FlushTime:
Towards Mitigating Flush-based Cache Attacks via Collaborating Flush Instructions and Timers on ARMv8-A

Jingquan Ge, Fengwei Zhang

Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology

9th July, 2023
1 Motivation

ARMv8-A CPU
Cache-related attack
Flush-based cache-related attack
Hardware and software defense

2 Threat Model and Assumptions

3 Our Solution: FlushTime

4 Security Analysis

5 Performance Evaluation
1 Motivation

ARMv8-A CPU
Cache-related attack
Flush-based cache-related attack
Hardware and software defense

2 Threat Model and Assumptions

3 Our Solution: FlushTime

4 Security Analysis

5 Performance Evaluation
Why study ARMv8-A CPU?

• ARMv8-A-based devices (smart phones, tablet, vehicle systems and IoT) have flooded the market.
• ARMv8-A cloud servers have begun to disrupt the data center market.
Motivation

ARMv8-A CPU

Cache-related attack

Flush-based cache-related attack

Hardware and software defense

Threat Model and Assumptions

Our Solution: FlushTime

Security Analysis

Performance Evaluation

Conclusion

Jingquan Ge, Fengwei Zhang

Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology
• Increasing types of cache-related attacks have been presented by researcher since 1990s.

• With the continuous emergence of Meltdown, Spectre and their variants, cache-related attacks have become one of the biggest threats to modern processors and operating systems.

• Flush+Reload and Flush+Flush utilize cache flush instructions to reduce the noise and improve the resolution, called flush-based cache-related attack.

• Meltdown, Spectre and most of the discovered variants are also based on Flush+Reload.
Motivation

ARMv8-A CPU

Cache-related attack

Flush-based cache-related attack

Hardware and software defense

Threat Model and Assumptions

Our Solution: FlushTime

Security Analysis

Performance Evaluation

Conclusion
Why study Flush-based cache-related attack?

- Attackers only need to know target virtual addresses, physical address mapping is not required.
- Although the flush instructions greatly reduce the threshold of cache attacks, Prohibiting the flush instructions in user space is not feasible.
- It is an attractive topic to ensure the availability of cache flush instructions in user space while avoiding the security vulnerabilities posed by them.
1 Motivation

ARMv8-A CPU
Cache-related attack
Flush-based cache-related attack

Hardware and software defense

2 Threat Model and Assumptions

3 Our Solution: FlushTime

4 Security Analysis

5 Performance Evaluation
Defenses and their shortcomings

- Modifications to the hardware architecture cannot be deployed on existing devices.
- Software runtime defenses cannot cover all flush-based cache-related attacks and may bring significant performance loss.
- Browser defense countermeasures disable high resolution time API, which is not feasible in the operating system.
1 Motivation

2 Threat Model and Assumptions
   Security Assumptions
   Attacker capabilities

3 Our Solution: FlushTime

4 Security Analysis

5 Performance Evaluation

6 Conclusion
1 Motivation

2 Threat Model and Assumptions
   Security Assumptions
   Attacker capabilities

3 Our Solution: FlushTime

4 Security Analysis

5 Performance Evaluation

6 Conclusion
Security Assumptions

• Attackers can execute her code on the same machine with the victim process.
• There is shared memory between the attacker process and the victim process.
• Attackers do not have the root privilege and cannot use other attack methods to tamper with the kernel code or escalate privileges to obtain sensitive system information.
• Attackers do not have the ability to design an effective eviction strategy.
Motivation

Threat Model and Assumptions

Security Assumptions

Attacker capabilities

Our Solution: FlushTime

Security Analysis

Performance Evaluation

Conclusion

Jingquan Ge, Fengwei Zhang
Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology
Attacker capabilities

- The attacker knows the source code and address layout of the victim process or kernel.
- Attackers can rely on the flush instructions to clean up the cache lines of the shared pages and leak information.
Motivation

Threat Model and Assumptions

Our Solution: FlushTime

Overview of FlushTime
Design of FlushTime
Relationship between Flush and Timer
Implementation of FlushTime

Security Analysis

Performance Evaluation
1 Motivation

2 Threat Model and Assumptions

3 Our Solution: FlushTime
   Overview of FlushTime
   Design of FlushTime
   Relationship between Flush and Timer
   Implementation of FlushTime

4 Security Analysis

5 Performance Evaluation
Overview of FlushTime

• FlushTime is a framework that can resist all flush-based cache-related attacks while ensuring the availability of flush instructions and generic timers on ARMv8-A.

• FlushTime utilizes the instruction/register trap mechanism of ARMv8-A to trap cache flush instructions and generic timer access into the kernel interrupt handlers.

• In the kernel space, these two handlers cooperate with each other to handle the interrupts. When a process calls a cache flush instruction, the time resolution obtained from the generic timer will be temporarily reduced.
Architecture of FlushTime

Figure 1: The architecture of FlushTime on ARMv8-A Linux. \textit{LRDCounter} represents the low resolution delay counter, which counts the number of \texttt{context\_switch()} in low resolution state.
1 Motivation

2 Threat Model and Assumptions

3 Our Solution: FlushTime
   Overview of FlushTime
   Design of FlushTime
   Relationship between Flush and Timer
   Implementation of FlushTime

4 Security Analysis

5 Performance Evaluation
Design of FlushTime

- Modify the default configuration of Linux to trap flush instructions and generic timer into the EL1 level (kernel space).
- Modify the interrupt handler of the flush instructions.
- Modify the context switch of the process.
- Modify the interrupt handler for accessing the generic timer.
Handler Algorithms

Algorithm 1: user_cache_maint_handler() in FlushTime

**Input:** Virtual address to be flushed: vir_addr;
**Output:** Low resolution delay counter: LRDCounter;
1. Flush the cache line of vir_addr;
2. Store NumCSLR into LRDCounter;
3. Return to EL0;

Algorithm 2: cntvet_read_handler() in FlushTime

**Input:** Low resolution delay counter: LRDCounter;
**Output:** Time read from CNTVCT_EL0: CNTVCTime;
1. Read CNTVCT_EL0 into CNTVCTime;
2. if (LRDCounter! = 0) then
   3. Reduce the resolution of CNTVCTime by NumLRMB bits;
   4. end
   5. else
   6. Keep the high resolution of CNTVCTime;
   7. end
8. Return CNTVCTime to EL0;

(a) Algorithm of flush instruction handler. (b) Algorithm of timer access handler

Figure 2: Algorithms of the two handlers
1 Motivation

2 Threat Model and Assumptions

3 Our Solution: FlushTime
   - Overview of FlushTime
   - Design of FlushTime
   - Relationship between Flush and Timer
   - Implementation of FlushTime

4 Security Analysis

5 Performance Evaluation
Motivation

Threat Model and Assumptions

Our Solution: FlushTime

Security Analysis

Performance Evaluation

Conclusion

Relationship between flush instructions and generic timer

![Diagram showing the relationship between flush instructions and the generic timer when FlushTime is enabled.]

**Figure 3:** Relationship between flush instructions and generic timer when FlushTime is enabled. 

*NumCSLR* is the number of *context_switch()* in low resolution state.
Motivation

2 Threat Model and Assumptions

3 Our Solution: FlushTime
   Overview of FlushTime
   Design of FlushTime
   Relationship between Flush and Timer
   Implementation of FlushTime

4 Security Analysis

5 Performance Evaluation
Implementation of FlushTime

- Modify Linux boot function `smp_init()`.
- Modify the flush handler `user_cache_maint_handler()`.
- Modify process scheduling function `context_switch()`.
- Modify generic timer handler `cntvct_read_handler()`.
Motivation

Threat Model and Assumptions

Our Solution: FlushTime

Security Analysis

Performance Evaluation

Conclusion

Selection of NumLRMB and NumCSLR

Attack Results

Jingquan Ge, Fengwei Zhang
Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology
FlushTime
## Motivation

## Threat Model and Assumptions

## Our Solution: FlushTime

## Security Analysis

### Selection of $Num_{LRMB}$ and $Num_{CSLR}$

## Performance Evaluation

## Conclusion
Relationship between the two parameters and attack success rate

(a) The relationship between the parameter \( \text{NumLRMB} \) and attack success rate on TaiShan 200 server. Both the two attacks are multi-process attacks that execute 100 processes. The red line represents the optimal value of \( \text{NumLRMB} \).

(b) The relationship between the parameter \( \text{NumCSLR} \) and attack success rate on TaiShan 200 server. Both the two attacks are multi-process attacks that execute 100 processes. The red line represents the optimal value of \( \text{NumCSLR} \).

Figure 4: The relationship between the two parameters and attack success rate.
Further Study on \textit{NumLRMB} and \textit{NumCSLR}

\textbf{Table 1:} Selection of \textit{NumLRMB} and \textit{NumCSLR} on different hardware platforms.

<table>
<thead>
<tr>
<th>Platform</th>
<th># ARMv8-A cores</th>
<th>NumLRMB</th>
<th>NumCSLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaiShan 200</td>
<td>96</td>
<td>12</td>
<td>96</td>
</tr>
<tr>
<td>Raspberry Pi 4B</td>
<td>4</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>ZCU102</td>
<td>4</td>
<td>15</td>
<td>4</td>
</tr>
</tbody>
</table>
1 Motivation

2 Threat Model and Assumptions

3 Our Solution: FlushTime

4 Security Analysis
   Selection of NumLRMB and NumCSLR
   Attack Results

5 Performance Evaluation

6 Conclusion
Flush+Reload attack results

(a) Flush+Reload attack on original Linux without any defenses.

(b) Flush+Reload attack when flush instructions and generic timer are trapped into EL1.

(c) Flush+Reload attack when FlushTime is enabled (NumCSLR==96). The resolution of the generic timer is reduced by 12 bits in real time (NumLRMB==12).

Figure 5: Flush+Reload attacks on different system setups. It is a multi-process attack that execute 100 processes. The depth of the color corresponds to the number of cache hits in 1000 AES T-Table encryptions.
Flush+Flush attack results

(a) Flush+Flush attack on original Linux without any defenses.

(b) Flush+Flush attack when flush instructions and generic timer are trapped into EL1.

(c) Flush+Flush attack when FlushTime is enabled ($NumCSLR==96$). The resolution of the generic timer is reduced by 12 bits in real time ($NumLRMB==12$).

Figure 6: Flush+Flush attacks on different system setups. It is a multi-process attack that execute 100 processes. The depth of the color corresponds to the number of cache hits in 1,000 AES T-Table encryptions.
Spectre-BTB attack results

(a) Spectre-BTB attack on original Linux without any defenses.
(b) Spectre-BTB attack when flush instructions and general timer are trapped into EL1.
(c) Spectre-BTB attack when FlushTime is enabled (NumCSLR==96). The resolution of the generic timer is reduced by 8 bits in real time (NumLRMB==8).
(d) Spectre-BTB attack when FlushTime is enabled (NumCSLR==96). The resolution of the generic timer is reduced by 12 bits in real time (NumLRMB==12).

Figure 7: Spectre-BTB attacks on different system setups. It is a multi-process attack that execute 100 processes. ‘?’ represents a character that has not been cracked.
1 Motivation

2 Threat Model and Assumptions

3 Our Solution: FlushTime

4 Security Analysis

5 Performance Evaluation
   Performance of instructions and APIs
   System Performance (UnixBench)
   User Application Performance (SPEC_CPU 2017)

6 Conclusion
1 Motivation

2 Threat Model and Assumptions

3 Our Solution: FlushTime

4 Security Analysis

5 Performance Evaluation

   Performance of instructions and APIs
   System Performance (UnixBench)
   User Application Performance (SPEC_CPU 2017)

6 Conclusion
Flush Instructions, Generic Timer and Time API

Table 2: Average time delay of calling flush instructions, calling API and accessing generic timer.

<table>
<thead>
<tr>
<th>Instruction, API or timer</th>
<th>Original Linux time delay (cycle)</th>
<th>FlushTime enabled time delay (cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTVCT_EL0</td>
<td>12.14</td>
<td>27.73 (127%)</td>
</tr>
<tr>
<td>clock_gettime()</td>
<td>103.02</td>
<td>115.29 (19%)</td>
</tr>
<tr>
<td>DC CIVAC</td>
<td>38.21</td>
<td>28.15 (-26%)</td>
</tr>
<tr>
<td>DC CVAU</td>
<td>69.33</td>
<td>26.98 (-61%)</td>
</tr>
<tr>
<td>DC CVAC</td>
<td>69.58</td>
<td>28.11 (-60%)</td>
</tr>
</tbody>
</table>
Motivation

Threat Model and Assumptions

Our Solution: FlushTime

Security Analysis

Performance Evaluation

Conclusion

Performance of instructions and APIs

System Performance (UnixBench)

User Application Performance (SPEC_CPU 2017)
UnixBench results

Figure 8: Evaluation results of UnixBench.
# Motivation

# Threat Model and Assumptions

# Our Solution: FlushTime

# Security Analysis

## Performance Evaluation
- Performance of instructions and APIs
- System Performance (UnixBench)
- User Application Performance (SPEC_CPU 2017)

# Conclusion
SPEC_CPU 2017 results

Figure 9: SPEC2017 benchmark results.
1 Motivation
2 Threat Model and Assumptions
3 Our Solution: FlushTime
4 Security Analysis
5 Performance Evaluation
6 Conclusion
• FlushTime does not need to modify the hardware, which is easy to deploy on existing devices.

• FlushTime not only partially prevents instructions and timers from being maliciously exploited by flush-based cache-related attacks, but also ensures their availability in a normal system.

• Security and performance evaluation on the real hardware platform shows that FlushTime is not only more secure than other software solutions, but also has the lowest performance overhead.
Q&A