

Eccsmith: Turning the Blacksmith Rowhammer Fuzzer Into an ECC Validator

Patricia Norton

Student ID: 2177660

MSci Computer Science FT

60 Credits

Supervisor: David Oswald

Word Count: 6,556

Abstract

Error Correction Code memory (ECC) is a widespread hardware feature which is capable of detecting and correcting bit flip errors in RAM, and is relied upon for the stability of important systems. However, there is currently no easy and free method of checking if ECC is even functioning correctly on a system which claims to implement it. This report proposes a solution to this problem in the form of Eccsmith, a program which manually injects bit flip errors into RAM using the fault injection exploit Rowhammer, then detects whether or not these errors are corrected by ECC. Eccsmith is a modification of the pre-existing Rowhammer fuzzer Blacksmith, making it capable of functioning on DDR4 RAM which implements Target Row Refresh (TRR). In addition, Eccsmith makes several improvements to the ease of use of Blacksmith's Rowhammer fuzzing, bringing it closer to the goal of being easily usable by anyone seeking to validate the functionality of ECC on their system.

Git Repository

<https://github.com/patricia-9000/eccsmith>

Acknowledgements

I would like to thank the authors and developers of Blacksmith - Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi - and the developers of its JSON config branch, Jeremy Boy and Luca Wilke. I would also like to thank my supervisor David Oswald for his guidance with this project, and Jacqui Henes for providing the initial idea for the project.

Contents

1	Introduction	1
1.1	ECC	1
1.2	Rowhammer	2
1.3	Contributions	2
2	Literature Review	3
2.1	Rasdaemon	3
2.2	TRR	3
2.3	DRAMA	4
3	Requirements	5
3.1	Problem Statement	5
3.2	System Requirements	5
3.3	Software Requirements	6
4	Legal and Professional Issues	7
4.1	Using an Exploit for Good	7
4.2	Adapting Others' Work	7
5	Methodology	8
5.1	ECC Validation	8
5.2	Simplified Configuration	9
6	Results and Evaluation	11
6.1	Proof of Concept	11
6.2	Implementing ECC Validation	12
6.3	Implementing Simplified Configuration	14
6.4	Evaluation	15
7	Conclusion	17
7.1	Summary	17
7.2	Suggestions for Future Work	17
8	References	18
A	Appendices	20

Introduction

1.1 ECC

Bit flip errors in RAM can be caused by a variety of random environmental factors, such as electromagnetic interference and cosmic rays. Though unlikely to occur, these errors are responsible for a proportion of all system crashes, which can be major catastrophes when they happen on servers or other important systems.[1] Therefore in order to prevent these errors, such systems often implement error correction code memory, or ECC. This is a mechanism which detects and corrects bit flips, using a type of checksum referred to as a correction code. Using ECC, any data which gets stored in RAM has a correction code generated from it, which is then also stored separately in RAM. Then when that data gets accessed later, its correction code is also read out, which is then compared to a newly-generated correction code from the read data, to check if any bit flip errors are present in it. This mechanism is capable of correcting single-bit errors and resuming execution of the relevant process as normal, by using the correction code to deduce which bit was flipped. But in the much more unlikely event that a multiple-bit error occurs, it is usually not possible to know which bits were flipped, so the error may not be able to be corrected. The availability of ECC is hardware-dependent, as both the RAM modules and the motherboard must be compatible with this mechanism.[2]

Manufacturers of systems with ECC implemented usually check that it is working properly using a feature built in to the motherboard called ECC injection. This feature allows for the manual insertion of memory errors and the testing of ECC's response to these errors. But this feature is usually permanently disabled by manufacturers once they have run their own tests.[2] This means that in most cases, end-users have no easy way of verifying that ECC is actually functioning on their systems. In other words, they will only know if it is working or not if a bit flip error occurs naturally on their system, at which point it is already too late.

The main existing solution to this problem is the ECC Tester sold by PassMark Software.[3] This is a hardware device designed to be inserted into one of the DIMM slots of a motherboard, and then for an ECC-enabled RAM module to be inserted into the device itself. There is a button on the device which causes it to inject bit flip errors into the data read from the inserted RAM module when pressed. This allows the user to observe their system's response to these errors in a controlled environment, letting them see if ECC is working or not. The downside of the ECC Tester is that it costs \$260, making it less accessible for individuals not operating with the budget of a large company.

Beyond this option, the only other existing methods of validating ECC which some people resort to involve tampering with the hardware of the system itself, in order to cause hardware faults which produce bit flip errors.[4] Individuals seeking to validate their ECC may find this to be prohibitively difficult or risky. Therefore, there are issues with every currently existing method of validating ECC, so there is a need for an easy and free alternative.

1.2 Rowhammer

The transistors which make up RAM can leak the charge which they store, resulting in bit flip errors in the bits which they represent. Leaking happens naturally over time to the transistors in RAM, because they are so small and densely-packed. But a system's OS prevents this by periodically refreshing rows of RAM so that they are never left long enough to leak. This is done by storing the same value back into a transistor that it is already holding. And this is done in rows because bits in RAM can only ever be accessed in whole rows at a time.

Rowhammer is a software-based fault injection exploit which is capable of manually injecting bit flip errors into memory.[5] Accessing a row of RAM makes the surrounding rows slightly more likely to leak, because it disturbs the charge in their transistors. So if an "aggressor" row of RAM is repeatedly accessed enough - referred to as "hammering" - it will make the "victim" rows surrounding it leak faster than the OS refreshes them, resulting in random bit flips within those rows.

1.3 Contributions

This report presents a novel and practical use case for Rowhammer, as the means of triggering ECC in order to prove that it is functioning correctly. A practical application of this use case is also presented in the form of Eccsmith, which is a modification of the existing Rowhammer software Blacksmith.[6] Eccsmith builds upon Blacksmith by implementing the ability for it to detect and report on the ECC corrections it causes, by communicating with the hardware error event logger Rasdaemon.[7] It also builds upon Blacksmith further by automating a larger proportion of the process of configuring it to the host system. With these changes, Eccsmith attempts to solve the problem of validating ECC easily and for free.

Literature Review

2.1 Rasdaemon

When ECC corrects a bit flip error, details of the event are written to the OS's event tracking log. On Ubuntu and Debian, this is the file `/var/log/syslog`. Other hardware error events are also recorded in the same manner. In order to more easily keep track of these events, the Rasdaemon package exists on these Linux distributions. Rasdaemon reads from the event tracking log, and stores details of hardware error events in a database located at `/var/lib/rasdaemon/ras-mc_event.db`.^[7] Utilising Rasdaemon is a simple choice for how the ECC validator produced in this project should detect ECC corrections.

2.2 TRR

In response to Rowhammer's discovery in 2014, the standard for DDR4 RAM was amended to include support for a countermeasure called Target Row Refresh (TRR).^[8] This is a feature of RAM where a row is refreshed if its surrounding rows are accessed more than a certain number of times in a brief period, thus preventing that target row from leaking. This makes it impossible to perform the classic approach to Rowhammer on modern DDR4 RAM, which is widespread today.

In response to the adoption of TRR, new methods of performing Rowhammer were discovered that circumvent it - namely TRRespass in 2020,^[9] and then Blacksmith in 2022.^[6] TRRespass performs "many-sided hammering", where many aggressor rows surrounding a single victim row are hammered in varying ways, such that no single aggressor row is accessed enough times to trigger TRR. Then Blacksmith improved upon this technique with the introduction of "non-uniform" many-sided hammering, which makes the exploit effective on more DDR4 RAM modules. These are referred to as Rowhammer "fuzzers", borrowing the terminology from software fuzzing, because they generate many hammering patterns at random and try them until one is found to be effective.

There are certain difficulties inherent to creating useful software which utilises an exploit which has been widely known about for a decade. The ubiquity of TRR means that this project will be incapable of achieving its goals if a naïve approach to Rowhammer is attempted. The ECC validator which will be created should be able to function on modern RAM, therefore a Rowhammer approach which is capable of circumventing TRR must be used. However, these approaches are too complex to recreate in the time allowed for this project, so an existing implementation must be reused. Therefore, the ECC validator will be built off Blacksmith.

2.3 DRAMA

The many-sided hammering techniques employed by both TRRespass and Blacksmith involve accessing several rows of RAM which are in close physical proximity to each other. This is not something that can be achieved by normal means, because memory addresses as they are understood by programs are different to their actual corresponding physical locations inside RAM. A location in RAM is defined by six attributes:

1. The RAM channel (the bus from which a RAM module is accessed)
2. The RAM module accessed by that channel
3. The rank on that module (which of the two sides it is stored on)
4. The bank on that rank (the physical chip storing the data)
5. The row in that bank
6. The position in that row

The function which maps from addresses to physical locations is determined by the system's CPU microarchitecture, which intentionally obfuscates the relationship between them by making it so that locations which are adjacent in address space are not necessarily physically adjacent in RAM.

TRRespass and Blacksmith were made possible by the research done as part of the memory mapping reverse-engineering tool DRAMA.[10] DRAMA allows for the obtaining of previously undocumented memory address mappings via side channel analysis on memory access times. These access time discrepancies arise from the fact that when a single memory location is accessed, the whole row of RAM where it is physically stored has to be read at once. When this happens, the contents of the row are cached in the row buffer of the bank which that row belongs to. So when two memory locations are repeatedly accessed in alternation - while using `cflush` each time to prevent CPU caching - it will take longer to access them if they both belong to the same bank but to different rows within that bank, because a new row has to be read each time, rather than being read from that bank's row buffer. And conversely, if the access times are fast, then this means the two addresses likely belong to different banks altogether, since both of their rows will be read from their respective banks' row buffers on each access. After identifying whether enough address pairs belong to the same or different banks, the overall memory mapping function can be reconstructed.

While Blacksmith effectively uses results obtained from DRAMA to target RAM rows in close physical proximity to each other, the software released as part of this research is hard-coded with memory mappings for Intel Coffee Lake CPUs.[11][12] Therefore it is not readily portable to systems with other CPU varieties. The solution to this, created by separate researchers, is for the memory mapping to be provided to Blacksmith at runtime via a configuration file formatted in JSON. Each user of Blacksmith can then create their own configuration file using results from DRAMA. This version of the program currently exists on a separate GitHub branch, which will henceforth be referred to as the "JSON config branch".[13] In order for the ECC validator created in this project to be functional on as many systems as possible, it should be built off this particular branch of Blacksmith.

Requirements

3.1 Problem Statement

Given the research outlined in the prior chapters, the conclusion reached on how to achieve the goal of this project can be described as follows:

A new fork of the JSON config branch of Blacksmith will be created, and named “Eccsmith”. In this fork, monitoring of the Rasdaemon database will be implemented, such that a user is able to cause bit flip errors and see results of whether or not the ECC mechanism on their system was able to correct these errors. Furthermore, this whole process should be made as simple as possible, so that it is an easier way of validating ECC than the currently existing methods of doing so.

3.2 System Requirements

There is a set of requirements which the end user’s system must fulfill in order for them to be able to use Eccsmith. Firstly, there are the pre-existing system requirements of Blacksmith. The only packages which it requires are g++ and cmake, because the program is built with a fully comprehensive cmake setup which automatically installs its various required dependencies at build time. The developers also recommend Ubuntu version 18.04 LTS with kernel version 4.15 to be used, so Ubuntu will be the target platform of the software.[11]

Beyond these requirements, the main system modification which Blacksmith requires users to make is that 1 gigabyte hugepages must be enabled. Given that Blacksmith performs the many-sided hammering technique at various random memory addresses, it requires the availability of a large, contiguous region of memory for it to operate within. But the default memory page size available in Linux is only 4 kilobytes, which is too small for this. So before using Blacksmith, users are required to enable at least a single 1 gigabyte hugepage to be available on their system. Blacksmith’s developers provide instructions on how to do this.[14]

Then there are the additional requirements demanded by the changes made in Eccsmith. The main addition is that Rasdaemon must be installed by the user. While Rasdaemon is the de facto hardware error event logger for Ubuntu and Debian, it does not come pre-installed in either distribution, so end users will be required to manually install its apt package. Additionally, in order for Eccsmith to be able to communicate with Rasdaemon’s database, it needs to be built with the SQLite3 C++ library, which needs to be both included in the cmake build process and installed by the user.

3.3 Software Requirements

The experience of using Eccsmith from a user's perspective should be roughly as follows:

After building the program from source and configuring their system according to the System Requirements section above - presumably by following instructions from the repository's `README.md` file - the user will simply be able to run the main Eccsmith executable, without providing any command line arguments or performing any further work. The program will then undergo all of the necessary remaining configuration itself, to the point that it finds the correct reverse-engineered memory mapping for the host system. It will then enter the fuzzing stage of its operation, which may last a long time. The user can leave the program running in this time while they do something else. During this stage, Eccsmith will perform several different many-sided hammering patterns, and after each one it will both scan for caused bit flips and for logged ECC corrections. If either of these occur, details of them will be logged and displayed clearly with accompanying timestamps, so that the user can check the program's output occasionally and observe its progress. Once Eccsmith has gathered enough information to form a verdict on whether or not ECC is functioning properly on the host system, it will cease operation and display its findings.

Legal and Professional Issues

4.1 Using an Exploit for Good

The central conceit of Eccsmith is that in an attempt to ensure the stability of computer systems, the software must first leverage a fault injection exploit which has historically been used to undermine the security of computer systems.[15] While it is not illegal to simply use a security exploit on your own computer if you do not use it to access or modify data that does not belong to you,[16] this project does involve making advancements in the ease of use of a security exploit, which currently has no definitive countermeasure - not even ECC.[17]

In this project's defense, given the potential for Rowhammer's ability to do harm, it is a near certainty that there have already been improvements made in its effectiveness and usability by those with malicious intent, but that they simply have not made this knowledge public so that their methods can remain unimpeded. Therefore if the Eccsmith project is pursued, only those who seek to use it for its intended purpose will stand to gain anything from it.

4.2 Adapting Others' Work

Given that the software which will result from this project will be a modification of an existing piece of software, it is important to clarify exactly what code is a contribution of this project, and what code existed already. To this end, the Results chapter of this report makes this distinction clear in all cases. Furthermore, all changes can be viewed in exact detail in the commit history of Eccsmith's GitHub repository.

Beyond this point, there is the issue of having the permission to produce a modified version of the code in the first place. To clarify, the Blacksmith software was released under the MIT License, the terms of which explicitly grant permission to "use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software".[18] Additionally, the developers of Blacksmith and its JSON config branch were contacted and informed of this project, and they indicated that they were happy with their code being adapted.

Methodology

5.1 ECC Validation

Blacksmith has three different modes which it can be run in, that can be selected via command-line arguments:

1. **Fuzzy Hammerer:** Blacksmith's non-uniform, many-sided Rowhammer fuzzer.
2. **Replaying Hammerer:** Takes an input file containing information about a previously executed Fuzzy Hammerer run, and replays the most effective hammering patterns found, in order to test their reproducibility.
3. **Traditional Hammerer:** A classic implementation of Rowhammer.

Since circumventing TRR in order to function correctly on modern systems is the reason for using Blacksmith in this project, it would be unnecessary to include the Traditional Hammerer in Eccsmith, so it could simply be removed when adapting the software.

As for the Replaying Hammerer, this mode has the potential to cause bit flips much more frequently than Fuzzy Hammerer, since it only uses hammering patterns that have already been found to be effective. While using it requires doing a Fuzzy Hammerer run first, it would be possible for Fuzzy Hammerer to be run up until an effective hammering pattern is found, at which point program execution could shift over to Replaying Hammerer instead, which would replay the effective pattern. This technique would eventually result in a high frequency of bit flips for ECC to correct, which would be very effective at stress testing it.

For the sake of simplicity however, Eccsmith will only use Fuzzy Hammerer, and will stop running after some number of bit flips have either been corrected by ECC or have gone uncorrected. Eccsmith does not necessarily need to be a stress tester - it simply needs to verify that ECC is functioning. So Replaying Hammerer could also be removed when adapting Blacksmith.

At the centre of Fuzzy Hammerer's main loop are two key instructions: do hammering, and check for bit flips. Eccsmith will add a third, which is to check for bit flips corrected by ECC. It will do this by querying Rasdaemon's database to get an updated count of the number of ECC corrections on record. If this number has increased since the last check, it means that the hammering performed in this iteration of the loop has caused bit flips which ECC corrected. The number of new corrections will be added to a running count of all corrections from this run, alongside the running count of uncorrected bit flips which Blacksmith already maintains.

Blacksmith outputs lots of technical details to the console about the fuzzing run during execution, but it also offers the option to log this to a file instead. For the sake of simplicity, Eccsmith will use this functionality to log these details to a separate file by default, but will still print alerts to the terminal when corrected or uncorrected bit flips are detected, so that the user can easily see the run's progress. Then when either the corrected or uncorrected count reaches a certain threshold, Eccsmith will halt and display a final verdict on its confidence as to whether or not ECC is functioning properly. If it detected only corrected bit flips and no uncorrected bit flips, it will convey confidence that ECC is working, but any results worse than this will result in a negative verdict.

5.2 Simplified Configuration

The main executable in the JSON config branch of Blacksmith needs to be provided with an argument defining the path to a valid config file in order for it to run. These config files contain various values that mostly describe attributes of the host system, including its memory mapping. The repository contains some pre-made config files for some systems. The user is expected to create their own config file in order to use Blacksmith. They can do this by using DRAMA to reverse-engineer the memory mapping of their system, or by copying one from a pre-made config file if it matches their system.[13]

Two other important values in these config files are:

- **threshold**: The number of clock cycles which differentiates between a memory row buffer hit and a memory row buffer miss, i.e. A memory access featuring a row buffer hit will take a smaller number of cycles than **threshold**, and one featuring a row buffer miss will take a greater number of cycles than **threshold**.
- **acts_per_trefi**: The number of times a row of memory can be accessed within a short time period before TRR refreshes the surrounding rows. This value being correct for the host system is instrumental to Blacksmith's ability to circumvent TRR.

The repository also contains a number of helper programs intended to assist users in the creation of config files. The following is a short description of each of these helper programs:

- **determineConflictThreshold** and **visualize_access_timings.py** are intended to be used together to determine a value for **threshold** in the config file. The first program records the necessary timing measurements, and then the Python script plots them on a graph, which the user is expected to visually inspect in order to determine the **threshold** value.
- **determineActsPerRef** and **visualize_acts_per_ref.py** are intended to be used together to determine a value for **acts_per_trefi** in the config file. Again, the first program records the necessary timing measurements, and then the Python script plots them on a graph. But this script also actually calculates and outputs the resulting value for **acts_per_trefi** itself.

- `checkAddrFunction` can be used on a config file to test various row access timings and see if they conflict with the config's `threshold` value, to see if its memory mapping is correct.

In the development of Eccsmith, the functionalities of these helper programs could all be incorporated into the main Eccsmith executable, making `threshold` and `acts_per_trefi` be determined and the memory mapping be checked automatically at runtime, rather than expecting the user to figure out how to use these various different auxiliary programs. Furthermore, the main branch of Blacksmith without the features of the JSON config branch already contains code for calculating an equivalent to `acts_per_trefi` internally, which is only used when a config file contains an `acts_per_trefi` value of 0, so this code could simply be used instead of the `determineActsPerRef` code.

These changes would mean that `threshold` and `acts_per_trefi` could be removed from the format of config files. The impact of this is that Eccsmith would take slightly longer to start running, because it would have to calculate these values again every time, but it would make the software significantly easier to operate.

Of the remaining values which are required to be in config files, some more can be removed to simplify things further:

- `max_rows` and `hammer_rounds`: These values are only used in the Traditional Hammerer mode, so they would no longer be necessary.
- `drama_rounds`: This value is used as a parameter in some row access timing measurements. It simply determines the level of precision to use, so it could be replaced with some default value.

Aside from the `name` value which contains a user-defined name for the config file, the only values remaining in the config files given the removal of those above would be the following:

- `channels`, `dimms`, `ranks`, and `total_banks`: These are aspects of the host system's memory hardware. The user is expected to inspect their system's memory hardware in order to fill in these values.
- `row_bits`, `col_bits`, and `bank_bits`: These values comprise a memory mapping function. The user is expected to use DRAMA to reverse-engineer their system's memory mapping function in order to fill in these values.

The factors which determine the memory mapping function of a system are its CPU microarchitecture and its memory hardware (which will together henceforth be referred to as "memory setup"). Therefore, if the above values are the only values remaining in the config file format which Eccsmith will use, then its repository could simply be populated with config files for every common memory setup, meaning users would never have to create or edit config files themselves. Furthermore, Eccsmith could automatically choose the correct config file to use at runtime, by checking attributes of the host system in order to determine its memory setup. As a result, this would entirely eliminate the lengthy configuration process which Blacksmith normally requires, thereby making Eccsmith a much more usable and appealing piece of software.

Results and Evaluation

6.1 Proof of Concept

Before beginning to modify Blacksmith, the software was tested first in order to verify that the premise of this project would work. This testing began by initially using Blacksmith on a laptop which does not implement ECC, with the aim of simply causing bit flips, so that it could be used as a control against a system which does implement ECC. This laptop will henceforth be referred to as System A, and its full details can be found in appendix A.1.

System A has an Intel Core i5-8265U CPU, which is an Intel Whiskey Lake processor, and therefore uses the same microarchitecture as Intel’s Coffee Lake processors.[19] All three of the pre-existing config files included in the JSON config branch’s repository were also created for systems with this microarchitecture, but none of them matched System A’s memory hardware, so a new config file had to be created. This required the use of DRAMA to determine the memory mapping function of System A. This was initially very challenging, because DRAMA does not output complete memory mapping functions ready to be used in Blacksmith’s config files. Instead, it reports individual memory address bit positions which it deems likely to be involved in the mapping function, without any indication of how these are incorporated into the function. At this stage, one of the developers of the JSON config branch of Blacksmith was contacted, and he gave assistance on how to use DRAMA effectively for this purpose. Specifically, he offered the observation that the values of `row_bits` and `col_bits` in every memory mapping function seem to always be very similar. This assistance allowed for the construction of a memory mapping function which `checkAddrFunction` verified to be correct on System A. Of note is the fact that the resulting memory mapping function was identical to that of one of the pre-existing config files, despite the fact that they each specified different memory hardware. This memory mapping function can be found in appendix A.2.

However, Blacksmith was ultimately unable to produce any bit flips on System A, even after being run for several hours and using various different values for `acts_per_trefi` produced by the helper programs. It is unknown why this happened, but it is hypothesised that System A is simply not susceptible to Rowhammer, possibly because it uses a SODIMM as its memory rather than a regular DIMM. The decision was eventually made to move on to experiments with a system that does implement ECC.

To this end, a server implementing ECC was assembled. This server will henceforth be referred to as System B, and its full details can be found in appendix A.1. System B has an Intel Xeon E3-1220 v6 CPU, which is an Intel Kaby Lake

processor, and therefore also uses the same microarchitecture as Intel’s Coffee Lake processors.[20] By using `checkAddrFunction`, the same memory mapping function which worked on System A was found to be correct for System B as well, even though their memory hardware differs. This mapping function was implemented into a new config file for System B as a result.

The next stage would have been to test Blacksmith on System B with ECC temporarily disabled, but it was discovered that System B’s BIOS didn’t include the option to disable ECC. This stage was skipped as a result, and Blacksmith was instead run with ECC enabled, while Rasdaemon was running and monitoring for ECC corrections. To make this process easier, a small script was created which read from the logs of both programs and wrote the relevant details to the same terminal window, including details of any ECC corrections. After leaving this setup for several hours, Rasdaemon was found to have reported many corrected bit flips during the time frame. This verified the fact that it would be possible to validate ECC functionality using Rowhammer, meaning the modification of Blacksmith could begin.

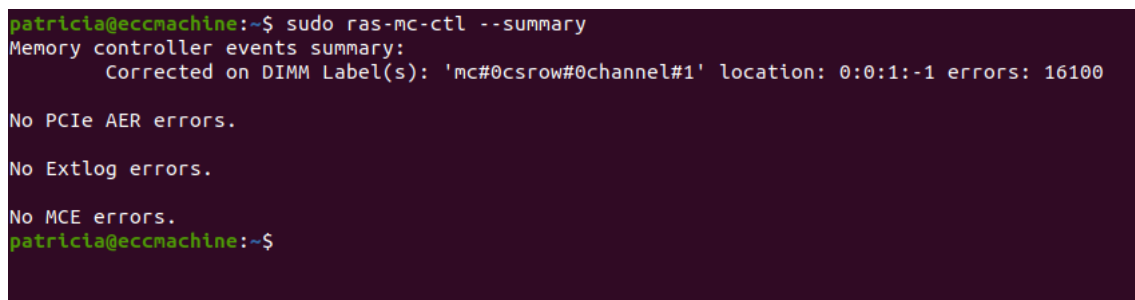
A terminal window with a dark background and light-colored text. The prompt is 'patricia@eccmachine:~\$'. The command 'sudo ras-mc-ctl --summary' has been executed. The output shows 'Memory controller events summary:' followed by 'Corrected on DIMM Label(s): 'mc#0csrow#0channel#1' location: 0:0:1:-1 errors: 16100'. Below this, it lists 'No PCIe AER errors.', 'No Extlog errors.', and 'No MCE errors.'. The prompt returns to 'patricia@eccmachine:~\$'.

Figure 6.1: Rasdaemon showing that 16,100 ECC corrections had been caused on System B by the end of the project

6.2 Implementing ECC Validation

After Blacksmith was forked to create the Eccsmith repository, ECC validation was implemented using the newly created class `RasWatcher`. This class handles all communication with Rasdaemon’s database, and encapsulates the number of corrections counted during the run. In addition, the Fuzzy Hammerer mode of operation was modified such that in the centre of its main loop, it calls a `RasWatcher` method which retrieves an updated count of how many ECC corrections have occurred. This number of new corrections then gets implemented into Fuzzy Hammerer’s evaluation of which hammering patterns are effective, alongside the existing count of uncorrected bit flips which is already used in this way. Then the other modes of operation - Replaying Hammerer and Traditional Hammerer - were disabled and removed, along with their corresponding program arguments.

The logging system was also reworked, such that only details about the start of the run, corrected bit flips, uncorrected bit flips, and the end of the run are printed, and all other technical details are recorded in a log file. After the chosen run time limit elapses, or after three effective hammering patterns have been found, Eccsmith ends the fuzzing run and displays a verdict on whether or not ECC is functioning correctly.

Eccsmith was found to regularly take roughly an hour to produce any corrections on System B, prompting its default runtime to be increased to 3 hours, from Blacksmith’s default of 2 minutes. It was also found to almost exclusively report exactly 39 corrections at a time. Upon manual inspection of Rasdaemon’s database, this figure was found to be consistent with its records. However, after further investigation, Rasdaemon was found to not be recording corrections accurately. By manually inspecting System B’s event tracking log using `dmesg` while running Eccsmith, it was found that when ECC corrections were reported by `dmesg`, they were not immediately reported by Rasdaemon, and by extension Eccsmith. Rasdaemon only appeared to be reporting these errors once 39 had been logged in `dmesg`, at which point they would all be logged in Rasdaemon at once.

This bug in Rasdaemon was found to have previously been reported by various other people.[21] The bug was found to actually have been caused by a change to the Linux kernel, which has already been fixed. The breaking change was made in kernel version 6.1, and it was fixed in version 6.2.[22] For the bug to be completely fixed however, Rasdaemon also needed to be amended, which was done in version 0.8.0 of Rasdaemon.[23] System B was using Ubuntu version 22.04 LTS, which uses kernel version 6.5, so the bug was fixed within its kernel. However, the highest version of Rasdaemon available on this Ubuntu version is 0.6.7, meaning it was not fixed within Rasdaemon itself, so it was not possible to prevent the bug.

Unfortunately, there was not enough time in the project to solve this issue. The next step needed to investigate it further would have been to install an earlier version of Ubuntu on System B which uses a kernel version from before the bug was introduced. The only impact of this issue is that Eccsmith regularly took roughly an hour to visibly produce any corrections, because it needed to produce up to 39 first before any could be reported. This may not be the case on other platforms.

```

patricia@eccmachine:~/Documents/eccsmith/build$ sudo ./eccsmith -c ../config/test-config.json
[+] General information about this fuzzing run:
Start time: 15:29
Run time limit: 3 hours
Config name: test-config
Hostname: eccmachine
Commit SHA: 26a552b20a6fb452da12c1762c69be1040134349
[+] Initializing memory with pseudorandom sequence.
[+] Opening connection to Rasdaemon database, with a total of 14859 prior ECC corrections on record.
[+] Fuzzing has started. Details are being written to run.log. Any detected bitflips will also be written to the console.
[!] ECC successfully corrected 39 bitflip(s) at 0 hours 41 minutes 39 seconds.
[!] ECC successfully corrected 39 bitflip(s) at 0 hours 51 minutes 3 seconds.
[!] ECC successfully corrected 39 bitflip(s) at 1 hours 2 minutes 9 seconds.
[!] ECC successfully corrected 39 bitflip(s) at 1 hours 29 minutes 23 seconds.
[!] ECC successfully corrected 39 bitflip(s) at 1 hours 29 minutes 38 seconds.
[!] ECC successfully corrected 39 bitflip(s) at 1 hours 43 minutes 24 seconds.
[!] ECC successfully corrected 34 bitflip(s) at 2 hours 14 minutes 35 seconds.

```

Figure 6.2: A run of Eccsmith on System B after implementing ECC validation, featuring the 39 corrections bug

6.3 Implementing Simplified Configuration

The format of valid config files was altered by removing all of the requirements for the `threshold`, `acts_per_trefi`, `max_rows`, `hammer_rounds`, and `drama_rounds` fields from the existing code. The helper programs were also removed from the repository. In exchange, in order to make Eccsmith determine `acts_per_trefi` on its own, the code was modified to make it default to the existing built-in solution for finding this value, in the `DramAnalyser` class.

However, there were some issues with this code. Firstly, it assumed a `threshold` value of 1000 in order for it to tell when a row miss occurs and TRR refreshes the victim rows. This was fixed by making the code first determine an accurate value of `threshold` instead. This was implemented by importing the pre-existing code from `determineConflictThreshold` to take the necessary measurements, and then creating a new part which finds the averages of all the row hit and row miss timings, and returns the midpoint between these two values. This circumvents the step where `visualize_access_timings.py` would formerly require the user to visually inspect a plot of the data to determine this value.

Secondly, the code had a bug which made it never finish. This is presumably why the `determineActsPerRef` and `visualize_acts_per_ref.py` helper programs were created in the first place. One option at this point would have been to import the code from these helper programs into the `DramAnalyser` class too, and to use this as a replacement. However, upon inspection of both methods, the code within the `DramAnalyser` class was determined to be more sophisticated, and likely to produce more accurate results if it could be fixed. This is because it iteratively measures the number of row activations needed to trigger TRR, until the standard deviation of all measurements drops below 3, at which point it returns the mean of all measurements. Whereas the combined solution of `determineActsPerRef` and `visualize_acts_per_ref.py` simply takes 1000 measurements and finds their mean, with no accounting for outliers. Therefore the decision was made to fix the existing code in the `DramAnalyser` class.

The existing code was getting stuck because outliers in its measurements caused the standard deviation to never drop below 3 regardless of how many measurements were taken. This was fixed by introducing a condition which abandoned all measurements and tried again if the standard deviation did not drop below 3 after 2000 measurements. In addition, to reduce the likelihood of outliers in the next attempt, the `threshold` value would be raised by 10 on each retry. Then if enough retries were done that `threshold` was raised to be 200 greater than its original value, it would be reset back to its original value in the next retry. Finally, measurement would also be restarted if `acts_per_trefi` was found to be less than or equal to 5, to prevent the case where so many outliers occur that the standard deviation does drop below 3 and `acts_per_trefi` is erroneously found to be a very small value.

With all of these changes implemented, the `DramAnalyser` class no longer gets stuck when run on System B, and it consistently produces effective values for `acts_per_trefi`. As a result, Eccsmith is able to automatically determine all parameters at runtime. In addition, the pre-existing code from `checkAddrFunction` was also integrated into the `DramAnalyser` class, such that the selected config file's mapping function is checked at the start of a run, using the `threshold` value which is already determined earlier.

However, there was not enough time available in the project to implement the automatic selection of the correct config file. As a result, Eccsmith requires the user to manually select the correct pre-made config file from the repository, if one which is correct for their system already exists. Furthermore, there was not enough time to create any more pre-made config files for Eccsmith users to choose from, leaving only a selection covering four very similar memory setups. These four config files only actually use two different memory mapping functions between them, both of which can be found in appendix A.2. Both functions had already been discovered as part of Blacksmith’s JSON config branch prior to this project.

CPU Microarchitecture	Channels	DIMMs	Ranks	Banks
Intel Coffee Lake	1	1	1	8
	1	1	2	8
	1	1	1	16
	1	1	2	32

Table 6.1: All memory setups for which there are pre-made config files in Eccsmith’s repository

```

patricia@eccmachine:~/Documents/eccsmith/build$ sudo ./eccsmith -c ../config/coffee-lake-1-1-1-8.json
[sudo] password for patricia:
[+] General information about this fuzzing run:
Start time: 18:24
Run time limit: 3 hours
Config name: coffee-lake-1-1-1-8
Hostname: eccmachine
Commit SHA: 22300f8bb4a61d38322364e3c642c1b245ac45e1
[+] Memory initialized with pseudorandom sequence.
[+] Determined row conflict threshold to be 316.
[+] Selected config file has been checked, and seems to be correct.
[+] Determined number of row activations per refresh interval to be 86.
[+] Connected to Rasdaemon database, with a total of 16100 prior ECC corrections on record.
[%] Fuzzing has started. Details are being written to run.log. Any detected bitflips will also be written to the console.

```

Figure 6.3: The beginning of a run of Eccsmith after implementing the changes to the program configuration

6.4 Evaluation

Eccsmith partially meets the criteria laid out by the Requirements chapter of this report. It is a successful implementation of the novel and practical use case for Rowhammer initially stated in the Introduction chapter - as the means of triggering ECC in order to prove that it is functioning correctly.

Although, it is arguably not an easy method of validating ECC. In particular, Eccsmith is not capable of fully configuring itself to the host system automatically. The user is still required to manually select a config file from its repository, if a correct one does already exist. Even selecting the correct pre-existing config file is challenging, because a user is required to find out their system’s CPU model, then figure out which microarchitecture it uses, and then to understand their system’s memory hardware, in order for them to know which one to choose. In addition, the repository was not populated with a wide variety of config files to account for all common memory configurations, meaning the user may be required to create one

themselves using DRAMA if no suitable config file already exists. Finally, the error involving ECC corrections only being reported in groups of 39 would slow down Eccsmith's ability to reach a verdict on ECC's functionality, if it were to also occur on a prospective user's system. These factors reduce the likelihood that anyone seeking to validate the functionality of ECC on their system would realistically use Eccsmith for this purpose, because it would simply be too difficult.

However, if the automated selection of config files could be implemented, it is possible that Eccsmith may slowly be widely adopted, as users start to populate its repository with config files which they make themselves. Therefore Eccsmith has come reasonably close to meeting the criteria laid out by the Requirements chapter, given the time allowed for this project.

Conclusion

7.1 Summary

This report has demonstrated a novel and practical use case for Rowhammer, as the means of triggering ECC in order to prove that it is functioning correctly. This use case has then been applied in practice, by modifying the Blacksmith Rowhammer fuzzer to convert it into an ECC validator. The ultimate goal of this project was for this ECC validator to be easy to use. The project did not fully achieve this goal, but it came reasonably close to doing so, and it still may be achieved with future work.

7.2 Suggestions for Future Work

The main priority of any future work on Eccsmith should be to confirm the cause of the error involving ECC corrections only being reported in groups of 39, by testing it on more systems using differing versions of Ubuntu, the Linux kernel, and Rasdaemon. Providing a means of either fixing or avoiding this would then greatly improve Eccsmith's efficiency.

Beyond this, the next priority should be to implement the automated selection of config files, which is likely to be relatively easy. Once this has been achieved, Eccsmith needs to be tested on more systems to verify other areas of its functionality. In particular, it is important to test the improved `threshold` and `acts_per_trefi` measurement process on other systems.

Additionally, there is a need for Eccsmith's repository to be populated with more config files in order for it to be compatible with other systems. This process could be made easier by improvements to DRAMA, with the aim of making it able to automatically generate full memory mapping functions, rather than just suggesting which bits may be involved in the function, which requires trial-and-error on the user's behalf. It would also be beneficial to begin utilising the relatively recent research into the reverse-engineering of AMD's processors, given that DRAMA is only capable of reverse-engineering Intel processors.[24]

References

- [1] James F. Ziegler et al. “IBM Experiments in Soft Fails in Computer Electronics”. In: *IBM Journal of Research and Development*. Vol. 40. 1996, pp. 3–18. URL: <https://ieeexplore.ieee.org/document/5389432>.
- [2] PassMark Software. *Memtest86 - ECC Technical Details*. 2021. URL: <https://www.memtest86.com/ecc.htm>.
- [3] PassMark Software. *ECC Tester*. 2023. URL: <https://www.passmark.com/products/ecc-tester/index.php>.
- [4] *Inducing ECC Errors the Hardware Way*. 2021. URL: <https://forum.level1techs.com/t/inducing-ecc-errors-hardware-way>.
- [5] Yoongu Kim et al. “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors”. In: *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 2014, pp. 361–372. URL: <https://ieeexplore.ieee.org/document/6853210>.
- [6] Patrick Jattke et al. “Blacksmith: Scalable Rowhammering in the Frequency Domain”. In: *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. 2022, pp. 716–734. URL: <https://ieeexplore.ieee.org/document/9833772>.
- [7] Mauro Chehab et al. *Rasdaemon*. 2013. URL: <https://github.com/mchehab/rasdaemon>.
- [8] Yichen Jiang et al. “TRRScope: Understanding Target Row Refresh Mechanism for Modern DDR Protection”. In: *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2021, pp. 239–247. URL: <https://ieeexplore.ieee.org/document/9702274>.
- [9] Pietro Frigo et al. “TRRespass: Exploiting the Many Sides of Target Row Refresh”. In: *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. 2020, pp. 747–762. URL: <https://ieeexplore.ieee.org/document/9152631>.
- [10] Peter Pessl et al. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *Proceedings of the 25th USENIX Security Symposium*. 2016, pp. 565–581. URL: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl>.
- [11] Patrick Jattke et al. *Blacksmith Rowhammer Fuzzer*. 2021. URL: <https://github.com/comsec-group/blacksmith>.

- [12] Patrick Jattke. *Blacksmith on Non-Coffee Lake CPUs*. 2021. URL: <https://github.com/comsec-group/blacksmith/issues/4#issuecomment-972629985>.
- [13] Jeremy Boy and Luca Wilke. *Blacksmith JSON Config Branch*. 2023. URL: <https://github.com/UzL-ITS/blacksmith/tree/jsonconfig>.
- [14] Patrick Jattke. *Instructions on enabling hugepages*. 2021. URL: <https://github.com/comsec-group/blacksmith/issues/2#issuecomment-971810211>.
- [15] Matthew Dempsky and Thomas Dullien. *Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges*. 2015. URL: <https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html>.
- [16] *Computer Misuse Act*. 1990. URL: <https://www.legislation.gov.uk/ukpga/1990/18/contents>.
- [17] Lucian Cojocar et al. “Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks”. In: *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 55–71. URL: <https://ieeexplore.ieee.org/document/8835222>.
- [18] The Open Source Initiative. *The MIT License*. URL: <https://opensource.org/license/mit>.
- [19] Intel. *Intel Core i5-8265U Processor*. 2018. URL: <https://ark.intel.com/content/www/us/en/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html>.
- [20] Intel. *Intel Xeon E3-1220 v6 Processor*. 2017. URL: <https://www.intel.com/content/www/us/en/products/sku/97470/intel-xeon-processor-e31220-v6-8m-cache-3-00-ghz/specifications.html>.
- [21] Harshit Mogalapalli. *Rasdaemon does not report new records*. 2023. URL: <https://lore.kernel.org/all/31eb3b12-3350-90a4-a0d9-d1494db7cf74@oracle.com/>.
- [22] Shiju Jose and Steven Rostedt. *Fix poll() and select() do not work on per_cpu trace_pipe and trace_pipe_raw*. 2023. URL: <https://github.com/torvalds/linux/commit/3e46d91>.
- [23] Shiju Jose. *Fix poll() on per_cpu trace_pipe_raw blocks indefinitely*. 2023. URL: <https://github.com/mchehab/rasdaemon/commit/6986d81>.
- [24] Martin Heckel and Florian Adamsky. “Reverse-Engineering Bank Addressing Functions on AMD CPUs”. In: 2023. URL: <https://dramsec.ethz.ch/papers/revengamd.pdf>.

Appendices

A.1 System Details

System A

Model: Lenovo V15-IWL

CPU: Intel Core i5-8265U (1.6 GHz, Quad-core, x86-64)

RAM: SK Hynix 8 GB DDR4 SODIMM (2.4 GHz, Dual-rank, 8 Banks)

OS: Ubuntu 22.04.3 LTS, Kernel ver. 6.5

System B

Model: Supermicro X11SSL-CF

CPU: Intel Xeon E3-1220 v6 (3 GHz, Quad-core, x86-64)

RAM: Samsung 8 GB DDR4 DIMM with ECC (2.4 GHz, Single-rank, 8 Banks)

OS: Ubuntu 22.04.3 LTS, Kernel ver. 6.5

A.2 Memory Mapping Functions

Mapping Compatible With System A and System B

row_bits: [29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17]

col_bits: [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

bank_bits: [[6, 13], [14, 17], [15, 18], [16, 19]]

Other Mapping

row_bits: [29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18]

col_bits: [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

bank_bits: [[6, 13], [14, 18], [15, 19], [16, 20], [17, 21]]