Reconstructing Hash Reversal-Based Proof of Work Schemes

Jeff Green, Joshua Juen, Omid Fatemieh, Ravinder Shankesi, Dong Jin, Carl A. Gunter
University of Illinois at Urbana-Champaign

Abstract

Proof of work schemes use client puzzles to manage limited resources on a server and provide resilience to denial of service attacks. Attacks utilizing GPUs to inflate computational capacity, known as *resource inflation*, are a novel and powerful threat that dramatically increase the computational disparity between clients. This disparity renders proof of work schemes based on hash reversal ineffective and potentially destructive. This paper examines various such schemes in view of GPU-based attacks and identifies characteristics that allow defense mechanisms to withstand attacks. In particular, we demonstrate that, hash-reversal schemes which adapt solely on server load are ineffective under attack by GPU utilizing adversaries; whereas, hash-reversal schemes which adapt based on client behavior are effective even under GPU based attacks. [1]

1 Introduction

Denial of Service (DoS) attacks are a serious threat to the availability and reliability of Internet services [5, 7, 6]. Numerous DoS defense mechanisms have been proposed in the literature. One common form of defense is to use CAPTCHA mechanisms [22]. These mechanisms depend on human presence to solve the puzzles and are not appropriate for protecting automated protocols [12]. Recent work [21] suggests that, even given human presence, proof of work schemes may be a better fit for regulating requests in some settings.

We focus on Proof of Work (PoW) mechanisms, in which a server demands that clients prove they have done work before it commits resources to their requests [30, 8, 13, 27, 23]. Most PoW mechanisms are puzzle-based techniques in which clients solve processing intensive puzzles. Puzzle-based schemes have been proposed for defense in a broad range of contexts. For instance, *Hash Cashes* [10] are puzzle-based mechanisms that aim to prevent an attacker from sending too much spam. Other classes of PoW schemes are discussed in detail in [29].

One challenge with such schemes is that valid clients may have differing computational capabilities. As attacks use more resources, and consequently the puzzle difficulties increase, weaker valid clients may experience unacceptable requirements to obtain service. While disparity is inevitable, it has been argued that these schemes can be effective if they adapt to attackers dynamically. Parno *et al.* [27] observe that a while there is a computational disparity of approximately 40x between high and low-end computing systems; this disparity is low enough for CPU cycles to be useful as a DoS countermeasure. While computationally weaker clients would experience longer latencies during an attack, it would be significantly more functional than a protocol without the PoW based defense. In this paper, we reconsider these claims in view of adversaries that utilize *resource inflation* techniques to significantly increase their computational capacity. If powerful enough, the arguments that disparity is acceptable fail, as valid clients are unable to obtain service.

Using Graphical Processing Units (GPUs) offers a powerful technique for launching resource inflation attacks. We show that attackers can use cheap and widely available GPUs to inflate their ability to solve typical hash reversal based puzzles by a factor of more than 600. This and other resource inflation techniques are discussed in more detail in our technical report [29]. While the previously proposed adaptive puzzle schemes [30, 26] may be able to tolerate a disparity of 40x, we show that they fail when faced with a 600x disparity. In view of recent interest in using PoW for purposes as diverse as improving concert ticket sales [21] and peer-to-peer currency [1], it is useful to consider the impact of GPU based resource inflation on these systems.

A contribution of this paper is the evaluation of Hash-Reversal PoW schemes in the presence of resource-inflated attackers. We show that client-based adaptation is necessary for providing adequate service to legitimate clients in this situation. Additionally, we show that an adaptive hash-reversal PoW scheme based only on server load will fail to provide service, and can create a novel DoS attack against legitimate clients. Thus, we propose the use of protocols which adapt based upon client behavior. These are shown to be effective. Given these results, hash reversal PoW schemes proposed for DoS protection mechanisms should keep track of client behavior given the emerging threat of GPGPU based attacks.

2 Background

2.1 Proof of Work

PoW schemes such as [30, 8, 13, 27, 20] are designed to limit the load attackers can impose. In these schemes, a server demands that the clients submit a "proof" of the "work" they have performed, before servicing their requests. The exchange in a typical PoW scheme is illus-

[1] In the 4th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '11), Boston, MA, Mar 2011.

trated in Figure 1. PoW schemes are based around multiple types of puzzles, but by far the most widely used are hash reversal puzzles.

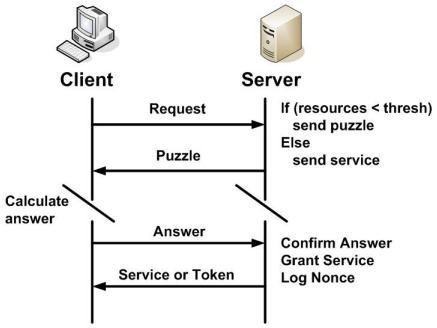

Figure 1: PoW Client-Server Interaction

PoW schemes have something of a "checkered past." Very promising [19] when initially introduced, PoW schemes were largely written off after a paper in 2004 titled "Proof-of-Work' Proves Not to Work" [24]. However, the paper contained a miscalculation and did not consider adapative puzzle schemes. There have been good cases for the efficacy of these schemes [25, 17] as well as promising new schemes [16, 14, 1, 21] in the years following.

Notably, recent PoW schemes rely on tracking individual client behavior, and require binding a client to an ID. Usually, merely including the client's IP in construction of puzzles, and requiring that a client be able to receive messages at the IP it claims to send from is considered sufficient.

2.1.1 Hash Reversal Puzzles

For a hash-reversal puzzle, the server presents a puzzle s to the client. The client finds the solution, x, by computing the hash $p = H(x||s||r)$, where r is a set of parameters whose exact contents are dependent on the puzzle scheme in question. The server verifies that the last l bits of p are zeros and that the client has used a valid seed. The value l determines the *difficulty level* of the puzzle. The work required to solve a puzzle of difficulty level l is of the order 2^l. There are a few variations of the general scheme above, which allow the server to enforce a more fine-grained amount of work on the client. One such is using multiple sub-puzzles [15].

A useful characteristic of a hash-reversal-based puzzle is that the cost paid by the server is low (generally a single hash computation and some verification) compared to the cost paid by the client. Also, generating harder puzzles does not require additional work. One major concern with these puzzles is that they can be parallelized.

2.2 Adaptation

It is desirable for PoW schemes to *adaptively* require payment from clients. Most often this adaptation pertains to the demand for server's resources. The puzzle cost should scale high enough to make it infeasible for the attacker, yet not scale to greater difficulty than is necessary for legitimate clients. Adaptation mechanisms proposed generally fall into one of the following categories: (1) *auctions* by the server (2) *probabilistic selective processing* at the server and (3) *ramp-up* at the client.

One sophisticated adaptation technique for varying difficulty based on server load is presented by Wang *et al.* [30]. This auction-based framework provides a server with a buffer of task request, solution difficulty pairs. The buffer is sorted by solution difficulty. When the buffer fills, lower difficulty requests are discarded. Clients failing to receive service increment the difficulty of solution they provide with their request. The authors show that this mechanism is efficient in the sense that the client can raise its bid just above the attacker's bid to win an auction.

2.3 Interval-based Proof of Work

The vast majority of PoW schemes, as mentioned previosly, require one puzzle to be solved per task requested. Schemes proposed by Waters et al [31], and more recently kaPoW, and mod_kaPoW [14] are fundamentally different. A server running kaPoW will provide one puzzle to each client per time interval. Once a client has solved that puzzle, the client has access to the server until the time interval expires. Additionally, unlike most PoW schemes, individual client behavior is tracked, and each client's puzzle difficulty is adjusted based on that client's past behavior. Client behavior is tracked through the use of a client side nonce which contains randomly generated numbers along with the client's IP address [14, 17].

3 Resource Inflation using GPUs

Modern GPUs are very efficient at processing large amounts of data in parallel. Unlike modern CPUs, which are designed to efficiently optimize the execution of single threaded programs using complex out-of-order execution strategies, a modern GPU's efficiency comes from executing massively data-parallel programs. This is often referred to as Single Instruction Multiple Data (SIMD) programming.

Recently, there has been significant interest in using GPUs for non graphical computation. This paradigm is known as General Purpose GPU (GPGPU) computing or Stream Computing. GPGPUs have been used to improve the speed of various programs such as Folding@Home [3], and computational chemistry [4]. Recently, GPUs were also used for finding MD5 chosen-prefix collisions [11].

GPU-based architectures consist of a large number of SIMD engines. Each engine contains a number of thread-processors. Each thread processor unit has access to its own general purpose registers as well as access the GPU's memory. The thread dispatcher manages various threads and is invoked by the client on the CPU.

Threads in GPUs are not analogous to processor threads on a general purpose computer. A GPU-based thread is a very lightweight thread that can be started with minimum overhead. All GPU-based threads (within a single SIMD engine block) need to execute the same instruction (pos-

sibly on different input data) for maximum efficiency. In effect, we cannot use different threads to perform completely distinct computations on different pieces of data. When two (or more) threads running on a thread processor in an SIMD engine need to execute different instructions, the GPU will ensure that only one of the threads executes at any given time. Although GPUs only support this limited notion of parallelism, for the right kinds of processing algorithms GPUs offer large advantages in efficiency.

3.1 GPGPU Programming Model

A GPU-based program is usually written in a way that takes advantage of the inherently data parallel programs (such as matrix multiplication, simulations, *etc.*). The GPU-based platform can run a program such that each thread of the program operates on a distinct block of data. All of such threads can run simultaneously on the stream processors as long as there are enough stream processors on the GPU. In case the data elements are larger than the number of stream processors, the Thread Dispatcher manages the available processors for the threads. For instance, consider the following snippet of code written in AMD's stream computing language:

```
kernel void
sum(float a<>, float b<>,
    out float c<>)
{ c = a + b; }
```

The inputs are the streams `a` and `b` and the output is the stream `c`. When this program, along with the stream reading and writing operations (not shown) is run, the GPGPU infrastructure reads the input streams from the CPU's main memory into the GPU's memory. Once loaded, each stream processor will run simultaneously and independently on a slice of the input, corresponding to the two input vectors and produces the result in the output. This program computes the sum of two vectors `a,b` and stores it in the output vector `c`.

Although each of the threads is working on a different piece of data, they are each effectively performing the same operation. This results in maximum efficiency in a data-parallel algorithm on a GPU.

3.2 Solving Hash Reversal Puzzles using GPUs

We take advantage of the inherently data-parallel nature of hash-reversal puzzles when solving them on the GPU and investigate the previously unconsidered impact of these techniques on PoW schemes . Solving these puzzles can be accelerated using GPUs in the following two ways. In the first approach, we can run each potential solution x to the hash puzzle as an independent thread on the GPU. In the second approach, given n puzzles with puzzle difficulty level l, we run each puzzle in a single thread that can be run simultaneously in the GPU (*i.e.*, ideally on as many individual processors as on a given GPU). Each thread effectively

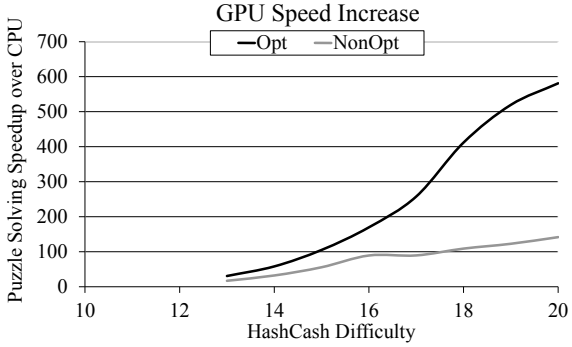

Figure 2: Resource-Inflation achieved by GPUs (Nvidia 9800 GX2 graphics card) over CPUs (Intel Core 2 Q9300)

searches a 2^l search space for the puzzle solution. Since all threads are running the same code, the GPU can run all threads *simultaneously*. If the hash computation were somehow *data-divergent*, the GPU threads would not be as efficient. The only data-divergence between various puzzle solving threads occurs when we verify if the last l bits of the hash (for a given guess x) are all 0. When this occurs, the successful thread stalls the other threads for a few instructions (while storing the solution). Since this occurs only once per puzzle, blocking has minimal effect on the efficiency.

In order to simulate the effects of an attacker utilizing a GPU for Hashcash calculation, the Hashcash algorithm was implemented on modern GPU hardware from NVIDIA. Figure 2 shows the speedup in using an NVIDIA GeForce 9800 GX2 graphics card using NVIDIA's CUDA programming API over a powerful desktop CPU. The 9800 GX2 provides access to 256 stream processors. For comparison, the SHA hashing algorithm was also run on an Intel core 2 Q9300 processor. Both tests were conducted on a Ubuntu machine using the standard C implementation of SHA-1 from RFC 3174[28]. The CPU hashcash solver's time increases throughout the experiment. Realistically, puzzle difficulty levels beyond twenty with over twenty seconds of solving time would be infeasible due to the decrease in performance for the end user; however, even the non-optimized GPU hashcash algorithm easily defeats puzzles up to a difficulty of twenty five in less than five seconds. The optimized hashcash algorithm is able to defeat puzzles up to a difficulty of twenty seven in less than five seconds. Comparitively, it took the CPU an average of ten minutes to solve a puzzle with difficulty of twenty seven. The experiments demonstrated that the GPU can be used to substantially decrease the solving time of hashcash puzzles with reasonable difficulty levels.

4 Proof of Work Schemes

We consider two primary types of PoW schemes. The first and more popular type requires one puzzle to be solved for each task requested [12][9]. For instance, if a client wishes to send ten emails, the client will be asked to solve ten puzzles. The second type requires the client to solve a puzzle

in order to get service for an interval of time [14]. Each of the two schemes is intended to limit the amount of service a greedy or malicious client can obtain.

4.1 Task-Based

A brief summary of a task-based PoW exchange follows. A client sends a request for service. The server responds with a puzzle of some difficulty. Once the client solves the puzzle, it sends an answer along with the original request to the server. If the answer is valid and resources are available, the server processes the request . If the answer is invalid or all server resources are in use, the server drops the request. This process is repeated for each client request.

One criteria for determining the effectiveness of a task-based PoW is determining the puzzle difficulty. Puzzle difficulty can be determined in a number of ways, including fixed difficulty for every request, adapting difficulty based on server load, or adapting difficulty based on client behavior. Adaptation based on server load increases the difficulty of the puzzles as the server runs out of resources. Notable examples of such schemes can be found in literature [30, 26].

The idea behind client based adaptation is to penalize clients who request many resources by giving them harder puzzles than clients with low request rates. In order to investigate the effectiveness of each type of adaptation, we employ simple algorithms demonstrating scaling based on server resources and client request behavior.

The algorithm used for scaling difficulty based on server resources utilizes a linear calculation to assign a puzzle difficulty between some arbitrary maximum and minimum. The easiest puzzles are provided when all server resources are available, and the hardest when none are available.

The algorithm used to scale difficulty based on client behavior requires the server to maintain a table of current clients along with a trailing average request rate for each client and a current puzzle difficulty. We assume that client behavior will be tracked through the use of a binding client nonce which includes identifying information, including the client's IP address. The server will keep track of a hash of the client's binding nonce as well as the number of requests in this time period and the trailing average. This reduces the state to three integers per client, allowing the server to track clients for a modest cost. Spoofing is difficult due to the client needing to receive requests at the claimed address. For more information regarding client tracking see [14] [17]. The state of client behavior is updated by the server at set intervals. At the start of each interval, the server updates the request rate for each client and checks to see if the rate exceeds the threshold. If a client's request rate exceeds the threshold, its difficulty is incremented, and its average request rate is reset to zero. Alternately, if the client's average request rate drops below half of the threshold, that client's difficulty is decremented. In both cases the average is reset so only clients who continually request excessive traffic are penalized.

4.2 kaPoW

As previously mentioned, kaPoW schemes operate differently than task-based PoW schemes. Because of the manner in which they work, tracking client behavior is intrinsic to these schemes. The way in which difficulty is determined for each client is designed to punish truly greedy clients (most often attackers) quite severely, and create a stable situation where all well-behaved clients can attain service. The scaling works by having difficulty for greedy clients scale up exponentially, and difficulty for well-behaved clients scale down linearly.

This adjustment occurs only at the end of each time interval. The algorithm used for determining puzzle difficulty uses a counting Bloom Filter [2]. This is used because it offers a good trade off between storage space and the probability of assigning high difficulty to client incorrectly. The algorithm for updating difficulty is below:

$$\mathrm{PD}_{i,j+1} = \begin{cases} \mathrm{PD}_{i,j} + & \mathrm{TR}_{i,j} - \mathrm{Decay} \\ \textsf{if} & \mathrm{TR}_{i,j} \leq \mathrm{Decay} \\ & \\ \mathrm{PD}_{i,j} + & 1.01^{\mathrm{TR}_{i,j} - \mathrm{Decay}} \\ \textsf{if} & \mathrm{TR}_{i,j} > \mathrm{Decay} \end{cases}$$

$\mathrm{PD}_{\mathrm{cid},t}$ puzzle difficulty for client i at j^{th} time interval

$\mathrm{TR}_{i,j}$ number of tasks requested from client i at j^{th} time interval

Decay is an application-specific constant that determines the maximum rate of requests for a well-behaved client. Clearly, more machinery is required on the server for this than for a non client-tracking PoW scheme. However, this may be worthwhile if it leads to improved performance under an attack.

5 Analysis of PoW schemes under attack

Four subtypes of PoW scheme were analyzed for resilience to resource-scaling attack. These were fixed-difficulty task-based schemes, two kinds of adaptive task-based schemes, and an interval based scheme. All schemes functioned effectively under attack by non-scaling attackers. So, after the fixed-difficulty scheme, only resilience to resource-scaling attack is discussed. This analysis was done in simulation.

5.1 Simulation Setup

In order to investigate the effectiveness of the various PoW schemes, simulations were performed using the ns-2[18] framework. The architectures above were implemented as ns-2 applications which communicate through high bandwidth UDP links. In order to model puzzle solving without requiring actual CPU cycles, delays were added to the client application to simulate the time required to solve the

puzzles. These delays were determined from the results of the experiments in Section 3.

The simulations support various options for client configuration. Legitimate clients have a fixed rate of delay between requests (0.1 seconds for results in this paper), and request a new task after solving the previous puzzle. Legitimate clients use the CPU to solve all puzzles. Malicious clients request multiple tasks at once in order to maximize their throughput. Experiments were performed with malicious clients using both the CPU (normal attacker) and the GPU (resource scaling attacker) to solve puzzles. Simulations were run with between 5 and 50 legitimate clients, and between 1 and 5 malicious clients. Results are shown for 15 legitimate clients, and 1-5 malicious clients.

The experiments were run with a simple topology using high bandwidth (1 GB) and low latency (3 ms) to prevent lost packets or delays due to slow links. A single server, capable of performing some fairly small number of parallel tasks (10 for all results shown, experiments were performed with between 1 and 100) each lasting no more than 1/10th of a second was connected to a primary router. Clients were connected to secondary routers that were then connected to the primary router as this configuration had no effect on results and streamlined changing experiment configuration. As only server resources and client computation time were limiting factors, the alterations made to the topology had no effect on the results of simulation.

5.2 Breaking

GPU based resource inflation rendered servers with both static difficulty and server-load based adaptive PoW schemes ineffective.

Fixed-Difficulty Server:

Attackers with, and without, resource-scaling capabilities flooded a server with requests, maximizing their resource utilization. Puzzles that required 0.05 seconds and .2 seconds to solve were used to examine the effects of attacks on a permissive and a restrictive server. Puzzles less time consuming than this provided no noticeable throttling. More time-consuming puzzles were considered too severe a penalty for legitimate clients when there was no attack.

Without resource-scaling, utilizing 0.05 second puzzles, even with as many as 1/3 of clients malicious, server availability was not significantly impacted. This can be seen on the left side of Figure 3. While a fraction of requests fail to be served, legitimate clients consistently see between 75% and 85% of their task requests granted. Additionally, while clients do commit significant resources, 70% of processor time is still unused, as would be entire additional cores.

This was markedly better than the service available to legitimate clients when the server provided harder puzzles, as can be seen on the right side of Figure 3. Here, the attacker's capabilities were so constrained that it barely exceeded the legitimate client's throughput, and clients had 100% of their tasks granted. Yet the puzzles were so time consuming that legitimate clients still received poorer performance than with a more permissive server. Only 15 tasks per time interval were granted, versus 20-25 with the easier puzzles. Also, legitimate clients are required to dedicate one full processor to solving the puzzles. The difference in throughput between legitimate clients and attackers is due to the fact that legitimate clients wait some fixed time period between task requests, while attackers request a new task immediately upon solving a puzzle.

With both the easier and the harder puzzles, the best efforts of a limited number of attackers failed to deny service to legitimate clients. This is the intent of PoW schemes and the results are not surprising.

One resource-scaling attacker, though, effectively denied all service to an almost unbounded number of legitimate clients. The computational disparity was so great that, unless the server is assumed to be unrealistically powerful, legitimate clients simply cannot receive service. The legitimate clients continue to solve around 25 (15 respectively) puzzles per time interval, but have 0, or very occasionally 1, task request granted in that same time. This is a clear illustration of the power available to an attacker inflating his resources by utilizing a GPU for hash-reversal.

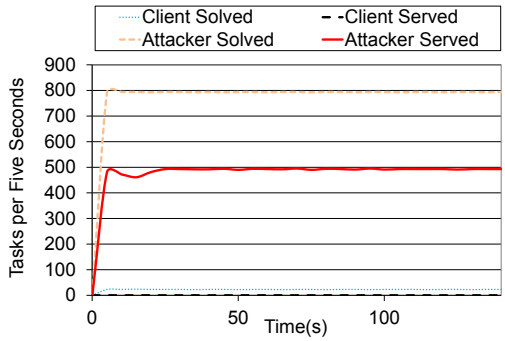

Figure 4: Fixed-Difficulty Server, Resource-Scaling Attacker.

In both the case of the easy and the difficult puzzle, resource-scaling attackers solved more than 750 puzzles in a 5 second period and presented more valid requests than a server can process. Here, the attacker requested tasks at a rate roughly 1.5x the server's work capacity. Clients received no service, as can be seen in Figure 4. Resource-scaling rendered this type of PoW scheme completely useless in the presence of even a tiny number of attackers in our scenarios.

Load-Based Adaptive Server:

The most frequent response to the threat of a resource scaling attack is to "use an adaptive server." Most techniques previously proposed for adaptation modify puzzle difficulty based upon server load, so these were examined first.

A server with load-based difficulty scaling is ineffective when an attacker utilizes GPU scaling, as shown in figure 5. A single resource-scaling attacker denied service to an effectively unbounded number of legitimate clients. With just one attacker, and 14 legitimate clients, legitimate clients

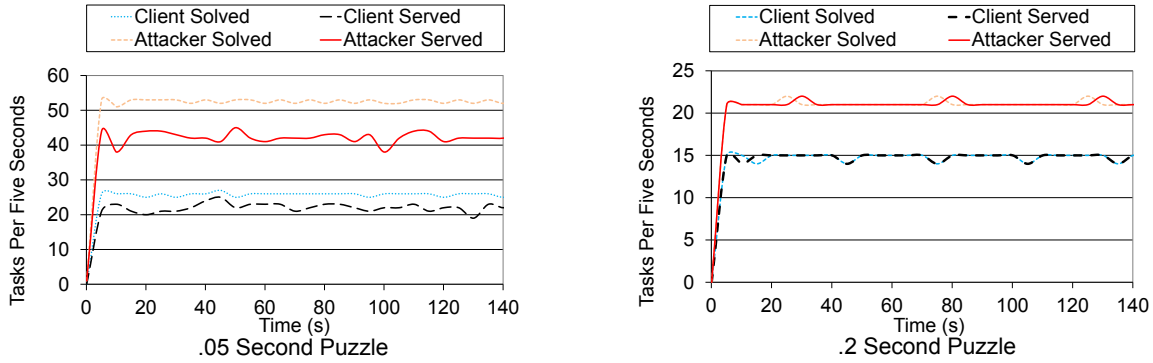

Figure 3: Fixed-Difficulty Server, Non-Scaling Attacker.

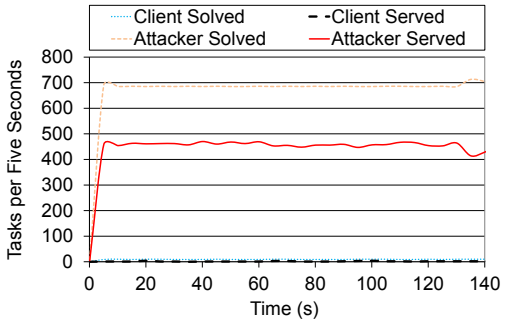

Figure 5: Load-Scaling Server, Resource-Scaling Attacker.

are able to solve around 10 puzzles per time interval and are granted roughly 15% of task requests. More sophisticated load-based adaptation techniques end up being even worse. As [30]'s scheme allowed more difficult solutions to displace tasks previously accepted, an attacker can simply provide only puzzles known to be too difficult for a legitimate client, completely denying all service, and leaving a well-behaved client solving extremely difficult and time consuming puzzles.

These clients are also dedicating 100% of a processor's cycles to solving these puzzles. In a resource-scaling attack against a fixed-difficulty server, legitimate clients received near-to-no service, but at least performed minimal work. In a resource-scaling attack against a load-based adaptive server, a legitimate client received near-to-no service, and dedicated all clock cycles for it. In short, in the presence of resource-scaling attackers, a load-based adaptive server such as those presented in [30, 26] provides no effective guarantee of service, but create a new vector for DoS attacks against their clients.

Having seen resource-scaling attackers render fixed-difficulty servers useless, and turn load-based adaptive servers into a new vector for DoS attacks, it is apparent that alternate techniques are needed. Intuitively, the issue with schemes seen so far has been a failure to differentiate between greedy (or cheating) clients, and regular clients. Techniques that adapt based on client behavior are considered next.

5.3 Rebuilding

Resource Inflation attacks were largely ineffective against servers with any form of client-behavior tracking adaptive server. Task-based servers of this type were relatively effective, and kaPoW servers were very effective.

Client-Behavior Tracking Adaptive Server:

Tracking the behavior of each client provides a means of identifying those clients responsible for disproportionate load. Resource-scaling attackers are capable of solving stunning quantities of puzzles, and generating massive load. Tracking should correctly identify, and penalize, these clients.

Fortunately, client tracking quite quickly rendered a single attacker ineffectual, seen in Figure 6 (left). Even with a third of all clients being resource-scaling attackers, this server provided reasonable service to legitimate clients, as seen in Figure 6 (right). At first attackers dominate the server and deny service, but in a matter of seconds, the server scaled puzzle difficulty adequately to constrain a nearly arbitrarily powerful attacker to imposing a manageable load.

The graphs are a bit difficult to read due to the large oscillation in attacker behavior, but with one resource-scaling attacker, while the attacker went through phases of being able to request a large number of tasks, after an initial spike it never requested more than around 135 tasks per time interval, versus around 800 in earlier simulations with non behavior-tracking servers. This was enough load to slightly interfere with legitimate clients, but they received between 85% and 95% of service requests granted. Performance was significantly worse with five resource-scaling attackers, but was still better than without client behavior tracking. Legitimate clients receive between 50% and 80% of service requests granted. In both of these cases, legitimate clients dedicate very few processor cycles to puzzle solving as puzzle difficulty is based only on past behavior.

As shown, a client tracking PoW server utilizing a far from optimal algorithm for client tracking was sufficient to effectively control resource-scaling attackers. Given

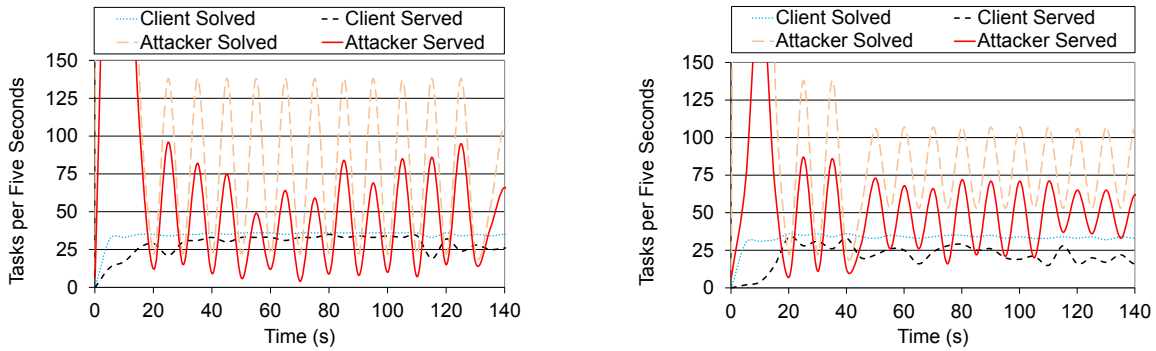

Figure 6: Client-Behavior Tracking Server, Resource-Scaling Attacker.

the ease of constructing hash-reversal puzzles, and the computational asymmetry they permit, it is heartening to see a framework that makes these puzzles viable. The kaPoW system is a system designed, from the start, to utilize client tracking.

kaPoW Server:

As with the client-tracking task-based PoW scheme above, kaPoW maintained effective service to legitimate clients even when a third of all clients were resource-scaling attackers.

In simulation, initially the server resources were consumed by the attacker and few tasks from normal clients were completed. The puzzle difficulty for the attacker increased dramatically. Within a few time intervals, attackers take longer than one time interval to solve a puzzle. For as many intervals as the attacker's puzzle remained too difficult to solve, server resources were entirely allocated to legitimate clients (see Figure 7).

When an attacker requested no tasks due to puzzle difficulty, its puzzle difficulty was decreased, which eventually allowed it to begin flooding the server with task requests again. Since difficulty is scaled up exponentially, and down linearly, this was not immediate, though it was regular.

This cycle repeated, with the attacker alternately locked out of the system and able to lock down the server. On average, the mod_kaPoW simulation managed to control attackers effectively and maintain a reasonable quality of service for legitimate clients when there are resource-scaling attackers. In fact, legitimate clients have an average of 80% of their requests granted, and solve one trivially easy puzzle every ten seconds. In terms of quality of service, this is a significant improvement from even the task-based behavior-tracking server above. Additionally, the server can create fewer puzzles and legitimate clients need to provide fewer solutions than in the situation above.

6 Results and Conclusions

Proof of Work schemes have been proposed, and extensively examined, as countermeasures to denial of service attacks. These methods generally assume there is an upper threshold to the computational disparity between an attacker and a legitimate client. We determined the extent of an attacker's ability to utilize graphics processors in order to gain an advantage far above that previously considered.

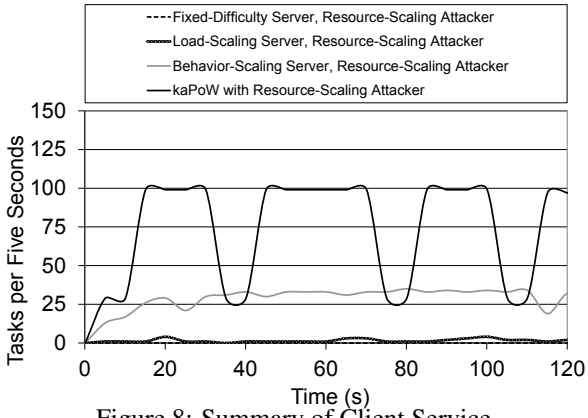

Figure 8: Summary of Client Service.

Knowing that an attacker can gain this advantage, we used simulation to investigate the adversary's impact on Hash-Reversal PoW schemes. The results, displaying the clients' level of service, are shown in Figure 8. An adversary utilizing GPU scaling rendered fixed difficulty server and load scaling server Hash Reversal PoW schemes useless by almost completely preventing service to legitimate clients. We examined a novel behavior scaling scheme which effectively limited an attacker's capability, allowing legitimate clients to receive roughly 60%of ideal service. Finally, we examined the kaPoW architecture and found that clients get an average of 75% of ideal service. With the kaPoW architecture, legitimate clients were also required to perform far less work than was required for the behavior-tracking task based server. In short, schemes which track client behavior and adapt puzzle difficulty proved useful at defeating resource scaling attackers, even those using sophisticated GPGPU methods.

In conclusion, Hash-Reversal PoW schemes can effectively restrict a resource scaling adversary's capabilities by adjusting puzzle difficulty based on past client behavior. Given the harm caused to many existing schemes by the emerging threat of GPGPU based attacks, it is comforting to find a means of continuing to use the most widely examined PoW technique. Also, by tracking client behav-

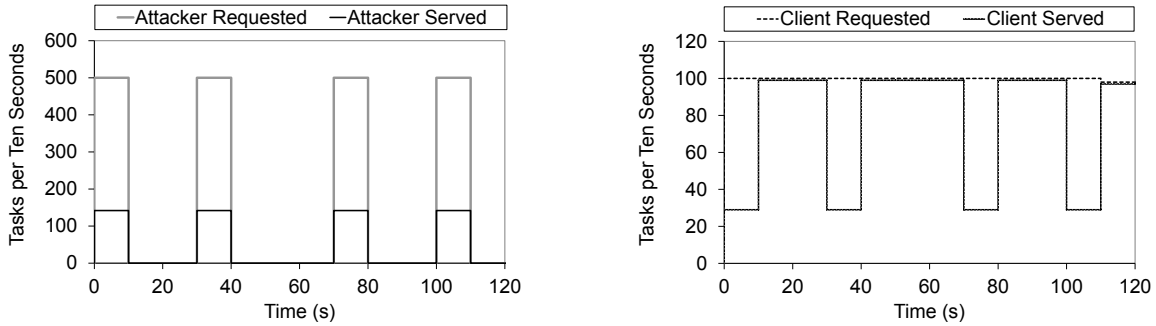

Figure 7: kaPoW with Resource-Scaling Attacker.

ior, resilience to coordinated attacks on larger networks is greatly improved. Interval-based PoW schemes such as kaPoW [14] were more effective than any task-based schemes, providing better quality of service and requiring less work from both legitimate clients and servers.

Acknowledgements The authors would like to thank their shepherd, Guofei Gu, and the reviewers for their comments. This work was supported in part by HHS 90TR0003-01, NSF CNS 09-64392, NASA 09-VVFCS1-09-0010, NSF CNS 09-17218, NSF CNS 07-16626, NSF CNS 07-16421, and grants from the MacArthur Foundation, Boeing, and Lockheed Martin. The views expressed are those of the authors only

References

[1] Bitcoin. http://www.bitcoin.org/.

[2] Bloom Filter. http://en.wikipedia.org/wiki/Bloom_filter.

[3] Folding@Home. http://folding.stanford.edu/.

[4] Nvidia Computational Chemistry. http://www.nvidia.com/object/computational_chemistry.html.

[5] Two root servers targeted by botnet. *PC Advisor (pcadvisor.co.uk)* (02/07/2007).

[6] Phish fighters floored by DDoS assault. *The Register (theregister.co.uk)* (02/20/2007).

[7] Surge in hijacked PC networks. *BBC (bbc.co.uk)* (03/19/2007).

[8] ABADI, M., BURROWS, M., MANASSE, M., AND WOBBER, T. Moderately hard, memory-bound functions. *ACM Trans. Inter. Tech. 5*, 2 (2005), 299–327.

[9] AURA, T., NIKANDER, P., AND LEIWO, J. Dos-resistant authentication with client puzzles. In *8th Inernational Workshop on Security Protocols* (2000).

[10] BACK, A. Hashcash - a denial of service counter-measure. Tech. rep., 2002.

[11] BEVAND, M. Md5 chosen-prefix collisions on gpus. *Black Hat USA* (2009).

[12] DEAN, D., AND STUBBLEFIELD, A. Using client puzzles to protect TLS. In *Proceedings of the 10th USENIX Security Symposium* (August 2001).

[13] DWORK, C., GOLDBERG, A., AND NAOR, M. On memory-bound functions for fighting spam. *CRYPTO* (2003).

[14] E. KAISER, W. F. mod_kapow: Mitigating dos with transparent proof-of-work. In *Proceedings of the 2007 ACM CoNEXT Conference* (New York, New York, December 2007).

[15] FENG, W., FENG, W., AND LUU, A. The design and implementation of network puzzles. In *In Proc. Annual Joint Conf. of IEEE Computer and Communications Societies (INFOCOM)* (2005), pp. 2372–2382.

[16] FENG, W., KAISER, E., AND LUU, A. The design and implementation of network puzzles. In *IEEE INFOCOM* (March 2005).

[17] FENG, W.-C., AND KAISER, E. The case for public work. In *in Proceedings of Global Internet 2007* (May 2007).

[18] INSTITUTE, I. S. The network simulator - ns-2.

[19] JAKOBSON, M., AND JUELS, A. Proofs of work and bread pudding protocols. *Communictions and Multimedia Security* (1999), 258–272.

[20] JUELS, A., AND BRAINARD, J. G. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA* (1999).

[21] KAISER, E., AND CHANG FENG, W. Helping ticketmaster: Changing the economics of ticket robots with geographic proof-of-work. In *Global Internet* (2010).

[22] KANDULA, S., KATABI, D., JACOB, M., AND BERGER, A. W. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *2nd Symposium on Networked Systems Design and Implementation (NSDI)* (Boston, MA, May 2005).

[23] KHANNA, S., VENKATESH, S. S., FATEMIEH, O., KHAN, F., AND GUNTER, C. A. Adaptive selective verification. In *INFOCOM '08: IEEE Conference on Computer Communications* (Phoenix, AZ, April 2008), IEEE.

[24] LAURIE, B., AND CLAYTON, R. "proof-of-work" proves not to work. In *WEAS 04* (2004).

[25] LIU, D., AND CAMP, L. Proof of work can work. In *Fifth Workshop on the Economics of Information Security* (June 2006).

[26] MANKINS, D., KRISHNAN, R., BOYD, C., ZAO, J., AND FRENTZ, M. Mitigating distributed denial of service attacks with dynamic resource pricing. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference* (Washington, DC, USA, 2001), IEEE Computer Society, p. 411.

[27] PARNO, B., WENDLANDT, D., SHI, E., PERRIG, A., MAGGS, B., AND HU, Y.-C. Portcullis: protecting connection setup from denial-of-capability attacks. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2007), ACM, pp. 289–300.

[28] RFC 3174 - US Secure Hash Algorithm 1 (SHA1). http://www.faqs.org/rfcs/rfc3174.html.

[29] SHANKESI, R., FATEMIEH, O., AND GUNTER, C. A. Resource inflation threats to denial of service countermeasures. UIUC Tech. Report, `http://hdl.handle.net/2142/17372`, Oct. 2010.

[30] WANG, X., AND REITER, M. K. Defending against denial-of-service attacks with puzzle auctions. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2003), IEEE Computer Society, p. 78.

[31] WATERS, B., JUELS, A., HALDERMAN, J. A., AND FELTEN, E. New client puzzle outsourcing techniques for dos resistance. *in Proceedings of Computer and Communications Security* (2004).