leftarrow x \bmod m$
7      if $(e_i = 1)$ then
8        $x \leftarrow xb$
9        $x \leftarrow x \bmod m$
10     endif
11   done
12   return $x$
13 end
```

Fig. 3. Exponentiation by Square-and-Multiply [1].

In Figure 3 we have one of the functions that is frequently called in RSA encryption which simply calculates $b^e mod\ m$. An important fact to note is that the second input to the function, e, is a private encryption key for RSA, making it critical that it remains secret from all other processes. To calculate the exponent, a technique called squared and multiply is used in order to reduce the time complexity to O (log e), down from O(e). The function starts by setting the variable x to 1 then converting e into its binary representation and it iterates through the binary one bit at a time, starting from the most significant bit. For each bit, x is squared, and if that bit is equal to 1, x is multiplied by b. Lines 8 and 9 in the figure is contained within the IF block and it only runs whenever the current bit in e is equal to 1, therefore an attacker that is using Flush + Reload attack will want to monitor these two particular instructions in the cache. So, whenever the spy process reads a fast reload time, they know that the current bit in e is a 1, and it is a 0 if it has a slow reload time. Overall the flush + reload attack was able to successfully recover 96.7% of the bits for the private key of victim process that is in a separated core, as well as a victim process that is running in a separate virtual machine.

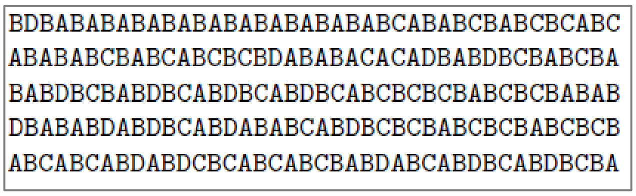
V. METHODOLOGY

A. *Description of the Attack*

In this section, we describe a step-by-step procedure of carrying out Flush + Reload attack through a command line web browser called Links. Links was used instead of other common browsers like Google Chrome or Firefox because it has a much smaller code base which makes it easier to understand quickly. In addition to that, the list of functions probed to help differentiate between various web pages were much easier to analyze using Links than other common web browsers. The idea behind this attack is to know which pages the victim has visited out of a set of known pages. Hornby found that it is possible to reliably distinguish between the top 100 Wikipedia pages of 2013 [3]. By running an automation tool developed by Hornby, a list of HTML parsing functions inside Links were generated to help probe specific cache lines that store important information about data shared between the victim and the attacker. The list of functions includes:

- html_stack_item (): corresponds to letter A in the probe sequence
- html_stack_dup (): corresponds to letter B in the probe sequence
- html_a (): corresponds to letter C in the probe sequence
- parse_html (): corresponds to letter D in the probe sequence

The Flush + Reload Spy Tool used during the attack generates a probe sequence. The probe sequence is a string of letters that correspond to the HTML parsing functions that Links browser used while rendering the page that the victim has visited (example of probe sequence is shown in figure 4 below). The probe sequence string is used later in identifying which page exactly the victim has visited out of a set of known pages to the attacker.

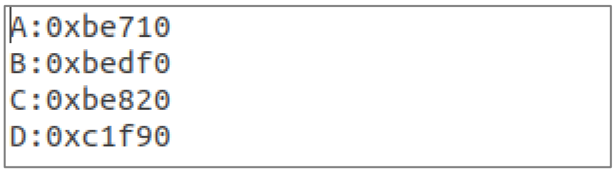


```
BDBABABABABABABABABABABABCABABCBABCBCABC
ABABABCBABCABCBCBDABABACACADBABDBCBABCBA
BABDBCBABDBCABDBCABDBCABCBCBCBABCBCBABAB
DBABABDABDBCABDABABCABDBCBCBABCBCBABCBCB
ABCABCABDABDCBCABCABCBABDABCABDBCABDBCBA
```

Fig. 4. Probe sequence string generated by spy tool.

B. Configuration:

In this section, we will discuss hardware and software configurations used to carry out Flush + Reload attack. The attack was performed on Ubuntu 18.04 LTS Operating system. The two processes that represented the attacker and the victim were unprivileged processes. The processor on which the attack was carried out is Intel i9-9880H. The sizes of LLC and the physical memory are 16 MB and 32 GB respectively. We started by downloading all the content of the GitHub repository that includes the code to carry out the attack [4]. Next, we executed preliminary steps to set up the environment needed to run the code. The preliminary steps included installing all dependencies that are required to carry out the attack. A list of dependencies can be found in the GitHub repository [4]. After successfully setting up the environment, we had to configure the code to probe specific cache lines of shared data between the attacker and the victim. We did so by running an automation tool that gave us four cache line addresses as seen in figure 5 below. We proceeded by saving the cache line addresses in a specific text file called "links-demo.probes" to be used later while running the spy tool. In the next section, we will discuss a step-by-step process of running the attack using unprivileged user account.

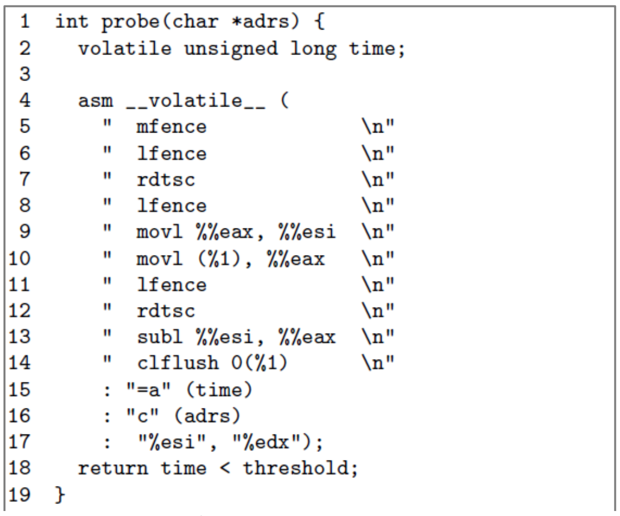


```
A:0xbe710
B:0xbedf0
C:0xbe820
D:0xc1f90
```

Fig. 5. Addresses returned by automation tool.

C. Step-by-step attack:

In this section, we give a novel procedure of carrying out Flush + Reload attack. Consider a system of the specifics mentioned earlier with two users: Alice and Scarlet. Alice has just been diagnosed with an illness and is using Wikipedia to research it. Scarlet on the other hand knows that Alice was just diagnosed. Scarlet knows that Alice is going to visit one of, say, 100 Wikipedia pages to find description about her illness. When Scarlet tries to render one of those web pages that Alice has visited using Links browser, the OS maps Scarlet's page tables to use the same copy of physical memory that Alice is using which is a practical utilization of the concept of page sharing explained earlier to improve performance of the system by avoiding to use duplicate physical memory space. However, page sharing in this situation poses a security risk in the form of a potential leak of information to the attacker through monitoring access time to the shared data. Scarlet exploits that security risk by training on the victim's system. The training comes in the form of generating probe sequences of all 100 Wikipedia pages that Scarlet think Alice might visit. The attack is most successful if the training is done on the victim's machine. The reliability drops by nearly 20% if the training is done on a system that is not the victim's system. The training can be done before or after the attack has taken place. After Scarlet's training is done, she may start the spy tool which performs the following steps (the code in figure 6 below implements the following steps):



```
int probe(char *adrs) {
  volatile unsigned long time;

  asm __volatile__ (
    "  mfence              \n"
    "  lfence              \n"
    "  rdtsc               \n"
    "  lfence              \n"
    "  movl %%eax, %%esi   \n"
    "  movl (%1), %%eax    \n"
    "  lfence              \n"
    "  rdtsc               \n"
    "  subl %%esi, %%eax   \n"
    "  clflush 0(%1)       \n"
    : "=a" (time)
    : "c" (adrs)
    :  "%esi", "%edx");
  return time < threshold;
}
```

Fig. 6. Spy tool code for probing the victim.

1) Scarlet's spy tool waits until Alice has visited one of the pages from the known set of Wikipedia pages that Scarlet has trained on.

2) Scarlet's spy tool monitors the cache lines identified earlier with the help of HTML parsing functions of Links browser.

3) Scarlet's spy tool flushes those cache lines (using clflush instruction) and then waits for a specific amount of time for Alice to browse more.

4) Alice browses more and Scarlet's spy tool monitors the time it takes to access the data that has been flushed out of the cache:

a) If the access time is large, then the data is brought from physical memory for the first time and Alice has NOT triggered any probe recently to bring the data to the cache.

b) If the access time is small enough, then Alice has recently triggered one of the four HTML parsing functions/probes. The spy tool then concatenates the letter associated with that specific probe to the probe sequence to be later analyzed (four HTML parsing functions and their letter association is mentioned in "Description of the attack" section).

5) After the probe sequence has been constructed, Scarlet may stop her spying tool and begin analyzing the probe sequence to identify which page Alice has visited.

The last stage of the attack is identifying which Wikipedia page that Alice has visited exactly by comparing the probe sequence Scarlet now has against all 100 training probe sequences obtained earlier. The comparison is done using Levenshtein's algorithm. The Levenshtein distance is a string metric for measuring difference between two sequences. Informally, the Levenshtein distance between two strings is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one string into the other. Upon running Levenshtein algorithm on the spy and training samples, the Wikipedia page that Alice had visited is identified immediately. The success rate of this attack is greater than 90% if the training is done on the victim's machine. The reliability of the attack drops to 70% if the training is done on a different system.

VI. Defense Techniques

Defending against side-channel attacks is difficult. Solutions that have been proposed so far pose a huge overhead in terms of performance on the processor. One proposed solution is to remove page sharing which is the main idea behind the Flush + Reload attacks. However, removing page sharing will pose the potential of having duplicate physical memory space of the same data. Another solution proposed is to remove the inclusiveness of Last Level Cache. While this solution seems to be a good defense technique against Flush + Reload attacks by disallowing a process on one core of flushing data loaded in the cache of another core by another process, it will potentially affect the entire memory hierarchy model by increasing the miss rate of data cached in separate cores. A third solution is to add a hardware counter accompanied with a trained algorithm to identify and prevent frequent flushes of LLC. However, a possible drawback of this solution is that the hardware counter can be vulnerable to malicious attacks to block supervisory processes from performing their tasks. To conclude, defense research against side-channel attacks in general, and Flush + Reload attacks to be specific, should be headed toward providing a long-lasting solution that neither affects the performance of modern processors nor does it introduce new vulnerabilities to the architecture of modern CPUs.

VII. Conclusion

In this paper, we demonstrated the Flush + Reload attack, and how it can be used to leak information to unprivileged spying processes on a separate core through only a single observation. In some instances, the information leaked through Flush + Reload attacks can be catastrophic to the victim because of the attacker's ability to compromise the secret key of GnuPG across multiple cores and across virtual machine boundaries. Future work includes testing the effectiveness of the attack on a more commonly used web browser like Google Chrome or Firefox. Hornby's research left two questions unexplored. The first is whether the Links command line browser attack can be made to work across virtual machine boundaries, and the second is whether the same attack can be made to work on processors with exclusive cache architectures, like AMD. In conclusion, along with the huge potential that side-channel attacks offer in compromising sensitive information, defense research against such attacks in general, and Flush + Reload attacks to be specific, remains a very interesting topic with a lot of questions yet to be answered.

VIII. References

[1] Y. Yarom, K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," The University of Adelaide, 2013.

[2] R. L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21, 2 (February 1978), 120–126.

[3] T. Hornby, "Side-Channel Attacks on Everyday Applications: Distinguishing Inputs with FLUSH+RELOAD," University of Calgary, 2016.

[4] Most viewed articles on Wikipedia 2013. https://web.archive.org/web/20141215053145/http://tools.wmflabs.org/wikitrends/2013.html. Accessed:2015-01-19.