Communication Papers of the 18th Conference on Computer Science and Intelligence Systems pp. 61–64
DOI: 10.15439/2023F5820
ISSN 2300-5963 ACSIS, Vol. 37

Proof-of-Work CAPTCHA with password cracking functionality

Szymon Chadam*, Paweł Topa†
Faculty of Computer Science, Electronics and Telecommunications,
AGH University of Kraków
Kraków, Poland
Email: *szymon@chadam.pl,
†topa@agh.edu.pl

***Abstract*—This document proposes an alternative CAPTCHA system that implements a proof-of-work mechanism to protect resources (usually web services) from being accessed by automatic entities called bots. Normally, CAPTCHA forces the user to do some work in order to prove that he is not a machine. The proposed system utilizes a novel alternative to Proof-of-Work algorithm that utilizes the user's computing power to crack password hashes.**

I. Introduction

THE PROBLEM of unwanted traffic and automated applications and scripts (commonly referred to as bots) has plagued the Internet almost since its inception. Netacea estimates that unwanted bot traffic costs companies up to 250 million annually [1]. Existing solutions such as reCAPTCHA capture almost 97% of the market [2]. However, Google's solution raises privacy concerns [3] as it collects and stores information about the user's browser.

This paper presents an alternative CAPTCHA-like system. This system uses a Proof-of-Work mechanism to impose a computational cost on the user. The user's computational power is used to attempt to crack password hashes provided by the system. Based on other metrics, the computing power required to access an online service can be arbitrarily increased to combat unwanted traffic.

A common method of storing passwords securely is to convert the password phrase into a hash, which is generated by a hash function or key derivation function, and store it in a database. The properties of the hash function or KDF function ensure that the input phrase cannot be reconstructed from the hash. The only effective method of attacking such protected passwords is dictionary cryptanalysis, i.e. calculating hashes for a huge dictionary of potential passwords and comparing the calculated hashes with the attacked password.

Normally, it is undesirable for users' passwords to be cracked. However, in the case of law enforcement, we often need to obtain suspects' passwords in order to access encrypted evidence. The obvious solution is to build powerful (and expensive) dictionary cryptanalysis computers. A less obvious approach is to use the distributed power of web users' computers, as has been done in the Seti@Home (https://setiathome.berkeley.edu/ — suspended project) or Folding@Home projects (https://foldingathome.org/). The proposed approach can therefore support law enforcement activities while providing the desired functionality to the web community.

The paper is organized as follows. The next section shortly reminds the history of CAPTCHA mechanisms. Section III presents the general idea of the Proof-of-Work and its applications. Chapter IV cites examples of several solutions to stop unwanted web traffic. The next section presents details of the proposed solution. At the end, we add some concluding remarks.

II. Overview of the current CAPTCHA solutions

CAPTCHA ("Completely Automated Public Turing test to tell Computers and Humans Apart") is a type of challenge–response test used in computing to determine whether the user is human [4]. Since its inception, there have been many implementations with varying characteristics and effectiveness.

Released in 2007, the first iteration of reCAPTCHA (reCaptcha v1) asked the user to re-type a blurry and distorted set of words to gain access to the web resource. With the increasing rise and accuracy of optical character recognition software, a second version of reCAPTCHA has been introduced, taking a different approach. This time, the user was asked to select pictures containing a given object out of a set of images. Finally, in 2017 Google released reCAPTCHA v3 which allows verifying whether the user is a bot without any additional user interaction. The mechanism tracks user's interaction with the website and returns a score determining how likely it is that this user is a bot.

While all the solutions mentioned above work in stopping unwanted traffic, they also significantly cause reduced User Experience. Selecting images of traffic lights or fire hydrants has become an inseparable and frustrating part of numerous online forms. Additionally, even the most recent reCAPTCHA solutions implementing artificial intelligence can be bypassed with more than 90% success rate [5]. Moreover, there are numerous Captcha Solving Services [6] that provide correct solutions for given CAPTCHAs for a small fee making it trivial for sufficiently motivated opponents to bypass all of the most popular CAPTCHA implementations.

Thematic track: Cyber Security, Privacy and Trust

III. Proof-of-Work mechanism

Proof-of-Work is a mechanism designed to provide the verifying party a cryptographic proof that the user utilized a specified amount of computing power to perform a task. While there are many approaches to implementing a Proof–of–work algorithm, the most popular approach is to perform a digest (or hash) function of a given data alongside a nonce value until certain criteria have been met. This approach is based on the fact that a good digest function is preimage resistant which means that it is computationally infeasible to find any input m that has a given digest $h = H(m)$. The only way of finding such a message is brute-force. An example of such criteria can be the number of zeroes in the binary representation of the resulting hash. If the output of the hash function does not satisfy it, the nonce value is incremented and the whole process starts from the beginning. Upon finding the correct nonce value that produces a hash output described by the verifying party — the user sends the nonce value to be verified. The party in charge of verifying the work performed by the user needs to perform only one hash function with the nonce value sent by him to determine whether the proof is correct or not. This property makes it easy for the system to scale for a large number of users while keeping only one party responsible for the verification of the Proof–of–Work. Worth noting here is the fact that current Proof-of-Work algorithms require the solver to find an exact nonce value rather than a range of acceptable solutions. Additionally, Proof–of–Work has been criticized for the high energy consumption required to perform the task.

IV. Related work

There have been numerous attempts to stop unwanted web traffic by utilizing users' computing power. Some of them are described below.

A. Hashcash

Proposed in 1997 by Adam Back, Hashcash[7] is a mechanism originally proposed to combat email spam abuse. When sending an email, the user performs a Proof-of-Work algorithm on the whole body of the email message. Additional data is appended such as timestamp and string of random characters. Computation is performed until the sender reaches a desired number of zero bits in the hash output and a resulting counter value is obtained. Finally, a new header is appended to the email containing information used to prove the work. Although the proposed system did not achieve significant adoption, a version similar to it has been implemented into Bitcoin's [8] mining mechanism.

B. CoinHive CAPTCHA

Around the year of 2018, a company called CoinHive launched its Proof-of-Work captcha widget, a reCAPTCHA alternative where a user visiting the website can commit a portion of his device's computing power to mine cryptocurrency for the website owner instead of selecting a set of images containing a specified item. Unfortunately, CoinHive's solution became widely used for attackers to perform cryptojacking [9] — an attack when a user's computing power is used to mine cryptocurrency for the attacker.

C. Cloudflare Turnstile

Cloudflare, a company specializing in DDoS attack mitigation has introduced its own CAPTCHA alternative with Proof-of-Work mechanism. Turnstile [10] aims to replace frustrating CAPTCHAs by utilizing a set of non-interactive JavaScript challenges. Some of that challenges require the user's device to perform computations similar to how the Proof-of-work mechanism works. Although Cloudflare's solution is relatively similar to the one proposed in this paper, the computing power used for its challenges is not used in a way that can be used for other purposes.

V. Proposed solution

As shown above, a sufficiently motivated opponent should not have any difficulties bypassing the most popular CAPTCHA implementations. Taking that into consideration, an alternative approach has been taken. The proposed solution borrows some of its features from Hashcash [7]–style Proof–of–Work with additional mechanisms. Instead of requiring from user to select a set of images, the proposed solution utilizes a small portion of user's computing power to solve a cryptographic puzzle. Moreover, rather than utilizing CPU power to compute meaningless hash functions that can be seen as a waste of electricity, the system uses it to perform brute-force attacks on password hashes stored within the system's database.

Instead of specifying a strict value for the target as is usual in Proof-of-Work implementations, the system provides an *upper* and *lower* bounds of target values of the resulting hash. These bounds can be arbitrarily changed by the system to reduce the puzzles complexity or make it more difficult for suspicious traffic.

This approach can be visualized by plotting all possible target values with the length of *n* bytes in a circle as shown in Figure 1. The resulting image resembles a clock, where the upper and lower bounds can be thought of as the hands of that clock. The radius of the bounds is the range for which the hash value must be found in order to complete the puzzle.

Additionally, to correctly test the entire set of characters, the system provides the user with the *starting_point* value which binary representation should be considered as starting point for the puzzle and any value below it will be rejected by the system as an incorrect solution.

A sample CAPTCHA puzzle request has been presented below. The *token* value can be treated as a random ID used to identify the puzzle within the system.

```
{
    "hash_type": "SHA256",
    "starting_point": "20AA",
    "lower_target": "11FF41",
    "upper_target": "3228AF",
    "token": "c41...4e9"
}
```

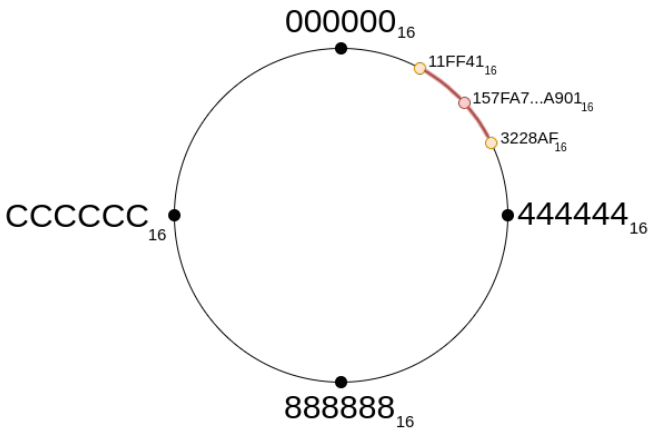

Figure 1. Visualistion of 3-byte space of values encoded in hex. Lower and upper target bound values have been highlighted in yellow while the target hash value has been represented by the red dot.

Next, the user attempts to solve the puzzle by incrementing the value of *starting_point* decoded to its binary format. The resulting binary data is then hashed using the digest algorithm negotiated between server and client and a hash value is obtained. Finally, the users compare the first *n* bits of resulting hash where *n* is the length of the *upper_target* in its binary form with the upper and lower target values. If the resulting value fits within the boundary selected by the system, the puzzle has been completed successfully.

Once the correct solution has been found, a preimage of the resulting hash is sent to the server for verification. If the verification process is completed, access is granted. A sample response to the system has been presented below.

```
{
    "token": "c41...4e9",
    "value": "20AA221F"
}
```

In short, the system guides the solver to the hash it is looking for, while implementing certain character requirements that must be met to mark the CAPTCHA puzzle as complete. The lower and upper bounds are used to adjust the difficulty of the puzzle and guide the solver toward the system's desired hash, while the starting point value is used to ensure that all possible values have been checked.

By carefully tweaking the bounds and starting point values for every user or for a batch of users, the system can test the entire range of values by utilizing users' computing power.

A. Puzzle verification

By referencing the *token* value within the system's internal database, information about the digest function, start point, and target bounds is obtained, along with a timestamp indicating when the puzzle request was made. The system then hashes the value provided by the user with the appropriate hash function and checks that it fits within the target bounds. Next, the system checks whether the value provided by the user is greater than the given starting point. Finally, the server checks the time difference between the generation of the puzzle and the submission of the solution to enforce a time-based expiration of the CAPTCHA. Only when all checks have been successfully completed can the user be granted access.

Additionally, to detect potentially malicious behavior, the system calculates an expected average amount of hash calculations that need to be performed in order to satisfy the puzzle where n is the length of the hash function in bits.

$$x = \frac{2^{(n-1)}}{upper\_target - lower\_target} \tag{1}$$

Similarly, the amount of hashes calculated if the user was honest is derived by subtracting the user-supplied solution by the puzzle *starting_point*.

$$Hashes\_Calculated = Solution - Starting\_Point \tag{2}$$

In a case where a user-supplied solution requires much more hash functions to be calculated than what was expected by the system, access still should be granted to the user to avoid false positives, but his solution should not be taken into account when generating new starting point value for the next batch of users.

Worth noting here is the fact, that when the server is verifying the puzzle, it also compares the resulting hash with the hash the system is trying to crack. Additionally, even when the user correctly finds the hash preimage, he can not distinguish that his solution in fact cracks the hash — server responds to the correct solution value in the same manner.

B. Adjusting starting point value

In a perfect world, where all users are honest, the starting point of one user should be equal to the solution value of the previous user incremented by 1. Unfortunately, that assumption can not be made. Despite this, that approach is still viable when a larger batch of users is taken into account. The system should serve the same puzzle (although with a different *token* value) to a small group of users. Only after a sufficient number of solutions have been proposed, the server can treat the solution as correct and correctly incremented if at least 51% consensus in regards to submitted value has been achieved amongst these users. After that, *starting_point* value of the next batch of users should be the same as the solution of the previous batch incremented by 1. Ideally, the system should contain many hashes ready to be converted to CAPTCHA puzzles to avoid the situation where one user lands in the same batch and receives the same puzzle more than once.

C. Puzzle difficulty adjustment

Just like all the other CAPTCHA solutions, there is a need for a mechanism to make the puzzle easier or harder for the user to solve. In the case of the proposed solution, decreasing the radius between the lower and upper bounds acts as a tool for difficulty adjustment. By making the bounds closer together, the system decreases the number of potential values that can be used as a solution.

VI. Benefits of the proposed solution

The proposed solution provides an interesting alternative to the popular image-based CAPTCHA system. It requires no user interaction making the user experience seamless and better suited for users with accessibility issues. Additionally, the system successfully prevents attack utilizing machine learning as the best strategy for solving the cryptographic puzzle is by a brute–force attack.

Compared to solutions that use essentially meaningless computation tasks, proposed solutions manage to utilize users' computing power to achieve an ultimately beneficial goal. The ability to quickly and cheaply crack password hashes of seized, but encrypted devices, would be of great help for law enforcement bodies. By giving a purpose to the computation task that would have to be performed nonetheless, the overall energy consumption is reduced. Moreover, there is a clear need for a Proof–of–Work style verification as shown by the Cloudflare Turnstile solution (see Section IV-C). The proposed solution can be thought of as an improvement to aforementioned mechanism, adding a meaningful purpose to the computational task..

VII. Threat modeling

In this section, we analyze how the proposed solution could be attacked to gain an unfair advantage or bypass it completely.

A. Low electricity cost

The cost of electricity can be widely different across different regions of the world making it difficult to estimate the amount of computing power that would be considered sufficient to scare off the attacker. As a result, the server responsible for generating and scheduling the puzzles should dynamically adjust the difficulty based on a range of gathered metrics. Aside from the IP address metrics could include data gathered by the JavaScript code run on the solver's machine such as time zone, default language, or hardware information. The cost of electricity used to solve a standard task has not been measured and are a subject of further research.

B. Use of FPGA, ASIC or botnet network to solve cryptographic puzzles

A dedicated attacker could use a botnet network to harvest cheap and accessible computing power for solving cryptographic puzzles and abusing the system. This case can be compared to existing and commonly used services of independent CAPTCHA solvers. Worth noting is the fact that increased CPU usage and power consumption make the botnet more susceptible to detection as the victim's computers would be negatively affected by the computation. Additionally, the time needed to solve a cryptographic puzzle can be vastly reduced by specialized hardware such as FPGA or ASIC. While this attack would in fact be successful, it would require a large upfront investment by the attacker on top of the already increased electricity bill. We are not able to share performance benchmarks of the proposed solutions as proposed solution is still at the work–in–progress stage.

In general, we conclude with the estimate that the security implications of the proposed solution are similar to that of the modern CAPTCHA solutions used across the web today.

VIII. Conclusion

The system introduces a real-world cost on the attacker requiring a greater electricity use to successfully pass the puzzle. Combined with unobtrusive network traffic analysis techniques such as originating IP address it can reduce the complexity of the puzzle for users categorized as low-risk. Computing power used to solve cryptographic puzzles is used to crack password hashes stored within the system's database and can help law enforcement gain access to seized devices. The system could potentially be adapted to utilize computing power already used for Proof–of–Work style verification, thus reducing overall electricity usage and introducing an additional benefit to the computation.

Acknowledgment

The research presented in this paper was realized with funds from the Polish Ministry of Science and Higher Education assigned to AGH University of Krakow and it was supported in part by Cybercrypt@gov project (no. DOB-BIO9/32/03/2018).

References

[1] Netacea, "Businesses lose up to $250m every year to unwanted bot attacks," https://netacea.com/blog/businesses-lose-up-to-250m-every-year-bots/, [Accessed 13-March-2023].

[2] Wappalyzer, "reCAPTCHA market share compared to an alternative hCAPTCHA," https://www.wappalyzer.com/compare/recaptcha-vs-hcaptcha/, [Accessed 13-March-2023].

[3] Fastcompany, "Google's new recaptcha has a dark side," https://www.fastcompany.com/90369697/googles-new-recaptcha-has-a-dark-side, [Accessed 13-March-2023].

[4] Wikipedia, "CAPTCHA — Wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/CAPTCHA, 2023, [Accessed 04-February-2023].

[5] I. Akrout, A. Feriani, and M. Akrout, "Hacking google recaptcha v3 using reinforcement learning," 2019. doi: 10.48550/ARXIV.1903.01003. [Online]. Available: https://arxiv.org/abs/1903.01003

[6] 2Captcha, "a captcha solving solution," https://2captcha.com/, [Accessed 13-March-2023].

[7] A. Back, "Hashcash – a denial of service countermeasure," 2002, [Accessed 06-February-2023]. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf

[8] S. Nakamoto, "Bitcoin: A peer–to–peer electronic cash system," 2008, [Accessed 02-May-2023]. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[9] Interpol, "Cryptojacking," https://www.interpol.int/en/Crimes/Cybercrime/Cryptojacking, [Accessed 02-March-2023].

[10] Cloudflare, "Turnstile," https://developers.cloudflare.com/turnstile/, 2023, [Accessed 25-February-2023].