Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications

Hasan Hassan†  Yahya Can Tuğrul‡†  Jeremie S. Kim†  Victor van der Veen§
Kaveh Razavi†  Onur Mutlu†

† ETH Zürich  ‡ TOBB University of Economics & Technology  § Qualcomm Technologies Inc.

Published in MICRO 2021

Aleksandar Terzić
Seminar in Computer Architecture
14.04.2022
Executive Summary

• The RowHammer vulnerability in DRAM is a critical threat to system security

• Problem: To protect against RowHammer, DRAM vendors commit to security-through-obscurity
  - Undocumented, proprietary, on-die mitigations commonly called Target Row Refresh (TRR)

• Goal: Develop a methodology to assess the security guarantees of modern TRR mechanisms

• Methodology: U-TRR
  - Key idea: Data retention failures in DRAM enable a side channel that leaks information on how TRR refreshes potential victim rows.

Exploiting the DRAM rowhammer bug to gain kernel privileges

- 45 DDR4 modules from three major DRAM manufacturers
- Each module found to be vulnerable to new custom TRR-aware access patterns

SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript

Finn de Risi
ETH Zurich
VU Amsterdam

Paul Fierens
VU Amsterdam

Cristiano Giuffrida
VU Amsterdam

Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript

Daniel Gruss, Clémentine Maurice†, and Stefan Mangard
Graz University of Technology, Austria

Kaveh Razavi
ETH Zurich
1: DRAM Organization and Operation
DRAM Organization

- DRAM is organized in a hierarchical manner
- DRAM cells are organized in 2-D arrays, called subarrays
- Bits are stored as charge in DRAM cell capacitors
Accessing DRAM

1. **Activate** a DRAM row, bring data to row buffer
2. **Read** data from row buffer
3. Close the DRAM row and **precharge** the bitlines
Dynamic-RAM

• DRAM cell charge leaks over time
• To ensure data is not lost, rows must be periodically refreshed
2: RowHammer
RowHammer

- Rapidly activating a row can induce bit flips in neighboring rows

**Single-location RowHammer attack**
Hammering Strategies

- Two hammering strategies most often encountered in literature
- Paper uses the more effective double-sided hammering strategy
Technology Scaling

- RowHammer is **getting worse** as technology scales
- Average number of hammers needed to induce a bit flip is **decreasing**

J. Kim et. al. “Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques“, ISCA 2020
The Issue with TRR

• What we know:
  • TRR refreshes DRAM rows upon detecting a RowHammer attack

• Undisclosed information:
  • Where in the hierarchy is TRR implemented?
  • How does TRR detect aggressor/victim rows?
  • How often does it refresh victim rows?
  • How many victim rows can it refresh at a time?

• Impossible to analyze TRR security guarantees
3: U-TRR

A new methodology for uncovering proprietary TRR mechanisms
Overview of U-TRR

Two key components:

1. **RowScout (RS)**
   - Profile row data retention times
   - Find a set of profiled DRAM rows which fulfill desired conditions
   - Return a list of retention profiled rows (RPRs)

2. **TRR Analyzer (TRR-A)**
   - Use RS-provided rows to detect TRR-induced refreshes
   - Build an understanding of underlying TRR mechanism
     - Allows for new TRR-aware access patterns
RowScout

Key goals:

1. Find rows with **consistent retention times**
   - DRAM cells may exhibit **variable retention time**

2. Find **multiple rows at configurable distances, with the same retention time T**
   - Observe a group of rows at a given distance to each other
   - Infer whether the TRR mechanism is capable of refreshing several rows at once
RowScout Operation

1. Write data pattern
2. Wait for time $T$
3. Read out data

Find DRAM rows with retention time $T$

row addresses

Combine rows to match the group layout

candidate row groups

Are the candidates enough?

NO

increase $T$

NO

increase $T$

YES

Verify retention time consistency

YES

row groups

ENOugh row groups pass?

ENOugh row groups pass?

YES

Retention Profiled Rows (RPR)

NO

5. Increase $T$

6. Verify retention time consistency

Row layout R-R-R
TRR-Analyzer Operation

High-level operation:
1. **Initialize** victim and aggressor rows. Wait for time T/2
2. **Hammer** aggressor rows
3. **Issue refreshes**
   - TRR refreshes piggyback off of regular refresh commands
4. **Monitor data retention** in victim rows
   - If correct data was retained, victim row was targeted by either a regular refresh or a TRR refresh
   - Easy to distinguish between the two as regular refreshes are periodic
Example Experiment

• Uncovering whether TRR can refresh several rows at once:

1. Find group of rows of layout RRR-RRR
2. Initialize rows, wait for time T/2
3. Hammer aggressor row
   • Don’t hammer often enough to induce bit flips, but enough to (hopefully) trigger TRR
4. Issue REF command
   • If REF is TRR-capable, it will target some of the victim rows
5. Read out victim row data at time T
   • Victim rows which exhibit bit flips were not targeted by TRR
TRR-Analyzer Operation, Detailed

1. Initialize V and A
2. Reset TRR’s internal state
3. Hammer A [and D]
4. Issue REFs
5. Read V and check for bit flips

Experiment Configuration:
- aggressor (A) row addr.
- dummy (D) row count
- hammering mode
- number of rounds
- A/D hammer counts
- REF count
- ...

V: victim (RS-provided) rows
A: aggressor rows
D: dummy rows
4: Uncovering and Circumventing TRR
Experimental Setup: SoftMC

- FPGA-based DRAM testing infrastructure
- Precise control over DDR commands issued to DRAM module
- Precise operating temperature control
  - Data retention time temperature dependent
  - All experiments run at 85 degrees C
Experimental Setup (cont.)

- Investigated 45 DDR4 modules from 3 major DRAM vendors (Vendor A, B, C)
- Fabrication dates range from late 2016 to late 2020
Vendor A’s TRR Implementation

- TRR-capable refreshes occur periodically
- Two different types of TRR-capable refreshes (TREF₁, TREF₂)

Vendor A’s TRR keeps track of aggressor rows using a 16-entry counter table

- TREF₁ refreshes neighbors of highest-count table entry
- TREF₂ iterates through all of the table entries circularly

<table>
<thead>
<tr>
<th>Row ID</th>
<th>Counter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TREF₂
Circumventing Vendor A’s TRR

**Observation:** Upon insertion of new row, TRR evicts the lowest-count entry from the table

**Key Idea:** Ensure aggressor rows evicted from the table before a REF command

- $A_i$: aggressor rows
- $D_i$: dummy rows

Diagram:

```
REF → ACT ([A₁, A₂]) → ACT(D₁) → ACT(D₂) → ⋮ → ACT(D₁₆) → REF

N times                       N+1 times                       N+1 times                       N+1 times

[A₁, A₂] not refreshed by TRR
```
Vendor B’s TRR Implementation

- TRR-capable refreshes occur periodically
- A single type of TRR-capable refresh (TREF)

TRR probabilistically samples the address of an activated row

TRR keeps track of only the most recently sampled row across all banks in a DRAM chip
Circumventing Vendor B’s TRR

Key Idea: Maximize the probability of sampling a dummy row

1. Run a double-sided RowHammer attack
2. Hammer dummy row as often as possible before next TREF

\[ \text{TREF} \rightarrow \text{ACT ([A}_1, \text{A}_2]) \rightarrow \text{ACT(D}_1) \rightarrow \text{TREF} \]

- ACT ([A}_1, \text{A}_2]) \text{ACT(D}_1\] \text{times}
- [A}_1, \text{A}_2] not refreshed by TRR

\[ \hspace{1cm} \text{N times} \quad \text{M times} \]
5: Experimental Analysis
Vulnerable Rows

• System-level attacks need to target vulnerable rows
• Over 99.9% of rows vulnerable in 21 out of 45 modules
• Vendor B has RowHammer resistant rows
  • Large number of hammers needed to induce a bit flip
• Vendor C has a unique row organization
  • Pairs of rows isolated from each other
  • Additionally, modules C0-6 have RowHammer resistant rows
Error-Correcting Codes

- On-die SECDED ECC corrects 1 error per 8-byte data chunk
- New access patterns capable of inducing 3 and more errors per chunk
- Traditional ECC as a protection mechanism is not enough
- More sophisticated ECC schemes incur large overheads

Distribution of 8-byte data chunks with different RowHammer bit flip counts in a single DRAM bank from each module
## Overview of Results

<table>
<thead>
<tr>
<th>Module</th>
<th>Date (yy-ww)</th>
<th>Chip Density (Gbit)</th>
<th>Organization</th>
<th>HC(_{first})†</th>
<th>Version</th>
<th>Aggressor Detection</th>
<th>Aggressor Capacity</th>
<th>Per-Bank TRR</th>
<th>TRR-to-REF Ratio</th>
<th>Neighbors Refreshed</th>
<th>% Vulnerable DRAM Rows†</th>
<th>Max. Bit Flips per Row per Hammer†</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>19-50</td>
<td>8</td>
<td>1 16 8</td>
<td>16K</td>
<td>A(_{TRR1})</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>73.3%</td>
<td>1.16</td>
</tr>
<tr>
<td>A1-5</td>
<td>19-36</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
<td>A(_{TRR1})</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.2% - 99.4%</td>
<td>2.32 - 4.73</td>
</tr>
<tr>
<td>A6-7</td>
<td>19-45</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
<td>A(_{TRR1})</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.3% - 99.4%</td>
<td>2.12 - 3.86</td>
</tr>
<tr>
<td>A8-9</td>
<td>20-07</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-14K</td>
<td>A(_{TRR1})</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>74.6% - 75.0%</td>
<td>1.96 - 2.96</td>
</tr>
<tr>
<td>A10-12</td>
<td>19-51</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-13K</td>
<td>A(_{TRR1})</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>74.6% - 75.0%</td>
<td>1.48 - 2.86</td>
</tr>
<tr>
<td>A13-14</td>
<td>20-31</td>
<td>8</td>
<td>1 8 16</td>
<td>11K-14K</td>
<td>A(_{TRR2})</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>94.3% - 98.6%</td>
<td>1.53 - 2.78</td>
</tr>
<tr>
<td>B0</td>
<td>18-22</td>
<td>4</td>
<td>1 16 8</td>
<td>44K</td>
<td>B(_{TRR1})</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>2.13</td>
</tr>
<tr>
<td>B1-4</td>
<td>20-17</td>
<td>4</td>
<td>1 16 8</td>
<td>159K-192K</td>
<td>B(_{TRR1})</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>23.3% - 51.2%</td>
<td>0.06 - 0.11</td>
</tr>
<tr>
<td>B5-6</td>
<td>16-48</td>
<td>4</td>
<td>1 16 8</td>
<td>44K-50K</td>
<td>B(_{TRR1})</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>1.85 - 2.03</td>
</tr>
<tr>
<td>B7</td>
<td>19-06</td>
<td>8</td>
<td>2 16 8</td>
<td>20K</td>
<td>B(_{TRR1})</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>31.14</td>
</tr>
<tr>
<td>B8</td>
<td>18-03</td>
<td>4</td>
<td>1 16 8</td>
<td>43K</td>
<td>B(_{TRR1})</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>2.57</td>
</tr>
<tr>
<td>B9-12</td>
<td>19-48</td>
<td>8</td>
<td>1 16 8</td>
<td>42K-65K</td>
<td>B(_{TRR2})</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>36.3% - 38.9%</td>
<td>16.83 - 24.26</td>
</tr>
<tr>
<td>B13-14</td>
<td>20-08</td>
<td>4</td>
<td>1 16 8</td>
<td>11K-14K</td>
<td>B(_{TRR3})</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/2</td>
<td>4</td>
<td>99.9%</td>
<td>16.20 - 18.12</td>
</tr>
<tr>
<td>C0-3</td>
<td>16-48</td>
<td>4</td>
<td>1 16 8</td>
<td>137K-194K</td>
<td>C(_{TRR1})</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>1.0% - 23.2%</td>
<td>0.05 - 0.15</td>
</tr>
<tr>
<td>C4-6</td>
<td>17-12</td>
<td>8</td>
<td>1 16 8</td>
<td>130K-150K</td>
<td>C(_{TRR1})</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>7.8% - 12.0%</td>
<td>0.06 - 0.08</td>
</tr>
<tr>
<td>C7-8</td>
<td>20-31</td>
<td>8</td>
<td>1 8 16</td>
<td>40K-44K</td>
<td>C(_{TRR1})</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>39.8% - 41.8%</td>
<td>9.66 - 14.36</td>
</tr>
<tr>
<td>C9-11</td>
<td>20-31</td>
<td>8</td>
<td>1 8 16</td>
<td>42K-53K</td>
<td>C(_{TRR2})</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>99.7%</td>
<td>9.30 - 32.64</td>
</tr>
<tr>
<td>C12-14</td>
<td>20-46</td>
<td>16</td>
<td>1 8 16</td>
<td>6K-7K</td>
<td>C(_{TRR3})</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/8</td>
<td>2</td>
<td>99.9%</td>
<td>4.91 - 12.64</td>
</tr>
</tbody>
</table>
Conclusion

• The RowHammer vulnerability in DRAM is a critical threat to system security
• DRAM vendors employ proprietary protection mechanisms commonly called TRR
• U-TRR enables the analysis of TRR security guarantees, and it finds that...

SECURITY BY OBSCURITY IS NOT A GOOD IDEA
6: DISCUSSION
Strengths

• First work to uncover proprietary TRR implementations
  • Enables security analysis of TRR
• Proposed methodology is simple and highly effective
• A convincing demonstration of the methodology’s power via new RowHammer access patterns
• Demonstrates that ECC is not enough as a protection mechanism
• An extensive and detailed work
• An experimental demonstration of how poor of an approach security by obscurity is
Weaknesses

• Does not propose new protection mechanisms
• Could not find an effective attack on $B_{\text{TRR2}}$
• Goes into a lot of detail, but does not fully explain everything
• Some minor inconsistencies
  • E.g. in 5.3, claims to reverse engineer logical-physical row mapping via a double-sided RowHammer attack, but then describes a single-location attack
• Repetitive writing style
Discussion Points (1):

What is a fundamental weakness of the modern TRR implementations?

- Fully implemented inside DRAM
- No across-the-stack communication
- Can we do better by co-designing DRAM and the memory controller?
Discussion Points (2):

Are TRR-like solutions the best approach?

Other possible avenues:
• Fundamentally more secure circuit design (Remember Vendor C)
• Increased refresh rate
• Isolation of sensitive data
• Throttling row activations
• Something else?

Potential advantages/drawbacks?
Discussion Points (3):

Could other issues arise as technology continues to scale?

- Read disturbance errors have been reported in MLC NAND Flash memory
- Phase-Change-Memory promising for emerging in-memory computing systems
  - Current research on enabling many-level cells
  - As all technology does, so too will PCM be scaled down
  - Should we already now start thinking about its security?
Recommended Reading

The RowHammer Problem
and Other Issues We May Face as Memory Becomes Denser

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
https://people.inf.ethz.ch/omutlu
Thank you for your attention!
Vendor C’s TRR Implementation

- TRR-capable refreshes occur periodically
- A single type of TRR-capable refresh (TREF)

TRR detects an aggressor row only among the first 2k activations after TREF

The earlier a row is activated, the more likely it is to be sampled by TRR
Circumventing Vendor C’s TRR

Key Idea: Maximize the probability of sampling a dummy row

1. Hammer a dummy row
2. Run a double-sided RowHammer attack

\[ \text{TREF} \rightarrow \text{ACT}(D_1) \text{ \(N\) times} \rightarrow \text{ACT}([A_1, A_2]) \text{ \(M\) times} \rightarrow \text{TREF} \]

\([A_1, A_2]\) not refreshed by TRR
### More Details

Mention the table but don’t go into it

<table>
<thead>
<tr>
<th>Module</th>
<th>Date (yy-ww)</th>
<th>Chip Density (Gbit)</th>
<th>Organization</th>
<th>( H_{C, \text{first}} \uparrow )</th>
<th>Version</th>
<th>Aggressor Detection</th>
<th>Aggressor Capacity</th>
<th>Per-Bank TRR</th>
<th>TRR-to-REF Ratio</th>
<th>Neighbors Refreshed</th>
<th>% Vulnerable DRAM Rows ( \dagger )</th>
<th>Max. Bit Flips per Row per Hammer ( \dagger )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>19-50</td>
<td>8</td>
<td>1 16 8</td>
<td>16K</td>
<td>( A_{TRR,1} )</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>73.3%</td>
<td>1.16</td>
</tr>
<tr>
<td>A1-5</td>
<td>19-36</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
<td>( A_{TRR,1} )</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.2% - 99.4%</td>
<td>2.32 - 4.73</td>
</tr>
<tr>
<td>A6-7</td>
<td>19-45</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
<td>( A_{TRR,1} )</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.2% - 99.4%</td>
<td>2.12 - 3.86</td>
</tr>
<tr>
<td>A8-9</td>
<td>20-07</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-14K</td>
<td>( A_{TRR,1} )</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>74.6% - 75.0%</td>
<td>1.96 - 2.96</td>
</tr>
<tr>
<td>A10-12</td>
<td>19-51</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-15K</td>
<td>( A_{TRR,1} )</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>74.6% - 75.0%</td>
<td>1.48 - 2.86</td>
</tr>
<tr>
<td>A13-14</td>
<td>20-31</td>
<td>8</td>
<td>1 8 16</td>
<td>11K-14K</td>
<td>( A_{TRR,2} )</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>94.3% - 98.6%</td>
<td>1.53 - 2.78</td>
</tr>
<tr>
<td>B0</td>
<td>18-22</td>
<td>4</td>
<td>1 16 8</td>
<td>44K</td>
<td>( B_{TRR,1} )</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>2.13</td>
</tr>
<tr>
<td>B1-4</td>
<td>20-17</td>
<td>4</td>
<td>1 16 8</td>
<td>159K-192K</td>
<td>( B_{TRR,1} )</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>23.3% - 51.2%</td>
<td>0.06 - 0.11</td>
</tr>
<tr>
<td>B5-6</td>
<td>16-48</td>
<td>4</td>
<td>1 16 8</td>
<td>44K-150K</td>
<td>( B_{TRR,1} )</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>1.85 - 2.03</td>
</tr>
<tr>
<td>B7</td>
<td>19-06</td>
<td>8</td>
<td>2 16 8</td>
<td>20K</td>
<td>( B_{TRR,1} )</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>31.14</td>
</tr>
<tr>
<td>B8</td>
<td>18-03</td>
<td>4</td>
<td>1 16 8</td>
<td>43K</td>
<td>( B_{TRR,1} )</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>2.57</td>
</tr>
<tr>
<td>B9-12</td>
<td>19-48</td>
<td>8</td>
<td>1 16 8</td>
<td>42K-65K</td>
<td>( B_{TRR,2} )</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>36.3% - 38.9%</td>
<td>16.83 - 24.26</td>
</tr>
<tr>
<td>B13-14</td>
<td>20-08</td>
<td>4</td>
<td>1 16 8</td>
<td>118K-14K</td>
<td>( B_{TRR,3} )</td>
<td>Sampling-based</td>
<td>1</td>
<td>✓</td>
<td>1/2</td>
<td>4</td>
<td>99.9%</td>
<td>16.20 - 18.12</td>
</tr>
<tr>
<td>C0-3</td>
<td>16-48</td>
<td>4</td>
<td>1 16 x8</td>
<td>137K-194K</td>
<td>( C_{TRR,1} )</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>1.0% - 23.2%</td>
<td>0.05 - 0.15</td>
</tr>
<tr>
<td>C4-6</td>
<td>17-12</td>
<td>8</td>
<td>1 16 x8</td>
<td>130K-150K</td>
<td>( C_{TRR,1} )</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>7.8% - 12.0%</td>
<td>0.06 - 0.08</td>
</tr>
<tr>
<td>C7-8</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>40K-44K</td>
<td>( C_{TRR,1} )</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>39.8% - 41.8%</td>
<td>9.66 - 14.36</td>
</tr>
<tr>
<td>C9-11</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>42K-53K</td>
<td>( C_{TRR,2} )</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/9</td>
<td>2</td>
<td>99.9%</td>
<td>9.30 - 32.04</td>
</tr>
<tr>
<td>C12-14</td>
<td>20-46</td>
<td>16</td>
<td>1 8 x16</td>
<td>6K-7K</td>
<td>( C_{TRR,3} )</td>
<td>Mix</td>
<td>Unknown</td>
<td>✓</td>
<td>1/8</td>
<td>2</td>
<td>99.9%</td>
<td>4.91 - 12.64</td>
</tr>
</tbody>
</table>

\( \dagger \) Denotes vulnerability based on observed TRR behavior.