Hash-Based Authentication Revisited in the Age of High-Performance Computers

Niclas Hedam*
IT University of Copenhagen
nhed@itu.dk

Jakob Mollerup
IT University of Copenhagen
jmol@itu.dk

Pınar Tözün
IT University of Copenhagen
pito@itu.dk

ABSTRACT

Hash-based authentication is a widespread technique for protecting passwords in many modern software systems including databases. A hashing function is a one-way mathematical function that is used in various security contexts in this domain. In this paper, we revisit three popular hashing algorithms (MD5, SHA-1, and NTLM), that are considered weak or insecure. More specifically, we explore the performance of the hashing algorithms on different hardware platforms, from expensive high-end GPUs found in data centers and high-performance computing centers to relatively cheaper consumer-grade ones found in the homes of end-users. In parallel, we observe the behavior of different hardware platforms. Our results re-emphasize that despite their theoretical strength, the practical utilization of widely used hashing algorithms are highly insecure in many real-world scenarios; i.e., cracking a password of length 6 takes less than 6 seconds using a consumer-grade GPU.

1. INTRODUCTION

Hashing is a security technique for authentication and password protection used in many modern software systems [7, 11] including database management systems. Hashing prevents passwords from being visible in files and databases that keep track of user passwords. Therefore, it prevents adversaries from gaining access to the users' password upon a database breach. For a hashing function to be secure, it must be a one-way function. Thus, it should be easy to compute the hash of a string, but hard to compute the original string of a given hash.

The security and viability of a hashing algorithm relies on the difficulty of reversing a hash to its original value or finding hash collisions. In this work, we focus on the former. A traditional approach to reversing a hash is brute-force attack [11]. A brute-force attack works by taking a predefined set of words and characters, hashing them and comparing the output to the original hash. This method relies on users having simple and unvarying passwords.

Service	Length	Alphanumeric entropy[1]
Wikipedia	1	62
Netflix	4	14,776,336
Facebook	6	56,800,235,584
Reddit	6	56,800,235,584
Amazon	6	56,800,235,584
LinkedIn	6	56,800,235,584
Instagram	6	56,800,235,584
Ebay	6	56,800,235,584
Yahoo	8	218,340,105,584,896
Google	8	218,340,105,584,896
Microsoft	8	218,340,105,584,896

Table 1: A list of the minimum password length of various popular internet services including the alphanumeric entropy.

Troy Hunt, a well-renowned security expert, has done a study on the password requirements of popular websites [5]. The results of the study can be seen in table 1 including the calculated minimum alphanumeric entropy. The entropy denotes how many guesses a brute-forcing adversary has to do before being guaranteed to guess the password, given that the password is alphanumeric and only lives up to the least secure requirements.

A brute-force attack does not usually cause an issue for online web-services since the number of password attempts per second is limited due to the internet latency as well as any security measures setup by the web-service. However, if an adversary has already breached the database of passwords and is able to download this database to a local system, then the adversary is only bound by the processing capabilities of the hardware in that local system. Today, thanks to the advances in computer architecture, even regular end-users can get access to computers that are highly powerful.

One way of limiting the viability of offline brute-forcing is to design hashing algorithms with a performance penalty. MD5, one of the most well-known hashing algorithms, induces this performance penalty by using an inner loop of

*First two authors contributed equally to the paper.

[1]The minimum alphanumeric entropy is calculated as the number of characters to the power of the minimum password length. We choose to not calculate the entropy including special-characters as the allowed special characters varies from service to service.

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and ADMS 2020. 11th International Workshop on Accelerating Analytics and Data Management Systems (ADMS'20), August 31, 2020, Tokyo, Japan.

1000 iterations [11]. When the hashing algorithm is slow, the number of hashes that can be computed per second is lower. Ideally, a hashing algorithm should be slow enough that brute-forcing is not viable, but fast enough that the algorithm is still functional for the regular end-users. A secure hashing algorithm should also give the ability to increase the performance penalty to protect passwords from the increasing computing power over time. Bcrypt is an example of a hashing algorithm with such functionality, where the administrator can define a penalty when creating a hash [11]. As a result, the bcrypt hashing algorithm can be used continuously, while algorithms with a static penalty, such as MD5 and SHA-1, have to be replaced over time. On the other hand, changing from another scheme to bcrypt increases the complexity of keeping track of which hashing scheme is used for which user. When using bcrypt, one must also keep track of which cost parameter is used for each user.

In this paper, we revisit and prove the practical insecurity of three widespread hashing algorithms: MD5 and SHA-1, two very widespread hashing algorithms with variants proven to be insecure [6], and NTLM, which is used by many Microsoft products including its operating systems. Our goal is to analyze the performance of these three hashing algorithms on a set of hardware platforms that range from expensive high-end GPUs that are commonly found in data centers and high-performance computing centers to consumer-grade GPUs that can be found at many households. Based on this analysis, we re-emphasize how insecure these hashing algorithms are, especially with the excessive availability of modern hardware today. In parallel, we discuss our key observations related to the computing power offered by a variety of GPUs.

Despite the big focus on data encryption, hash-based authentication has not been a subject that gained traction in database community even though it is widely used as the protection measure for user passwords in database systems and data-intensive applications. There has not been a thorough study of limitations and strengths of the practical ways different hash-based authentication algorithms utilized in real-world data-intensive applications and systems. We hope this focus changes over time. This work is a preliminary step in that direction, so that we can get a better understanding of a variety of methods that we use to securely access a database and data-intensive applications.

The rest of the paper is organized as follows. First, Section 2 describes our experimental setup and methodology. Then, Section 3 presents our results and Section 4 discusses the highlights of the results. Finally, Section 5 concludes.

2. EXPERIMENTAL SETUP

To analyze the practical security of MD5, SHA-1 and NTLM, our methodology is to quantify the hashing throughput achieved by different modern hardware platforms. We can, then, assess the security implications of each hashing algorithm based on these throughput measurements.

2.1 Hardware

Today, GPUs are considered commodity hardware similar to general-purpose CPUs. A brute-force attack against hash-based authentication is an embarrassingly parallel process, which is a natural fit for acceleration by GPUs. Therefore, in this paper, we focus on GPUs as the hardware platform, while experimenting with both OpenCL and CUDA as frameworks.

Table 2 lists the GPU types and setups explored in this paper. These setups are picked with the goal of maximizing variety based on what was available to us at the time of experimentation through the computing infrastructure of IT University of Copenhagen and our home computers.

The setup that represents the enterprise grade CPU-GPU co-processors is the first one in the list, where a Tesla V100 GPU is connected via PCIe 3.0 to a Intel Xeon Gold 6136 (one GPU per CPU). This setup can be categorized as modern state-of-the-art CPU-GPU co-processor hosted at data centers or high performance computing centers. Then, there are three RTX 2070 GPUs connected to a 4-core desktop CPU (Intel i7 6700k) with different characteristics: (1) one connected to CPU via the older PCIe 2.0, while the other two are connected via PCIe 3.0, and (2) among the ones connected via PCIe 3.0, one of them is overclocked. These three setups represent a relatively high-end but still consumer-grade setup. To compare, the Tesla V100 setup has an order of magnitude the cost of the setup for these RTX 2070 GPUs. Finally, there are also older consumer-grade GPUs: one GTX 1070 and one GTX 770, both connected to a desktop CPU found in most households. The variety in GPU types and setups help us to observe the processing power across generations of hardware and different budget restrictions. We aim to see the level of hardware setup it takes for a successful brute-force attack.

2.2 Workload

To generate the workload for brute-forcing, we use *hashcat* [3]. Hashcat is a hash recovery and cracking tool that includes a benchmark mode, thus allowing us to quantify the hashing throughput of the hardware platforms being used. Hashcat's benchmark mode picks the workload that would utilize the given hardware as well as possible itself. Therefore, we do not explicitly configure this workload. We validated that all the GPUs were fully utilized in all experiments. Hashcat also reports MH/s (millions of hashes per second) value while benchmarking, which we use as the hashing throughput for different hashing algorithms and hardware setups.

Based on the hashing throughput on a platform, one can reason about the difficulty/easiness of cracking passwords by calculating the time it takes for a platform to hash all the passwords of a given class. For example, alphanumeric passwords are one class of passwords that contain only the characters a to z, A to Z and 0 to 9, which is 62 unique characters. A password of this class with a length of 6 would allow for $62^6 = 56,800,235,584$ different combinations. Given a GPU with a hashing throughput of $1,000,000,000$ hashes per second, one can hash all $56,800,235,584$ alphanumeric passwords in 57 seconds.

2.3 Possible heating issues

During the initial sensitivity analysis on RTX 2070 GPUs, we observed a variation of performance for sequential runs. More specifically, the performance degraded for each subsequent run until it was eventually stable. To investigate this pattern further, we introduced some waiting time in between subsequent runs, which impacted the overall GPU performance in a non-negligible way.

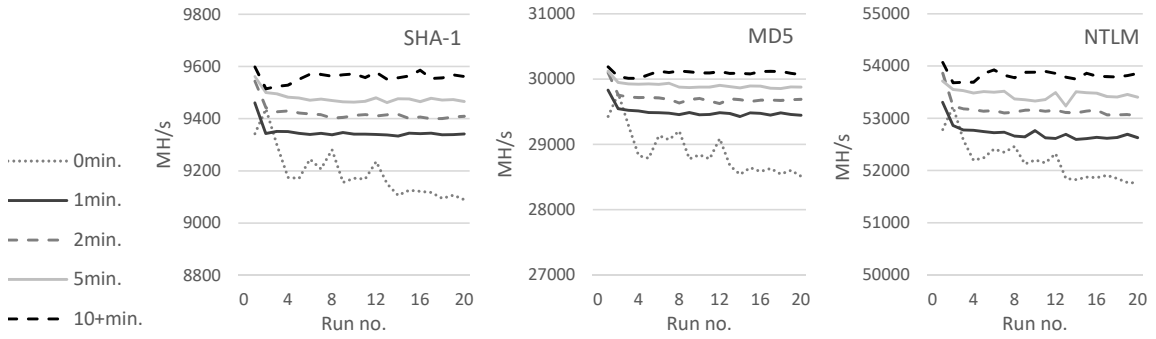

Figure 1: 20 sequential runs using the hashing algorithms with different waiting times in between runs.

GPU	CPU	Type
Tesla V100	Server	Enterprise
RTX 2070 (via PCIe 2)	Desktop	High-end Home
RTX 2070 (via PCIe 3)	Desktop	High-end Home
RTX 2070 (5% overclock)	Desktop	High-end Home
GTX 1070	Desktop	Home
GTX 770	Desktop	Home

Table 2: List of benchmarked hardware platforms.

Figure 1 plots the results of these series of runs with the hashing algorithms. Increasing the waiting time between each run lowers the negative impact on performance and yields more stable results. Increasing the waiting time to a minimum of 10 minutes results in nearly no decrease in performance.

The issue with performance instability was also occasionally observable on Tesla V100. However, the performance differences across runs were way less significant. Therefore, we only performed the sensitivity analysis with increasing waiting times on RTX 2070.

We suspect that the negative impact of running back to back experiments are due to the heating of the GPU. According to an NVIDIA product brief, the Tesla V100 GPU artificially slows down if it becomes too hot [12]. While RTX 2070 is different than V100, it is likely that a similar feature exist in the RTX 2070. When using a waiting time between runs, we expect the GPU to cool down before the subsequent benchmark run, and thus to avoid reaching the slowdown temperature.

It would have been possible to devise more detailed experiments to investigate the impact of heating more precisely using heating sensors, external cooling fans, etc. However, we only had remote access to these GPUs at the time of experiments due to COVID-19-related lockdowns. We plan to investigate this further as part of future work.

2.4 Iterations

When reporting results, in most cases, an average of three runs are used to determine the performance. On the other hand, while reporting the performance of PCIe 2.0 vs. PCIe 3.0 as well as CUDA vs. OpenCL, more runs were needed due to the close proximity of the results. Thus, 20 runs where used for these experiments. For the RTX 2070 GPUs that exhibit the heating behavior mentioned in Section 2.3, we wait 10 minutes in between each iteration.

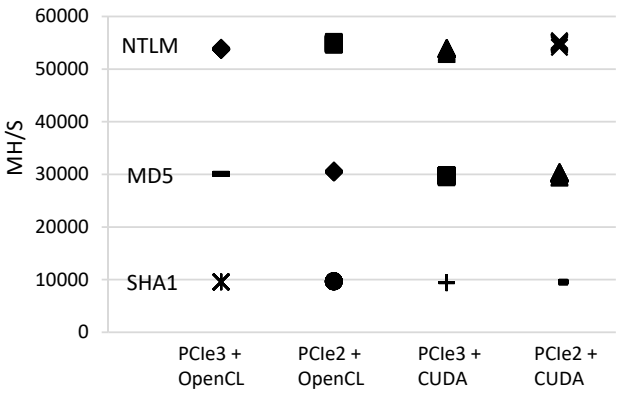

Figure 2: Performance on RTX 2070 PCIe 2.0 and PCIe 3.0 setups with the three hashing algorithms ran using OpenCL or CUDA.

3. RESULTS

The experiments can be divided into three groups based on the hardware setups the three hashing algorithms run on:

- PCIe 2.0 vs. PCIe 3.0.
- OpenCL vs. CUDA.
- Comparison across all GPU platforms including multi-GPU setups.

3.1 PCIe 2.0 vs. PCIe 3.0

The performance test comparing the impact of PCIe 2.0 and PCIe 3.0 are run using the second and third setups from the list in Table 2. Figure 2 plots the results for all hashing algorithms run using both CUDA and OpenCL. We can observe that the throughput with PCIe 2.0 connection is slightly higher than the throughput with PCIe 3.0. However, the difference in performance between the two slot types are not huge for our particular hashing algorithms. This is expected since the hash-based authentication is not a data-intensive operation. Therefore, the higher bandwidth of PCIe 3.0 cannot be utilized by this workload.

For a more detailed analysis of the results, Table 3 shows the difference in throughput between PCIe 2.0 and PCIe 3.0 for the lowest measured value, highest measured value, and average value for 20 runs. PCIe 2.0 is 1.16% - 2.07%

Slot	Alg.	Low	High	Average
PCIe3	SHA-1	9,514.50	9,598.20	9,559.60
PCIe2	SHA-1	9,626.20	9,760.60	9,670.87
Diff.	**#**	**111.70**	**162.40**	**111.27**
Diff.	**%**	**1.17%**	**1.69%**	**1.16%**
PCIe3	MD5	30,010.90	30,188.00	30,092.66
PCIe2	MD5	30,355.80	30,767.00	30,513.14
Diff.	**#**	**344.90**	**579.00**	**420.48**
Diff.	**%**	**1.15%**	**1.92%**	**1.40%**
PCIe3	NTLM	53,681.30	54,069.40	53,825.84
PCIe2	NTLM	54,643.80	55,188.00	54,823.18
Diff.	**#**	**962.50**	**1,118.60**	**997.34**
Diff.	**%**	**1.79%**	**2.07%**	**1.85%**

Table 3: The results on RTX 2070 PCIe 2.0 and PCIe 3.0 setups utilizing OpenCL - lowest, highest, and average values. Comparisons are given in numerical values (PCIe2 - PCIe3) and percentages ((PCIe2 - PCIe3) / PCIe3). All results are in MH/s.

Driver	Alg.	Low	High	Average
OpenCL	SHA-1	9,626.20	9,760.60	9,670.87
CUDA	SHA-1	9,338.70	9,644.30	9,572.54
Diff.	**#**	**287.50**	**116.30**	**98.33**
Diff.	**%**	**3.08%**	**1.21%**	**1.03%**
OpenCL	MD5	30,355.80	30,767.00	30,513.14
CUDA	MD5	29,354.60	30,455.40	30,170.52
Diff.	**#**	**1,001.20**	**311.6**	**342.62**
Diff.	**%**	**3.41%**	**1.02%**	**1.14%**
OpenCL	NTLM	54,463.80	55,188.00	54,823.18
CUDA	NTLM	54,136.80	55,425.90	54,954.17
Diff.	**#**	**327.00**	**237.90**	**130.99**
Diff.	**%**	**-0.60%**	**-0.43%**	**-0.24%**

Table 4: The results with OpenCL and CUDA on RTX 2070 PCIe 2.0 setup - lowest, highest, and average values. Comparisons are given in numerical values (OpenCL - CUDA) and percentages ((OpenCL - CUDA) / CUDA). All results are in MH/s.

faster than PCIe 3.0. Furthermore, the lowest measured PCIe 2.0 value exceeds the highest measured PCIe 3.0 value with the OpenCL driver. However, the differences are relatively small for all the hashing algorithms as also mentioned above. Therefore, the impact of using PCIe 2.0 instead of PCIe 3.0 is almost negligible for this particular workload.

3.2 OpenCL vs. CUDA

The performance test comparing the impact of OpenCL and CUDA are also run using the second and third setups from the list in Table 2. Figure 2 displays the average of the results across runs. Based on the figure, similar to the results from Section 3.1, there is no significant difference between using the different frameworks.

To look at the results in more detail, Table 4 reports the difference in throughput between OpenCL and CUDA for the lowest measured value, highest measured value, and average value for 20 runs. While the performance of hashing using OpenCL exceeds the performance when using CUDA for the SHA-1 and MD5 hashing algorithms, for the NTLM hashing algorithm, we can see that CUDA performs slightly better than OpenCL. However, the differences are low; at most 3%.

3.3 All GPU types & multi-GPU

This section compares the performance of the three hashing algorithms on all the GPUs listed in Table 2. We first focus on individual GPUs. Then, we also compare with two multi-GPU setups:

1. A distributed setup with 6 Tesla V100 GPUs, where each GPU is connected to a CPU from three two-socket Intel Xeon Gold 6136 servers, and

2. 4-core Intel i7 6700k desktop CPU connected to four RTX 2070 GPUs over a low-cost crypto mining rig motherboard ASRock H110 Pro BTC+ [1], where three GPUs are connected with PCIe 3.0, one of which is overclocked, and one is connected with PCIe 2.0 to CPU.

We use only the OpenCL driver for this comparison since we didn't observe a huge difference between OpenCL and CUDA in Section 3.2.

An overview of the results can be found in Table 5. The *normalized* column denotes the relative difference in performance with regards to a single Tesla V100 GPU. This allows us to compare GPU performance among various hardware platforms. For all hashing algorithms, using 6 Tesla V100 GPUs results in 6 times the performance of one, which shows the embarrassingly parallel nature of brute-force hashing, and also verifies the scalability of hashcat benchmark driver (Section 2.2) used by this study.

Similarly, the multi-GPU setup with four RTX 2070 also exhibits great scalability; i.e., four GPUs achieving roughly four times the performance of one. We can also observe that the different RTX 2070 GPUs behave almost identically. On the other hand, the overclocked GPU performs around 5% better.

Figure 3 shows the performance differences across all individual GPUs for all three hashing algorithms. While the price of Tesla V100 is an order of magnitude higher than the price of RTX 2070, the relative performance between the two GPUs is 2X for this workload.

4. DISCUSSION OF RESULTS

We have analyzed the performance of three hashing algorithms widely used for hash-based authentication by running brute-force attack on a variety of modern high-performance GPUs.

Initially, we explored an issue related to performance instability on the RTX 2070 GPUs connected to the desktop CPU. We found that waiting 10 minutes between each subsequent run mitigates certain negative effects of running back to back experiments such as throughput degradation and unstable throughput. We suspect this happens due to overheating of the GPUs even though we are not able to prove that heating was the exact cause of the performance degradation and instability. On the other hand, we know that some NVIDIA GPUs have a built in slowdown feature to prevent overheating [12]. If overheating is indeed the is-

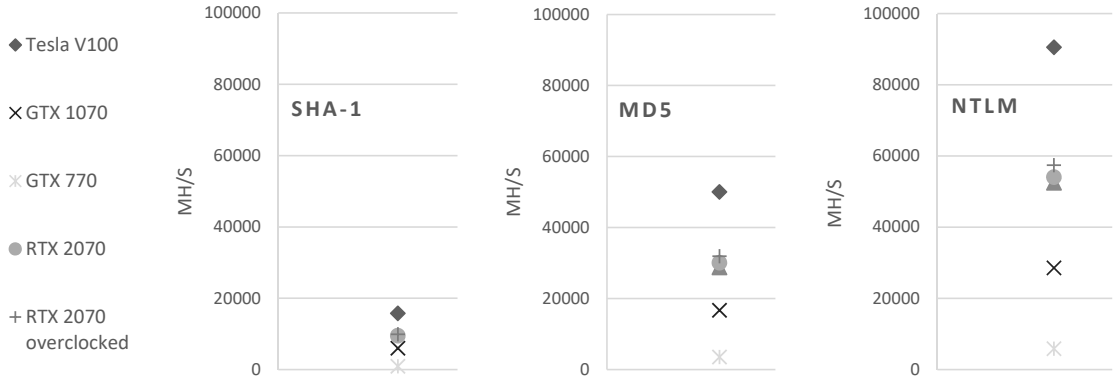

Figure 3: Performance of the three hashing algorithms on all GPUs listed in Table 2

GPU	Average	Normalized
SHA-1		
Tesla V100	15,775.60	100
6 x Tesla V100	94,854.40	601
RTX 2070	9,489.33	60
RTX 2070 overclocked	9,909.00	63
4 x RTX 2070	39,002.80	247
GTX 1070	5,983.07	38
GTX 770	941.67	6
MD5		
Tesla V100	50,056.00	100
6 x Tesla V100	300,666.67	601
RTX 2070	30,058.83	60
RTX 2070 overclocked	31,918.13	64
4 x RTX 2070	124,231.40	248
GTX 1070	16,713.83	33
GTX 770	3,525.57	7
NTLM		
Tesla V100	90,541.93	100
6 x Tesla V100	543,300.00	600
RTX 2070	54,046.73	60
RTX 2070 overclocked	57,410.53	63
4 x RTX 2070	223,606.43	247
GTX 1070	28,550.87	32
GTX 770	5,866.37	6

Table 5: Comparisons of the results using the OpenCL driver. All results are in MH/s.

Length[2] / GPU	6	8	10	12
Tesla V100, MD5	1 s	1 hr	194 d	2042 y
Tesla V100, SHA-1	4 s	4 hrs	616 d	6481 y
Tesla V100, NTLM	1 s	40 m	107 d	1129 y
RTX 2070, MD5	2 s	2 hr	323 d	3401 y
RTX 2070, SHA-1	6 s	6 hrs	1024 d	10774 y
RTX 2070, NTLM	1 s	1 hr	180 d	1892 y

Table 6: The maximum time it takes to brute-force passwords of various lengths with the OpenCL driver. All numbers are rounded.

Slot	Transfer rate	Throughput	Encoding
PCIe 2	5.0 GT/s	500 MB/s	8b/10b
PCIe 3	8.0 GT/s	1000 MB/s	128b/130b

Table 7: Transfer rates and throughput rates for the PCIe 2.0 and PCIe 3.0 slots [9, 10].

sue, then cooling of the GPU might play a considerable role when it comes to performance of various workloads.

Furthermore, we experimented with the hashing performance of two different drivers, CUDA and OpenCL, and did not observe drastic differences between the two. This comparison was only performed on one system and as such, the trends may not hold for other systems. CUDA is the proprietary GPU driver from NVIDIA [2] and contains specialized GPU accelerated libraries for various purposes such as mathematics. The hashing algorithms does not necessarily take advantage of the accelerated libraries from CUDA. Without further examining the mathematical operations used in the hashing algorithms, we cannot identify the expected performance differences between CUDA and OpenCL.

We also compared the performance difference between using GPUs mounted with PCIe 2.0 and PCIe 3.0. As shown in Table 7, the transfer rate and throughput of the PCIe 3.0 is higher than that of PCIe 2.0 [9, 10]. PCIe 2.0 uses the 8b/10b encoding, which means that to send 8 bytes of encoded data 10 bytes are transferred. PCIe 3.0 uses the 128b/130b encoding. It was therefore surprising that PCIe 2.0 slightly outperforms PCIe 3.0 with the same GPU even though the relative difference is at most 2%. The maximum throughput capacity of the PCIe connection may affect the performance of workloads with a lot of data transfers. This is not the case for hashing, as this operation only requires transferring a small set of data. Therefore, hashing algorithms are bounded by the processing capabilities of the GPU and not the bandwidth of the connection.

Finally, the main goal of this paper was to revisit the security of the widely used hashing algorithms for hash-based password authentication. The three hashing algorithms explored in this paper, SHA-1, MD5, and NTLM, are not insecure from a theoretical perspective, but can be practically insecure based on how they are used in real-world applications. As discussed in Section 1, major services allow users

[2]Assuming passwords with only alphanumeric characters.

to set passwords that are dangerously short. One particular example is LinkedIn, which allows users to set passwords of length 6 [5]. Unfortunately, the same major company had an extensive breach in 2012 exposing 164 million email and password pairs [4]. LinkedIn stored passwords hashed with the SHA-1 algorithm without any salting (adding additional bits to passwords to lengthen them before hashing) to protect the passwords.

Table 6 reports how long it would take to establish a successful brute-force attack on passwords of varying lengths using different GPUs and hashing algorithms based on the results reported at Table 5. With the average SHA-1 hashing throughput measured on a single RTX 2070, we are able to crack any alphanumeric passwords that adhere to the minimum requirements in less than 6 seconds[3]. A study of leaked passwords from other breaches showed a mean password length of 8 to 9 characters [8]. If we assume that the same distribution holds for the LinkedIn breach, we can crack all typical alphanumeric passwords in at most around two weeks[4]. Disturbingly, the RTX 2070 is a consumer-grade GPU that can be installed in many home computers.

5. CONCLUSION

Hashing is a popular security mechanism for password authentication used by many modern software systems including databases. In this paper, we revisited the practical security of the three widely used hashing algorithms (MD5, SHA-1, and NTLM). Our goal was to explore the feasibility of a brute-force attack to crack passwords. For this goal, we quantified the throughput of the three hashing algorithms on a variety of GPUs ranging from high-end to consumer-grade ones.

On the one hand, the longer and stronger passwords are still unbreakable using these three algorithms. On the other hand, we consider a hashing algorithm insecure, if an alphanumeric password of the mean length is breakable in reasonable time. Our results demonstrate that this is the case for all three algorithms. Thus, we can conclude that these three hashing algorithms are practically insecure with the hardware available to consumers in 2020 and the password requirements of major websites.

Going forward, several other performance aspects can be explored. First, the performance degradation and instability when running back to back experiments on GPUs is a very relevant topic and deserves further exploration using a variety of workloads, not just hashing, and measuring the impact of overheating more precisely. For example, a badly cooled Tesla V100 GPU may quickly reach 87°C and have its clock-rate artificially slowed down to 50%. Reaching 90°C can fully shutdown the GPU to prevent permanent damage. As GPUs are becoming commodity and more widely available to end-users, especially thanks to the rise of machine learning, it is highly important to understand the relation between their energy consumption, heating, and throughput more thoroughly.

In addition, performing the same study on other types of accelerators such as GPUs from other vendors (e.g., AMD), FPGAs, SIMD, etc. would be interesting to better understand the robustness of different password lengths using hash-based authentication. In such a study one can also dig deeper into the the impact of different drivers (OpenCL, CUDA, etc.) on different hardware platforms/accelerators.

Finally, a thorough survey and analysis of different authentication methods for data-intensive systems and applications would be very valuable.

[3] $(26+26+10)^6/9,489,330,000 = 5.9$ seconds.

[4] $(26+26+10)^9/9,489,330,000 = 1.427 \cdot 10^6$ seconds $= 16.51$ days.

References

[1] AsRock. *AsRock H110 Pro BTC+ Motherboard*. https://www.asrock.com/mb/Intel/H110\%20Pro\%20BTC+/index.asp. 2019.

[2] *CUDA Toolkit*. NVIDIA Corporation. URL: https://developer.nvidia.com/cuda-toolkit (visited on 05/02/2020).

[3] *Hashcat*. URL: https://hashcat.net/ (visited on 04/23/2020).

[4] *Have I Been Pwned: Pwned websites*. URL: http://haveibeenpwned.com/PwnedWebsites (visited on 05/02/2020).

[5] Troy Hunt. *How Long is Long Enough? Minimum Password Lengths by the World's Top Sites*. Feb. 6, 2018. URL: https://www.troyhunt.com/how-long-is-long-enough-minimum-password-lengths-by-the-worlds-top-sites/ (visited on 04/30/2020).

[6] Roman Jasek, Libor Sarga, and Radek Benda. "Security Review of the SHA-1 and MD 5 Cryptographic Hash Algorithms". In: 2013. URL: https://pdfs.semanticscholar.org/32bc/0b6b51905073890df67e2cc236d23726dd72.pdf (visited on 04/09/2020).

[7] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014, pp. 184–185. ISBN: 9781466570269. URL: https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269.

[8] Theodosis Mourouzis, Kyriacos E. Pavlou, and Stylianos Kampakis. "The Evolution of User-Selected Passwords: A Quantitative Analysis of Publicly Available Datasets". In: *CoRR* abs/1804.03946 (2018). arXiv: 1804.03946. URL: http://arxiv.org/abs/1804.03946.

[9] *PCI Express® 3.0 Frequently Asked Questions*. URL: https://web.archive.org/web/20140201172536/http://www.pcisig.com/news_room/faqs/pcie3.0_faq/#EQ2 (visited on 05/02/2020).

[10] Martin Rowe. *What does GT/s mean, anyway?* Mar. 2007. URL: https://www.edn.com/what-does-gt-s-mean-anyway/ (visited on 05/02/2020).

[11] William Stallings and Lawrie Brown. "Computer Security: Principles and Practice". In: 4th ed. Pearson Education, 2018, pp. 92–98. ISBN: 978-1-292-22061-1.

[12] *TESLA V100 PCIe GPU ACCELERATOR*. 5th ed. NVIDIA Corporation. Mar. 2018. URL: https://images.nvidia.com/content/tesla/pdf/Tesla-V100-PCIe-Product-Brief.pdf (visited on 05/02/2020).