REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
08-08-2023	Final Report	22-May-2017 - 17-Feb-2018

4. TITLE AND SUBTITLE
Final Report: Towards Provably Secure Malware Defenses

5a. CONTRACT NUMBER
W911NF-17-1-0253

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611102

6. AUTHORS

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES

University of California - Los Angeles
Office of Contract and Grant Administration
11000 Kinross Avenue, Suite 211
Los Angeles, CA 90095 -1406

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
69738-NC-II.3

12. DISTRIBUTION AVAILIBILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Rafail Ostrovsky
UU	UU	UU	UU		19b. TELEPHONE NUMBER
					310-206-5283

RPPR Final Report
as of 14-Aug-2023

Agency Code: 21XD

Proposal Number: 69738NCII **Agreement Number: W911NF-17-1-0253**
INVESTIGATOR(S):

Name: Rafail Ostrovsky
Email: rafail@cs.ucla.edu
Phone Number: 3102065283
Principal: Y

Organization: **University of California - Los Angeles**
Address: Office of Contract and Grant Administration, Los Angeles, CA 900951406
Country: USA
DUNS Number: 092530369 EIN: 956006143
Report Date: 17-May-2018 Date Received: 08-Aug-2023
Final Report for Period Beginning 22-May-2017 and Ending 17-Feb-2018
Title: Towards Provably Secure Malware Defenses
Begin Performance Period: 22-May-2017 **End Performance Period:** 17-Feb-2018
Report Term: 0-Other
Submitted By: Rafail Ostrovsky Email: rafail@cs.ucla.edu
Phone: (310) 206-5283

STEM Degrees: **STEM Participants:**

Major Goals: In modern architectures, there are several attacks which directly corrupt contents in memory. For example, the Direct Memory Access (DMA) enables components of the architecture (such as embedded devices and/or peripherals) to access a computer's memory directly, without OS supervision. While this feature permits extremely fast data transfers, it also allows a malicious (e.g., exploited) device to potentially steal sensitive data, or run malicious programs on the computer. Another typical memory-corruption attack is Rowhammer which if successful, causes some memory cells to interact with and flip the values in nearby memory rows.

Common mitigations used against these kinds of attacks include disallowing physical connections or access to the machine, disabling DMA, or enhancing the system such that damage is minimized from memory corruption (e.g. using error-correcting code to recover from corrupted memory). However,
while most existing solutions may cover part of the attack surface, they do not stop all feasible attacks with a similar effect.
For example, denying physical access to the device may not be always possible, and bit flipping faults can be induced without physical access to the device (e.g. Rowhammer over the network radiation). Moreover, error-correcting codes, an existing defense against Rowhammer, are ineffective against an attacker who can alter arbitrarily many bits. Notably, following the common trend in injection-detection, existing defenses against such attacks are typically heuristics, and do not come with a proof of security or a characterization of the class of attacks that they defend. In contrast, cryptographic solutions which do come with a security model and proof, induce too high slowdowns, rendering them inapplicable.
The above situation leads to the following natural question:
Can we formally define a class of realistic attacks which is general enough to capture the above types of memory corruption, and can we devise/implement an efficient scheme that is provably secure against any attack in this class?
In this work we answer the above question affirmatively. In a nutshell, we first define a model of computation which is as close as possible to the underlying system on top of which the scheme runs and devise appropriate cryptographic-style definitions of what it means to be ``secure'' against the attack class. We then describe a detection scheme for such attacks which is provably secure, and benchmark its performance various simulated settings. Even though the simulation of hardware components induces a clear slowdown in the execution, our results indicate that with only minimal extensions on the CPU list of commands (and no additional security assumptions from the CPU) we can get slowdown which can be acceptable for securing critical computations.

Accomplishments: We focus on detecting a subset of exploits which allows the attacker to target and overwrite any part of memory. We call this attack ``direct injection''. In particular, this includes (but is not limited to) a system

being the target of a successful write-only Direct Memory Access (DMA) attack. Our attack allows arbitrary leakage on the program which is executed and its inputs and outputs---the attacker knows (in fact, he chooses) which program is being executed and on what inputs. In Section “future works” we will discuss as future direction, how we can protect against an adversary who can read memory as well as write, and how we can extend our security to include other memory corruptions, such as bit-flips of Rowhammer attacks.
Our work extends the theoretical model and scheme of PI to include and handle features of modern architecture such as cache, the heap, variable instruction lengths, and a richer instruction set including function calls and heap operations. We devise cryptographic-style definitions of security in this more realistic model. We further improve the efficiency of this work in two ways. First, we replace the expensive random oracle invocation (implemented by a hash function) and multiplication steps with one AES operation, while formalizing the necessary assumptions on AES. (Roughly, we need it to have properties similar to a strong randomness extractors.) Second, we leverage the cache, which had not been considered before, to avoid the expensive repetition of the scheme's verification for data that was retrieved from cache (rather than from the corruptable memory). Using the cache, we can also remove the atomicity assumption that the adversary cannot attack between the MAC verification and data access.

We prove the security of our scheme by showing that it satisfies two properties which we formulate in our model: correctness, and detection accuracy. Given a system vulnerable to direct injection attacks, and a program whose code and data we wish to secure, we transform the program such that it behaves the same way as before when no attack has happened (correctness property), but is able to respond to attestation queries from a remote verifier and detect when a direct injection has occurred (detection accuracy property).

Lastly, we implement a proof of concept and benchmark the performance our scheme, and demonstrate that our scheme is composable with security mechanisms implemented in the program itself. Composability is implied by the correctness property of the scheme (as by correctness, our scheme does not alter the behavior of the program when no direct injection occurs); to demonstrate this, we implement stack canaries, a common defense against buffer overflows, and pass the program with stack canaries through our scheme. This yields a program that protects against any attack which stack canaries would prevent in addition to any direct injection attack. To our knowledge, our work is the first to demonstrate generic composition of defenses that provably protects against the union of the attacks that each of the composed defenses protects. Our benchmarks indicate that for programs which make moderate use of cache---the more use the better for our benchmarks---our scheme causes at slowdown of just 2x over the unprotected program.

Training Opportunities: I have advised a UCLA Ph.D. student who benefited on discussions on this important topic. Specifically, Mr. Kevin Garbe as a Ph.D. student at UCLA who has studied the problem.

Results Dissemination: We have published the following paper based on this research:

```
author       = {Yun Lu and
                Konstantinos Mitropoulos and
                Rafail Ostrovsky and
                Avraham Weinstock and
                Vassilis Zikas},
 title       = {Cryptographically Secure Detection of Injection Attacks},
 booktitle   = {Proceedings of the 2018 {ACM} {SIGSAC} Conference on Computer and
                Communications Security, {CCS} 2018, Toronto, ON, Canada, October
                15-19, 2018},
 pages       = {2240--2242},
 publisher   = {{ACM}},
 year        = {2018},
```

Honors and Awards: • 2018 RSA Excellence in the Field of Mathematics Award (Also known as "RSA Prize") “for contributions to the theory and to new variants of secure multi-party computation”
• IEEE Computer Society 2017 Edward J. McCluskey Technical Achievement Award:“for outstanding contributions to cryptographic protocols and systems, enhancing the scope of cryptographic applications and of assured cryptographic security”
• Fellow of Institute of Electrical and Electronics Engineers (IEEE), inducted in 2017:“for contributions to cryptography”

RPPR Final Report
as of 14-Aug-2023

Protocol Activity Status:

Technology Transfer: Nothing to Report

PARTICIPANTS:

Participant Type: PD/PI
Participant: Rafail Ostrovsky
Person Months Worked: 2.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Co-Investigator
Participant: Vassilis Zikas
Person Months Worked: 2.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Kevin Garbe
Person Months Worked: 1.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Partners

,

I certify that the information in the report is complete and accurate:
Signature: Rafail Ostrovsky
Signature Date: 8/8/23 11:08PM

FINAL REPORT:
Cryptographically Secure Detection of Injection Attacks

To: Network, Cyber, and Computational Sciences Branch (NC&CS)

SUBJECT: Proposal No. 69738-NC-II - Towards Provably Secure Malware Defenses

Attn: Dr. Cliff X. Wang, Program Manager

PI's: Rafail Ostrovsky, Vassilis Zikas.

Contributors to the report: Yun Lu, Konstantinos Mitropoulos, Rafail Ostrovsky Avraham Weinstock, Vassilis Zikas

Summary

Direct Memory Access (DMA) attacks can allow attackers to access memory directly, bypassing OS supervision or software protections. In this work, we put forth and benchmark a cryptographically secure attestation scheme, which detects DMA attacks. In fact, our scheme detects any attack in a more general class of attacks which we call *direct injection*.

We prove security of our scheme under a realistic machine model which extends in a non-trivial manner a cryptographic model proposed by Lipton, Ostrovsky, and Zikas (ICALP 2016.)

Despite the fact that our scheme, in its current form, protects against write-only attacks, both our security model and our scheme can be extended to allow the attacker to have additional read access to memory---thereby capturing leakage---as well as detecting more types of memory corruptions such as bit flips.

Introduction

In modern architectures, there are several attacks which directly corrupt contents in memory. For example, the Direct Memory Access (DMA) enables components of the architecture (such as embedded devices and/or peripherals) to access a computer's memory directly, without OS supervision. While this feature permits extremely fast data transfers, it also allows a malicious (e.g., exploited) device to potentially steal sensitive data, or run malicious programs on the computer (e.g. [3,9]) Another typical memory-corruption attack is Rowhammer [7,10], which if successful, causes some memory cells to interact with and flip the values in nearby memory rows.

Common mitigations used against these kinds of attacks include disallowing physical connections or access to the machine, disabling DMA, or enhancing the system such that damage is minimized from memory corruption (e.g. using error-correcting code to recover from corrupted memory). However,

while most existing solutions may cover part of the attack surface, they do not stop all feasible attacks with a similar effect.

For example, denying physical access to the device may not be always possible, and bit flipping faults can be induced without physical access to the device (e.g. Rowhammer over the network [2,8], radiation). Moreover, error-correcting codes, an existing defense against Rowhammer, are ineffective against an attacker who can alter arbitrarily many bits [4]. Notably, following the common trend in injection-detection, existing defenses against such attacks are typically heuristics, and do not come with a proof of security or a characterization of the class of attacks that they defend. In contrast, cryptographic solutions which do come with a security model and proof, induce too high slowdowns, rendering them inapplicable.

The above situation leads to the following natural question:

Can we formally define a class of realistic attacks which is general enough to capture the above types of memory corruption, and can we devise/implement an efficient scheme that is provably secure against any attack in this class?

In this work we answer the above question affirmatively. In a nutshell, we first define a model of computation which is as close as possible to the underlying system on top of which the scheme runs and devise appropriate cryptographic-style definitions of what it means to be ``secure" against the attack class. We then describe a detection scheme for such attacks which is provably secure, and benchmark its performance various simulated settings. Even though the simulation of hardware components induces a clear slowdown in the execution, our results indicate that with only minimal extensions on the CPU list of commands (and no additional security assumptions from the CPU) we can get slowdown which can be acceptable for securing critical computations.

Our Contributions

We focus on detecting a subset of exploits which allows the attacker to target and overwrite any part of memory. We call this attack ``direct injection". In particular, this includes (but is not limited to) a system being the target of a successful write-only Direct Memory Access (DMA) attack. Our attack allows arbitrary leakage on the program which is executed and its inputs and outputs---the attacker knows (in fact, he chooses) which program is being executed and on what inputs. In Section “future works” we will discuss as future direction, how we can protect against an adversary who can read memory as well as write, and how we can extend our security to include other memory corruptions, such as bit-flips of Rowhammer attacks.

Our work extends the theoretical model and scheme of [5] to include and handle features of modern architecture such as cache, the heap, variable instruction lengths, and a richer instruction set including function calls and heap operations. We devise cryptographic-style definitions of security in this more realistic model. We further improve the efficiency of this work in two ways. First, we replace the expensive random oracle invocation (implemented by a hash function) and multiplication steps with one AES operation, while formalizing the necessary assumptions on AES. (Roughly, we need it to have properties similar to a strong randomness extractor [6].) Second, we leverage the cache, which had not been

considered before, to avoid the expensive repetition of the scheme's verification for data that was retrieved from cache (rather than from the corruptable memory). Using the cache, we can also remove the atomicity assumption that the adversary cannot attack between the MAC verification and data access implicit in [5].

We prove the security of our scheme by showing that it satisfies two properties which we formulate in our model: correctness, and detection accuracy. Given a system vulnerable to direct injection attacks, and a program whose code and data we wish to secure, we transform the program such that it behaves the same way as before when no attack has happened (*correctness property*), but is able to respond to attestation queries from a remote verifier and detect when a direct injection has occurred (*detection accuracy property*).

Lastly, we implement a proof of concept and benchmark the performance our scheme, and demonstrate that our scheme is composable with security mechanisms implemented in the program itself. Composability is implied by the correctness property of the scheme (as by correctness, our scheme does not alter the behavior of the program when no direct injection occurs); to demonstrate this, we implement *stack canaries*, a common defense against buffer overflows, and pass the program with stack canaries through our scheme. This yields a program that protects against any attack which stack canaries would prevent in addition to any direct injection attack. To our knowledge, our work is the first to demonstrate generic composition of defenses that provably protects against the union of the attacks that each of the composed defenses protects. Our benchmarks indicate that for programs which make moderate use of cache---the more use the better for our benchmarks---our scheme causes at slowdown of just 2x over the unprotected program. In Section “Future Works we will also discuss directions in reducing performance overhead.

Related Works: Provable security against memory corruption

Our approach is inspired by the recently proposed theory by Lipton et al. [5], that assumes the simple *Random Access Stored Program Machine (RASP)* model abstracting a program and its memory as a bit string. The scheme has two main components: an additive sharing of a secret key, and MAC tags (Figure 1). First, between each word of memory, the scheme inserts a share of the secret key, such that the sum of all shares inserted is the secret key. During attestation, the key shares would be validated by a challenge-response process where the program is challenged to produce the correct decryption of a random encrypted message by reconstructing the secret key. The advantage over stack canaries, however, is that every strict subset of the key shares is a set of completely random values, and the secret key is not stored in any one location.

Note that the work of Boldyreva et al. [1] also inserts additive sharing of a secret key into memory. However, their work focuses on protecting remote heaps (rather than the entire memory), with the goal of detecting heap overflow rather than attacks like DMA.

Finally, we observe that inserting key shares is not enough if the attacker is able to skip over corrupting them. Thus, a message authentication (MAC) tag authenticating each word is used, keyed by the key share after each word. This MAC tag is verified before the data is read or executed, and updated after the data is changed via a write instruction.

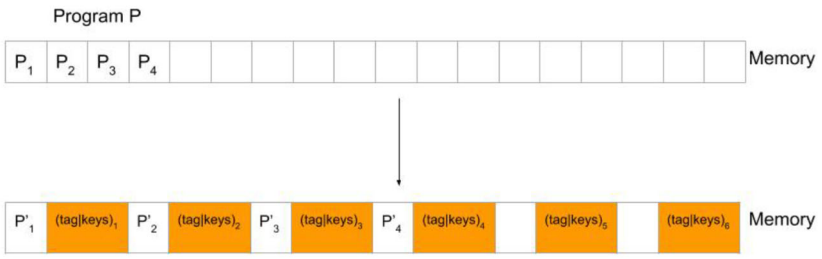

Figure 1: Mac tax corresponding to each word 1 to 4 make sure the words of data are not tampered with.

Overview of Our Scheme

Our scheme takes as input a program, written either in source code or assembly (our implementation takes either C source code or x86-64 assembly code). It first transforms the program to make space to interleave key shares and MAC tags, and changes instructions so that they do not write over or leak these key shares and MACs. It also inserts new code which performs set up, MAC tag verification, and responds to attestation challenges from the remote verifier (these are the only instructions allowed to access the key shares and MAC tags). Then, we run the modified program, where the set up code inserts the random key shares and computing MAC tags. The result of this scheme is a running process which is able to detect direct injections.

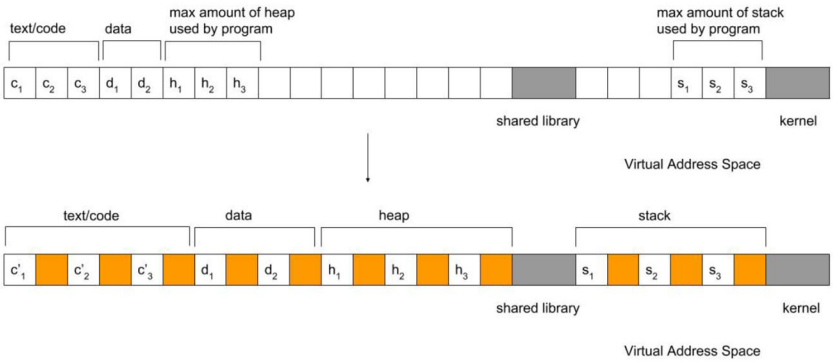

Figure 2: Key shares and MAC tags are inserted throughout each section in the virtual address space.

We show that our scheme is secure by proving the following two properties: correctness, and detection accuracy. Informally, correctness means that the modified program has the same outputs as the input program, when no direct injection attack has occurred. This is satisfied by our modification of the input program's instructions, so that they are unaware of the additional interleaved key shares and MAC tags. The second property, detection accuracy, says that when a direct injection attack has occurred, the remote verifier is able to detect it as the process is not able to answer the decryption challenge correctly. We find this property to be too strong, as it requires checking the MAC tags in the entire memory, even at locations which would never be accessed. Thus, we relax this notion of security slightly by requiring detection only in the case when the corrupted memory is either read or executed. This reduces overhead

dramatically by limiting MAC verification to only data we will access. Moreover, by taking advantage of the fact that cache is not part of memory and cannot be overwritten by direct injection attacks, we can further reduce performance overhead by only verifying the MAC tag when the data is first put into the cache.

Benchmarks

We implemented and benchmarked our scheme on a machine with four Intel Core i5-6267U at 2.9 GHz, and AES NI support, running Linux Mint 18.3 64 bit (kernel 4.13).

We tested our implementation while varying two major parameters: the number of bytes between consecutive key shares (which we call ``block size") n, and the cache size.

We tested how our implementation performed with different types of programs, such as finding primes, matrix multiplication, finding the determinant, and Dijkstra algorithm.

The benchmarks demonstrated the cache's effect on lowering the overhead of our scheme (by reducing the number of MAC verifications required). As we increase cache size towards what modern CPUs offer---i.e., 400 blocks in the figures---the performance increases almost exponentially. Moreover, when the block size n is larger, a similar overhead reduction is seen. At $n = 64$ bytes, we reach the current optimal of 2x slowdown for programs which make good use of cache or require less memory (eg. finding primes, solving Tower of Hanoi). Conversely, programs like Dijkstra algorithm and matrix multiplication suffer the most slowdown of 15x-20x. We find that composing our scheme with stack canaries incur no additional slowdown other than what was already incurred by this software protection.

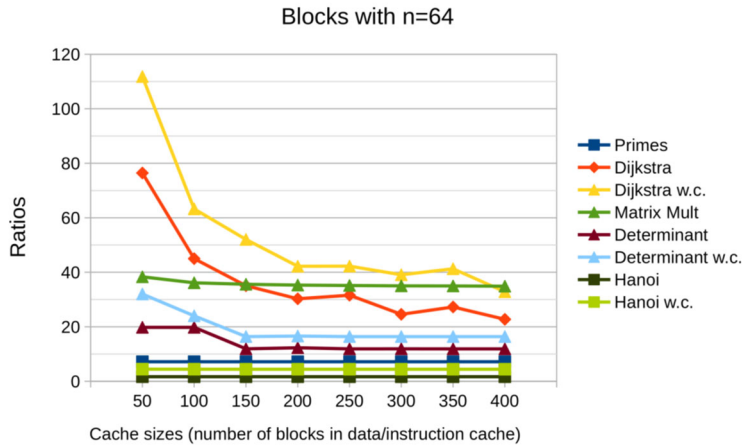

Figure 3: Slowdown of secure (with stack canaries (w.c.)) vs. insecure program run time as function of cache size and with block size 64. With larger cache (and more cache-friendly programs), blocks of data in memory may be verified less often, leading to smaller slowdown.

Future Directions

A useful extension of our current class of attacks and scheme is to include kinds of memory corruptions other than direct injection attacks, such as bit flipping. This would include attacks like Rowhammer [7,10]. Current mitigations include probabilistically refreshing rows, in addition to software defenses such as analyzing memory accesses to detect potential Rowhammer attacks (e.g. [4, 11]). However, the security of these mitigations have been shown experimentally rather than proven. Our scheme, while proven secure under a different attack, may also be secure under bit flipping attacks, as the adversary should not be able to produce a valid MAC tag by flipping bits.

Another direction is to provide security even against an adversary who could read, in addition to overwriting memory. This could be achieved partially by following the approach of [1] to periodically refresh all key shares so any strict subset of key shares leaked before a refresh become useless. However, this is an expensive operation, especially as we also need to refresh MAC tags. A possible solution would be to instead perform frequent ``local'' refreshes to small subsets of key shares.

Lastly, current efforts have been to improve the performance of the scheme. As our proof-of-concept is implemented in software, the lowest overhead has been a 2x slowdown. Thus, one direction is to develop a hardware implementation to perform MAC check and data access in one atomic operation. Another direction is use the observation that there is no need to place the MAC tags and their corresponding key shares at interleaved locations in memory. The MAC tags would authenticate each block of data regardless. Changing the layout of memory by separating program data and data used by the scheme (keys, tags) will help reduce the need to skip over the keys and tags when accessing data or executing instructions.

References

[1] Alexandra Boldyreva B, Taesoo Kim, and Richard Lipton. 2016. Provably-Secure Remote Memory Attestation for Heap Overflow Protection. SCN (2016).

[2] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. SP (2018).

[3] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2009. Lest We Remember: Cold-boot Attacks on Encryption Keys. Commun. ACM (2009).

[4] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping Bits in Memory Without Accessing Them. ISCA (2014).

[5] Richard J. Lipton, Rafail Ostrovsky, and Vassilis Zikas. 2016. Provably Secure Virus Detection: Using The Observer Effect Against Malware. In ICALP.

[6] Noam Nisan and David Zuckerman. 1996. Randomness is linear in space. J. Comput. System Sci. 52, 1 (1996), 43–52.

[7] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. BlackHat (2015).

[8] Andrei Tatar, Vu Amsterdam, Radhesh Krishnan, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Throwhammer: Rowhammer Attacks over the Network and Defenses. USENIX Security (2018), 213–225.

[9] Anna Trikalinou and Dan Lake. 2017. Taking DMA Attacks To The Next Level. BlackHat (2017).

[10] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. CCS (2016).

[11] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. 2018. GuardION: Practical mitigation of DMA-based rowhammer attacks on ARM. LNCS (2018).