Resource Inflation Threats to Denial of Service Countermeasures

Ravinder Shankesi, Omid Fatemieh, and Carl A. Gunter,
University of Illinois Urbana-Champaign

Abstract

Currency-based mechanisms have been proposed as a way to use *resource fairness* among contenders for a service to thwart Denial of Service (DoS) attacks. Under resource fairness, a server allocates its service to the clients in proportion to their payment of a resource, making the resource serve as a kind of currency. We consider the vulnerability of currency-based DoS defense mechanisms to various *resource inflation* attacks in which an attacker can substantially inflate its possession of the resource at low cost and in a way that may be either difficult or undesirable for a valid client to do. We provide a simple theoretical analysis of resource inflation attacks and investigate its application to a number of payment schemes to rank their likely vulnerability. We find that the threat of Graphics Processing Units (GPUs) for inflation attacks is especially severe: we are able to demonstrate inflation of up to 630x with common inexpensive GPUs. We also review threats from other capabilities, including multi-core processors, cloud computing, and bandwidth inflation schemes.

1 Introduction

Denial of Service (DoS) attacks on the Internet aim to prevent legitimate clients from accessing a service and are considered a serious threat to the availability and reliability of the Internet services [4, 7, 8, 6, 5]. DoS attacks vary significantly in many aspects, including, (1) the target layer (and protocol) in the network stack, (2) the distribution of attack sources, (3) the specific strategy employed, and (4) their impact. Despite this significant variety, there exists a common objective amongst almost all types of DoS attacks; the attackers aim to exhaust scarce resources, including CPU cycles, memory or disk space, and bandwidth, by generating too many requests. Such an attack is feasible because the attacker often pays very little for requesting a service, most of the time only the cost of sending a network packet.

Numerous DoS defense mechanisms have been proposed in the literature. Among them, we particularly focus on *currency-based* mechanisms. In these defense mechanisms a server under attack demands some type of payment from all clients in order to raise the bar for provoking work by the server. Classic currency examples in this context are *money* [33], *CPU cycles*, also known as puzzle-based schemes [41, 9, 20, 35], and *bandwidth* [24, 27, 40]. Puzzle-based mechanisms have been suggested as a defense mechanism in a broad range of contexts. For instance, *Hash Cashes* [14, 20] are puzzle-based mechanisms that aim to throttle the attacker from sending too much spam. In another context, the authors of [17] suggest using computational puzzles as a defense mechanism against Sybil attacks in peer-to-peer systems. Computational client-puzzles are also suggested as a way of protecting certain protocols such as SSL [18] and ZRTP [25] from DoS .

The broad design objective of the currency-based defense mechanisms is to achieve *resource fairness* [40, 35]. The idea is to provide access to the service proportional to the resource payment by each client. However, a major challenge with resource fairness is that valid clients will among themselves display a measure of *disparity* in their possession of a resource. As attacks demand increasing resource allocation to get service, the poorer valid clients may experience unacceptable resource demands to obtain service and the currency mechanism will not accomplish its goal. Disparity is inevitable, but it has been argued that it is sufficiently modest for proposed resources that currency-based schemes can be effective if they adapt to attackers dynamically. For example, [35] figures that there is a disparity factor of about 40x between high and low-end systems for computational cycles and that this is low enough to make a cycle resource currency usable as a DoS counter-measure. During an attack, the poorer clients would experience longer latencies as they take more time to solve puzzles, but this delay would be significantly lower than the delay that would have occurred had the DoS attack been allowed to proceed without payments to slow the attackers too.

In this paper we argue that such claims need to be analyzed in light of a threat that adversaries may have a way to achieve *resource inflation*, where they use resources or techniques that valid clients may not have implemented. If this inflation technique is good enough, then the arguments that disparity is acceptable begin to break down, as increasingly large numbers of valid clients are unable to muster the resources needed to obtain service. Of course, in many cases it is sufficient for the valid clients just to use the inflation technique themselves, thus leveling the playing field. However, in other cases this may not be possible or desirable. For instance, for bandwidth-based schemes, a client may increase their bandwidth resources by either deliberately misusing the wireless MAC layer, or ignoring congestion control back-off rules at the transport layer. This enables them to gain extra bandwidth at the expense of other hosts. Such modifications may require special administrative privileges or access to specialized software and can create DoS or congestion in the network so valid clients are not likely to use them. In such cases, it is hoped that the inflation threat is not too severe or that other measures, like detection of such behavior, can limit its effectiveness. In other cases, the issue may be more borderline and it may be debatable whether valid clients should use the same inflation tactics as attackers. In all three cases, however, it is prudent to have some way of analyzing the threat to determine whether the valid nodes should *match* the inflation technique, *ignore* it because its impact is small, or *change* to different currency-based schemes.

We consider a range of resource inflation strategies and attempt to access their likely effectiveness. These include: cloud computing (outsourcing of puzzles), multi-core parallel computations (enabled by many new processors), the use of Graphical Processing Units (GPUs), and the bandwidth inflation techniques mentioned above. Among these we find that GPUs pose the greatest threat and require the most immediate attention. We show that attackers can use cheap and widely available GPUs to inflate their ability to solve typical cycle exhaustion puzzles by more than a 600x factor. While adaptive puzzle schemes may be able to tolerate a disparity of 40x, they fail when faced with a 600x disparity. In this range, adaptively making the resource demands higher does not help, since adversaries are so much richer than valid clients that the latter are ‘priced out of the market’ and fail to get service in a reasonable time. Reasonable people could disagree on whether valid clients could or should even the score by using GPUs themselves to solve puzzles. Indeed, the software we demonstrate for attackers can itself be used by valid clients. On the other hand, we consider some of the drawbacks valid clients might experience if they use their GPUs to solve DoS puzzles.

The paper makes three primary contributions. First, we introduce and analyze the concept of resource inflation as a ‘thinking out of the box’ approach to defeating DoS countermeasures. This includes a simple quantitative model to assess the impact of resource inflation. We argue that currency-based schemes should always be analyzed with this model to estimate their resilience against resource inflation. Second, we illustrate a series of resource inflation attacks on existing DoS defense mechanisms to underscore the significance and implications of this threat. Third, while the general idea of using GPUs to solve puzzles may seem obvious to some, how to implement and measure a full range of strategies is not obvious and comprised the bulk of the work we needed to do for this paper.

The rest of this paper is organized as follows. Section 2 overviews a number of existing currency defense mechanisms, as well as the resource fairness concept. Section 3 provides a simple analysis of resource inflation attacks. Section 4 overviews the GPU devices on computers and reports on our experiments aimed at enhancing puzzle solving speeds using GPUs. Section 5 considers other avenues for performing resource inflation attacks. Section 6 raises discussion points, concludes the paper, and identifies areas of future work.

2 Background and Related Work

In currency-based DoS defense mechanisms a server under attack demands some type of payment from all clients in order to raise the bar for provoking work by the server. In this section we explain some of the existing work on two classes of currency-based mechanisms: puzzle-based and bandwidth-based. We also explain resource fairness as a goal that currency mechanisms aim to achieve.

2.1 Puzzle-Based Mechanisms

Puzzle-based defense mechanisms such as [41, 9, 20, 35, 26] try to correct the imbalance between the cost to the attacker for generating a request and cost to the server for processing a request by demanding a payment, in the form of a puzzle solution, from each client. In a typical puzzle-based scheme, a request must be accompanied by a proof of payment from the client. The payment may be in the form of computation or memory accesses that the client needs to perform to solve the puzzle. Since the amount of resources available to the attacker is limited (even if it is much more than that of the legitimate clients), the attacker will not be able to trivially amplify his attack. There are different kinds of schemes that build on this general principle. Laurie *et al.* [31] have analyzed proof-of-work schemes in the context of a spam deterrent mechanism and conclude that proof-of-work schemes do not work, because the cost involved for legitimate senders would be too high. However, their economic estimation contains a miscalculation. More importantly, their analysis considers fixed-cost puzzle schemes and does not analyze adaptive proof-of-work schemes proposed by recent DoS counter-measures.

2.1.1 Hash-Reversal based Puzzle Schemes

Hash-reversal-based puzzle schemes are among the most popular classes of puzzle schemes suggested in various DoS defense mechanisms. In such a scheme before a client is given access to a resource, the server (or a third-party) presents a puzzle-seed s to the client. The client then needs to find the puzzle solution x and compute 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 (or a participating router) verifies that the last l bits of p are zeros (and also that the client has used a valid seed and that it is not a duplicate). The value l determines the *difficulty level* of the puzzle. If the hash-algorithm H used is pre-image resistant, the client has no better algorithm than trying to guess x randomly. Therefore, 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 have a more fine-grained level of the work required by the client, for instance, by using multiple sub-puzzles or by giving the user a hint as to the starting value to guess [21].

A very useful characteristic of a hash-reversal-based scheme 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. As noted in [35] a typical PC can perform 2^{20} hashes per second. However, a concern with this scheme is that there is a scope for disparity amongst clients with differing computing powers (up to 38x).

Another concern is that the puzzle solving can be parallelized. However, since the attacker is going to solve the puzzle using a fixed (albeit large) amount of resources, this has generally not been considered a big problem.

2.1.2 Time-Lock Puzzle Schemes

Time-lock puzzles [38] were originally proposed to preserve information in a time capsule. The original goal was to ensure that a client cannot decrypt a given message until a given period of time into the future has elapsed. To ensure that a client cannot simply throw more computing resources to solve the problem in a shorter period of time, the puzzles were designed to be non-parallelizable. In the original construction, for a random a and difficulty l, the server asks the client to compute:

$$b = a^{2^l}(mod\, n)$$

where n is the usual RSA modulus (the product of p and q, two distinct large prime numbers). If the client does not know the factorization of n, then the only known way to compute b efficiently is by l modular squarings of a. Because the results of the next step in modularization depends on the output of the previous step, it is not possible to parallelize this operation.

The server can efficiently verify the same value by computing:

$$b = a^e(mod\, n)$$

where $e = 2^l(mod\, \phi(n))$ and $\phi(n) = (p-1)*(q-1)$ is the usual totient function. Both $\phi(n)$ and e can be easily computed by the server in possession of factorization of n. Due to the non-parallelizability property, this has been suggested as a puzzle-based scheme for DoS defense [42].

2.1.3 Memory-Bound Puzzle schemes

Noting the high disparity in computation between different clients, researchers have proposed using memory latency as a mechanism for resource payment [20, 9, 19]. Because the memory latency for accessing a single word from the main memory exhibits low disparity (5-10x) across different classes of machines, it is arguably a fairer puzzle mechanism.

In such schemes, the client solves the puzzle by successively looking up values from a pre-computed table in the main-memory. Care must be taken to ensure that the table is not too small as it may completely fit in the cache of a higher-end computer. It may also not be too large as it may be more than the memory available to a lower-end device. It is also important that the memory access pattern (while solving the puzzle) is fairly random, otherwise it may lead to higher cache hits.

Another concern is that the disparity between the server (in creating the puzzle) and the client (in solving it) is not as large as in the hash-reversal-based schemes. One way of hardening the puzzle is, for instance, by having the client solve multiple memory-bound puzzles for every single request [19].

2.2 Bandwidth-Based Mechanisms

In a bandwidth-based currency scheme clients use additional bandwidth to get access. It is often assumed that attackers are using all of the bandwidth available to them (or the maximum bandwidth they can afford to use without being detected by other mechanisms) to execute an attack, whereas legitimate clients are using only the resources they require to accomplish their less-demanding objectives. Hence legitimate clients have bandwidth to spare and can use this fact to reduce the attacker's chances of success. This strategy was introduced in [24] in the context of authenticated broadcast and extended to general Internet protocols in [40, 27].

Two general strategies have been explored by researchers in this domain. In selective verification [24, 39], clients send extra requests and the server selects from them probabilistically. The extra requests serve as a form of bandwidth payment that could adaptively change according to the severity of the attack [27]. Bandwidth auctions allow clients to build credit by sending bytes to an auction system from which the server periodically takes requests from clients that have built the most credit (in terms of bandwidth payment) [40].

2.3 Resource Fairness

A broad design objective of many of the currency-based defense mechanisms is to achieve *Resource Fairness* [40, 35]. The idea is to provide access to the service proportional to the resource payment by the client. If r_i is the amount of the resource payment by each individual client i (for instance, computation to solve a puzzle), given total server (processing) bandwidth s, the client i would be entitled to the following allocation of server's bandwidth:

$$a_i = \frac{r_i}{\Sigma_j r_j} s$$

Under such schemes, to achieve fairness in server resource allocation, all the clients must have little disparity in terms of the actual resource (currency) they own and are willing to spend. However, in general, different clients might have different resources available. For instance, the authors in [35] note that there is a disparity of 38x in terms of the time required to solve the computational puzzle scheme they use (comparing a Nokia 6620 cellphone to a desktop PC with a Hyper-Threaded Intel Xeon 3.20GHz processor). Since, the differences between memory latencies are much smaller (5x-10x), various memory bound puzzle schemes [9, 19] are proposed to make the allocation fairer.

2.4 Adaptation

It is desirable for the currency-based mechanisms to *adaptively* require payment from clients proportional to the severity of the attack, *i.e.* the demand for server's resources. The payment must be high enough to guarantee service for a client, while not too high to incur unnecessary cost. The cost can be both on clients and the server. Examples of cost on the client are solving computationally expensive puzzles or sending significant amounts of dummy bytes as bandwidth payment. An example of cost on the server is allocating a high-bandwidth channel to receive bandwidth payments. Among the mechanisms proposed for achieving adaptation, the following general paradigms stand out: (1) *auctions* by the server (2) *probabilistic selective processing* at the server and (3) *ramp-up* at the client. Many of the existing adaptive mechanisms are based on (a variation of) the above paradigms. Below we provide illustrative examples of such mechanisms.

In the context of puzzle-based schemes Wang *et al.* [41] provide an auction-based framework in which the server maintains a buffer containing client requests, which it processes periodically. Client i's request r_i contains a puzzle solution of difficulty *DIF*(r_i). As the server receives requests from clients, the buffer may become full. In this case, upon arrival of a new request r_j, if *DIF*(r_j) is greater than the lowest difficulty puzzle in the queue r_k, then r_k is replaced with r_j. Each client i has a *valuation* v_i for getting service, which roughly translates to the maximum number of hash operations it is willing to perform to solve puzzles to get service. A client initially starts its requests with no solution enclosed (*DIF*$(r_i) = 0$). If it does not get service, it increments the difficulty of the puzzles solution enclosed in the next request by *INCR*. This process is repeated until client gets service, or the required difficulty increases beyond the client's valuation. *INCR* is determined by v_i and the maximum latency the client is willing to tolerate. The authors show that this mechanisms is efficient in the sense that the client can raise its bid just above the attacker's bid to win an auction. Parno *et al.* [35] employ a similar strategy in routers that differs in terms of client ramp-up strategy. At at each step, the clients double the puzzle difficulty if they do not receive service. Further analysis on this scheme is provided in Section 3.3.

In the context of bandwidth-based schemes bandwidth auctions [40] allow clients to build credit by sending dummy bytes to an accounting system and the server periodically holds auctions to take requests from clients that have built the most credit. No particular strategy for governing the pace of bandwidth payments by clients is disclosed, except it is required to be in a congestion-controlled channel. The Adaptive Selective Verification bandwidth scheme [27] requires clients to respond to an attack by using time-out windows and "drop acknowledgements" from the server to adaptively boost request rates. The server selectively processes some of the requests, and sends drop acknowledgements for each request it denies. Drop acknowledgements work as a "please send more" message to the clients. The request rate by a client is doubled upon each failed attempt, up to a pre-determined maximum level. The authors show the solution is efficient in terms of bandwidth usage.

3 Resource Inflation Analysis

Most systems are designed with an approximate or exact expectation as to how its users would interact with them. Such an assumption could particularly be dangerous if the users can gain significant advantages by taking actions outside of the presumed behavior model. In this section we discuss how 'thinking out of the box' by some users of a system can result in unexpected surprises. In particular, we focus on cases where a limited set of users are willing to engage in such an activity, whereas a large number of ordinary users may not have the resources or permissions to do so. We mention examples where this can result in security breaches and focus on the possibility of resource inflation in currency-based DoS countermeasures. We provide a simple analysis of resource inflation in these mechanisms and specifically enumerate feasible resource inflation attacks on puzzle-based and bandwidth-based mechanisms.

3.1 Thinking out of the Box: Resource Inflation

There exist many examples in today's systems where a user can use out-of-the-box solutions to either gain a significant advantage over other users, or violate fundamental safety properties of a system. A prime example is the differential power analysis (DPA) attack on smart cards [28]. In a DPA attack, the attacker collects power consumption measurements from a tamper resistent device (*e.g.* a smart card) to find secret keys embedded in the device. This kind of attack was either unknown, or disregarded as a serious threat at the design time. It is also not easy to execute for the majority of non-sophisticated users of smart cards. Nevertheless, it has been shown that this threat is serious and needs to be addressed. Another example is greedy MAC-layer behavior in wireless networks. A greedy host can deliberately misuse the MAC protocol to gain significant extra bandwidth at the expense of other stations [30, 37, 36]. Again, despite such behavior is not feasible or desirable for the majority of ordinary wireless users, its significance and consequences cannot be overlooked.

In the context of currency-based DoS countermeasures, a goal for out-of-the-box solutions would be to create large disparities in terms of access to the resource that serves as payment. While different defense mechanisms take natural disparities in resource ownership between clients into account (usually by trading in latency to get the service allocated), the problem would be exacerbated if the attacker can find practical ways of inflating its resource payment using out-of-the-box solutions. This helps an attacker build an advantage in terms of payments, and consequently the share of service she secures for herself. Such attempts would be particularly effective if repeating the same strategy is impractical or undesirable for legitimate clients.

Botnets. Given the widespread existence of botnets, one may consider the analysis of resource inflation threats for DoS countermeasures unnecessary. We make the following argument to reject this perception. Almost all DoS countermeasures have some kind of 'breakdown point,' which is the amount of resources the attackers need to have in order to break it. Compromising remote computers to form bots is not free; it entails certain risks/costs to the attackers. Renting bots from botnet owners is not free either. Now consider the following examples. First, assume the number of bots required to completely break a particular DoS countermeasure on a server to be 60,000. If the attacker can use resource inflation to mount the same attack using only 100 nodes, it in fact faces a 'lower bar' in order to launch the desired attack. This effectively translates to higher probability or frequency of attacks in the long run. Second, imagine the largest botnet in the world to comprise 1 million machines. If the botnets owners can use resource inflation to effectively make it have the power of a 600 million botnet, then this would be a game changer in terms of what can be done (if at all) to protect any internet connected resource. So, we think regardless of the size and ease of access to botnets, it is prudent to fully consider the ramification of resource inflation attacks.

3.2 General Analysis

Consider a server S that can process requests at the rate of s requests per time unit. The legitimate clients of this server send requests at the average rate of q_l requests per time unit. The server is over-provisioned, so $s > q_l$. Attackers send requests at an aggregate rate of q_a per time unit in such a way that the server is overwhelmed, *i.e.* $q_l + q_a > q_a \gg s$. Assume that the server deploys a resource fairness scheme in which it allocates its processing bandwidth according to the corresponding resource payment by each client. In this scheme, if r_i is the amount of the resource payment by each individual client i (for instance, computation to solve a puzzle), the client i should ideally be entitled to the following allocation:

$$a_i = \min(q_i, \frac{r_i}{\Sigma_j r_j} s)$$

Therefore, if we represent the aggregate resource (payment) of legitimate clients and attackers (per time unit) by r_l and r_a, the legitimate clients get the aggregate allocation of $a_l = \min(q_l, \frac{r_l}{r_l + r_a} s)$. The allocation for the attackers would be $a_a = \min(q_a, \frac{r_a}{r_l + r_a} s)$. Given that $q_a \gg s$, we have $a_a = \frac{r_a}{r_l + r_a} s$.

Suppose the attackers are capable of inflating their resources by an *inflation factor* of α. As a result, the legitimate clients will get the allocation $a_l = \frac{r_l}{r_l + \alpha r_a} s$, and the attackers get the aggregate allocation of $a_a = \frac{\alpha r_a}{r_l + \alpha r_a} s$.

Example. Consider a simplified setting in which a server aims to protect itself against resource exhaustion attacks by asking clients to solve computational puzzles. Assume for simplicity that there is no disparity in computational power among legitimate clients. The attacker controls a botnet consisting of compromised hosts. Each compromised host has equal computational power to a legitimate client. The legitimate and botnet hosts represent 90% and 10% of the clientele, respectively. Under resource fairness, they would be entitled to an aggregate 90% and 10% of the server's processing bandwidth. Suppose the attacker uses the techniques in Section 4 to inflate its resources by a factor of $\alpha = 630$. As a result, the legitimate client's share of the server's processing bandwidth would fall to 1.4% (from 90%), and attackers will get 98.6% of the server's capacity. We also investigate the outcome for $\alpha = 40$ and $\alpha = 1300$, corresponding to resource inflation using cloud computing and high-end GPUs from our experiments in Sections 4 and 5. Table 1 summarizes allocation for different values of α.

Table 1: Resource Inflation Analysis.

Inflation Source	Inflation Factor	Attacker Population	Attacker Allocation
-	0	10%	10%
Cloud Computing	40	10%	81.6%
Stock GPU	630	10%	98.6%
High-end GPU	1300	10%	99.3%
-	0	2%	2%
Cloud Computing	40	2%	47%
Stock GPU	630	2%	92.8%
High-end GPU	1300	2%	96.4%

3.3 Analysis under Adaptive Schemes

As we discussed above, in the case of a naive resource payment mechanism, where the clients and the attacker make a payment commensurate with their available resource in each time-slot, the effect of resource inflation is very detrimental to normal clients. As discussed in Section 2 most puzzle schemes try to account for disparities between client resources by using an adaptive payment mechanism. Under such an adaptive scheme, if the client fails to get a connection in a given time-slot, it re-sends a request with a higher payment (typically twice as much [35, 27]). It takes clients more time to make successive resource payments, thereby increasing the time to get service. This is also referred to as the connection establishment time.

In Portcullis, the authors analyze the connection establishment time for a given client for their adaptive scheme. Under the assumption that all client and attacker nodes have uniform computational resources, the authors characterize the expected time for a client to traverse a bottleneck link as $O(n_m)$, where n_m is the number of malicious hosts. In fact, they show that if the server has no external ways of distinguishing legitimate senders from malicious attackers, then there is always an attacker strategy such that the expected time for a legitimate sender is $\Omega(n_m)$.

The analysis above is done under the assumption of uniform attacker and client nodes. If we have a resource inflation of α at each of the n_m attacker nodes, we can equivalently treat them as if we have $n_m * \alpha$ attacker nodes (of uniform computational resources). Therefore, the expected time for a legitimate client under the previous theorem is of order $O(\alpha n_m)$. For example, if in the presence of a 1000 node attack the connection establishment time is 100ms, with $\alpha = 600$ it may be as high as 60 seconds.

Unlike the assumptions made for the analysis, in general, the clients have non-uniform computational resources. For instance, there is a difference of about 38x when computing SHA-1 hashes between a Nokia 6620 cellphone and a Xeon 3.2GHz based computer [35]. Under those assumptions, we can extend the results regarding expected time for connection establishment to non-uniform clients as follows. If β_i is the ratio of computational resource between each of the n_m attacker hosts and client i, then the expected time for the client to establish a connection is of order $O(n_m \beta_i)$. If there is a resource inflation of α available to each of the attacker hosts, then the expected time is of order $O(n_m \beta_i \alpha)$. Using the previous example, assuming a pathological case in which a legitimate client uses a cell-phone, while attackers use PCs (*e.g.* $\beta_i = 20$), the legitimate client may face connection establishment times as high as 1200 seconds.

3.4 Overview of Resource Inflation Threats to DoS Countermeasures

We perform an extensive experimental case study on resource inflation attempts on puzzle-based mechanisms. In particular, we focus on three avenues. First, we investigate the feasibility of using GPUs for improving puzzle-solving speeds as GPUs are particularly efficient in handling large numbers of similar operations in parallel. Second, we study the feasibility of speeding up puzzle solving by leveraging multiple processors in a multi-core processor. Third, we consider the use of cloud computing facilities for parallelizing, and hence speeding up puzzle solving capabilities.

We also explore resource inflation possibilities against bandwidth-based mechanisms. In particular, we focus on three classes of resource inflation possibilities. First, we study the possibility and implications of attacks at transport layer, and in particular those that ignore or exploit congestion control mechanisms. Second, we consider the effect of greedy MAC-layer behavior by an attacker that tries to gain a larger share of the bandwidth by violating the protocol in subtle ways. Third, we briefly discuss how using more than one interface for network connection can help an attacker inflate its resources.

4 Resource Inflation using GPUs

Graphics Processing Units (GPUs) are dedicated devices used for rendering, manipulating, and displaying computer graphics. Modern GPUs are in general very efficient at processing large amounts of data in parallel. Unlike the modern CPU which is designed to efficiently optimize the execution of single threaded (or not highly multi-threaded) programs using complex out-of-order execution strategies, a modern GPU's efficiency comes from executing massively data-parallel programs. These are algorithms which perform simple operations on a large number of data points in parallel. This is often referred to as Single Instruction Multiple Data (SIMD) programming.

Recently, there has been significant interest in using GPUs' efficiency at executing data parallel algorithms for non graphical computation. This paradigm is known as General Purpose GPU (GPGPU) computing or Stream Computing in the hands of non-graphics programmers. There exist multiple tool-chains that support GPGPU programming. Two of the most popular ones are the Brook/Compute Abstraction Layer (CAL) from AMD [12] and Compute Unified Device Architecture (CUDA) from Nvidia [34]. GPGPUs have been used in improving the speed of various programs such as Folding@Home [1], computational chemistry [3] as well as various other fields. Recently, GPUs were also used for finding MD5 chosen-prefix collisions [16].

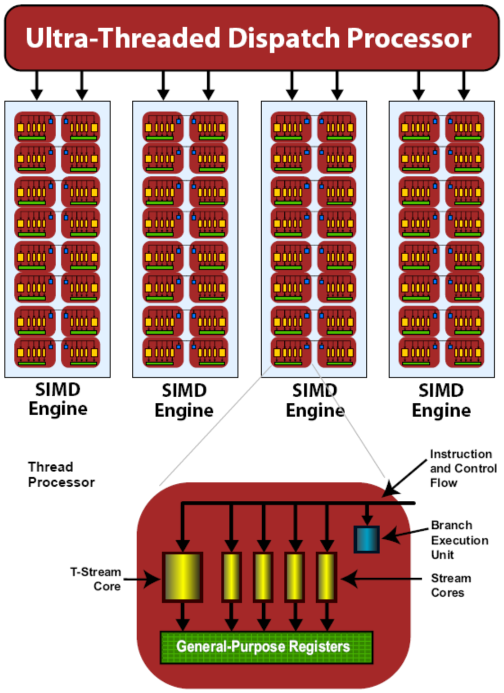


Figure 1: Logical view of GPGPU Architecture (from[13]).

For instance, AMD's GPU-based architecture is logically represented in Figure 1. As shown in the figure, the GPU consists of a large number of SIMD engines. Each engine in turn consists of a number of thread-processors. For instance, AMD's HD4850 contains 800 thread-processor instances. Each thread processor unit has access to its own general purpose registers and can also access the GPU's memory. The thread dispatcher manages various threads and is invoked by the client on the CPU.

It must be noted that threads in GPUs are not as 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 the GPU-based threads (within a single SIMD engine block) need to execute the same instruction (possibly on different input data) for maximum efficiency. In effect, we cannot use different threads to perform completely different 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 big advantages in efficiency. This is both in terms of price and power consumption. For instance, a mid-range card such as the ATI Radeon HD4850 (with 800 stream processors) is under $150 and much cheaper (although lot more inflexible) than buying a large number of processors.

4.1 GPGPU Programming Model

A GPU-based program is usually written in a way to take advantage of the inherently data parallel programs (such as matrix multiplication, simulations, *etc.*). Given a program, the GPU-based platform can run the program such that each thread of the program operate on a single data (or a single block of data). All of such threads can run simultaneously on the stream processors (Figure 1) as long as there are enough stream processors on the GPU card. In case the data elements are larger than the number of stream processors, the Thread Dispatcher manages the available processors between the various 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 main memory of the CPU into the memory of the GPU. Thereafter, 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. The net effect of this program is that the sum of two vectors `a,b` is computed and stored in the output vector `c`. If the size of the stream is much larger than that of the number of stream processors available on the card, this might take a few iterations to complete.

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

4.2 Solving Hash-Reversal Puzzles using GPUs

We take advantage of the inherently data-parallel nature of the hash-reversal puzzles when solving them on the GPU. Hash-reversal puzzles, such as the one described in Section 2.1.1, can be speeded up using GPUs in the following two ways.

In the first approach, we can run each individual guess for the puzzle solution x of the hash puzzle as an independent thread on the GPU. Since there is almost no data-dependent divergence in SHA-1, we will get close to the maximum efficiency when running a single puzzle split-up in multiple threads, and thereby speeding up solving of a single puzzle. The disadvantage with this approach is that there is a reasonable overhead involved in sending data to and from the card, and this overhead is shared by each solving of the puzzle.

In the second approach, given n puzzles with puzzle difficulty level l, we run each puzzle into 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 searches a 2^l search space for the puzzle solution. It is important to note that, because all of the threads are running the same code , the GPU can run all the different threads *simultaneously*. If the hash computation were somehow *data-divergent*, *i.e.* different operations were to be performed on the data based on the values of the data, 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.

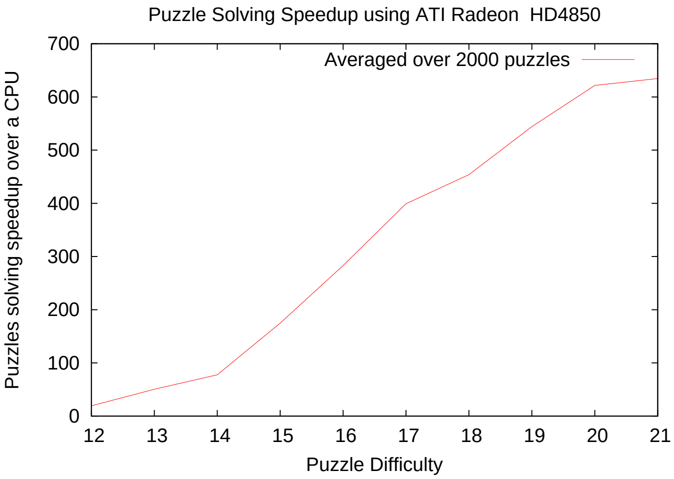


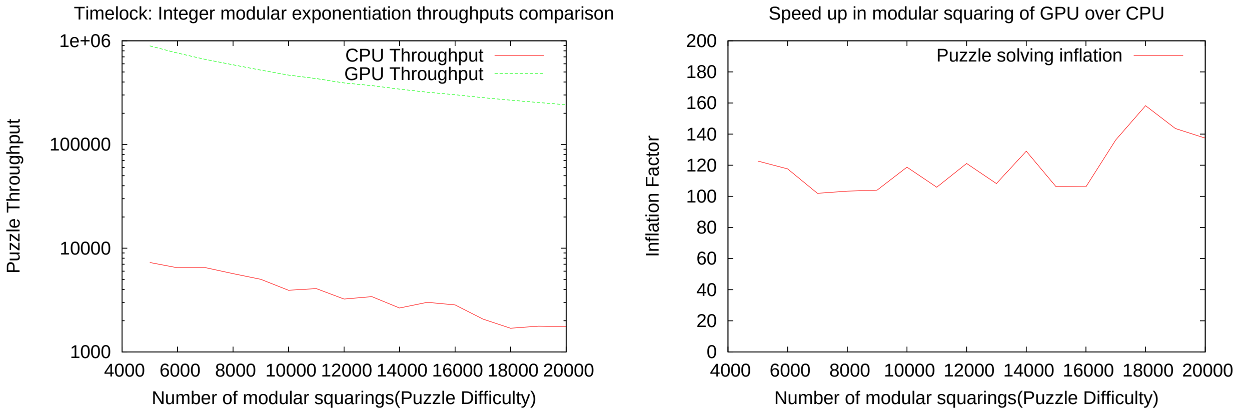
Figure 2: Speed up achieved in througphut of SHA-1-based hash-reversal puzzles on ATI HD 4850.

Analysis: We have measured the speedup achieved in solving n hash-reversal based puzzles on an mid-range ATI Graphics card (ATI HD 4850, costs about $150). The speedup shown is against a PC that solves 2^{20} SHA-1 hashes per second. The results show that an inflation factor of up to 630 could be achieved. We must also emphasize that this is not a high-end graphics card. We also experimented with a high-end graphics card (ATI Radeon HD 4870x2, costs about $450) that has twice the amount of streaming processors (1600 vs. the 800) on the card, and is also clocked slightly higher. Using that card, we achieved the inflation factor of up to 1230 for puzzle difficulty 21.

4.3 Solving Time-lock Puzzles using GPUs

Time-lock based puzzles as are explicitly designed to be non-parallelizeable. In effect, this means that the attacker cannot speed up the solving of any given individual puzzle. However due to the nature of GPU-based computation, the attacker can solve multiple time-lock puzzles at the same time. Although, the attacker will not be able to reduce the time taken to solve any individual puzzle, he can achieve resource inflation by solving a different time-lock puzzle in each individual thread of the GPU stream-computing processor.

Figure 3a compares the results of performing modular exponentiations of 32-bit integers on a Core2 Duo Processor versus a ATI Radeon 4870x2 GPU. The x-axis measures the puzzle difficulty and the y-axis gives the throughput in terms of puzzles solved per second. The results are averaged over 2000 puzzles per run. Note that the y-axis is in log-scale.



(a)

(b)

Figure 3: (a) Time-lock puzzle throughputs between ATI Radeon HD 4870x2 and Core2 Duo 2.4GHz. (b) Time-lock puzzle inflation comparing GPU versus the CPU.

Figure 3b provides the resource inflation achieved by the GPU versus the CPU for the results in Figure 3a. Note that we achieve an inflation factor of up to 158 for some parameters.

4.4 Attacks on Other Puzzle-Based Schemes

Since the hash-based puzzle schemes are inherently parallel in nature, the speed at which they are solved can be significantly improved by effectively solving the sub-puzzles on different computational resources.

Another significant class of puzzle schemes proposed are the memory-bound puzzle schemes ([20, 9, 19]). The key idea under such schemes is that the disparity in the memory latency between different machines is much smaller than the the disparity in terms of the CPU power between different machines.

While this is true for CPUs, we note that the memory latency between the GPU processor and its onboard memory is slightly lower than the latency between the CPU and its memory. Also, a single instance of a memory puzzle does not have enough of a disparity (between the server's puzzle creation and the client's puzzle solving), so usually the puzzle is hardened by having the client solve multiple instances simultaneously. A strategy employed by [19] aims to stop parallelization of multiple individual memory-bound puzzles. However, this hardening scheme stops parallelization only under the assumption that there is no effective way to synchronize between multiple puzzle-solving threads (which is not the case for GPU-based computation). We believe that some of these puzzle schemes need to be analyzed more thoroughly to see if they are amenable to a GPU-based resource inflation attack.

4.5 GPU Utilization by Legitimate Clients

At this juncture, it may be argued that the legitimate clients can also utilize GPUs to achieve parity with an attacker using GPUs. This, however, is not always a feasible counter-measure for a number of reasons.

Firstly, many of the clients may not have a GPU present to solve the puzzles. Even if they have a GPU, they may not be available to the user for general purpose computation. Typically these are the same clients which are amongst the less powerful nodes in the network such as cell phones or older desktops. Furthermore, the disparity in computational power of GPUs is vast as well. By making legitimate clients with GPUs increase their puzzle throughput, we will make it harder for those clients that either do not have access to a GPU or have GPUs with lesser computational power to compete with the rest of the clients.

Secondly, in the puzzle schemes that are resitant to parallelization, increased throughput in the puzzle solving does not help the client get a connection faster. The attacker can utilize the increased puzzle-solving throughput because his goal is to occupy the largest number of available connections. The client may only be interested in getting one of his requests accepted by the server and therefore solving multiple puzzles at a given point may not help his cause.

Furthermore, if we allow the legitimate clients to use GPUs to solve multiple puzzles simultaneously, the server will have to generate and verify a lot more puzzles. Depending on the puzzle-scheme, it will also need to store a lot more of the valid puzzle solutions to prevent replays.

5 Other Resource Inflation Threats

In this section we study other avenues for resource inflation on puzzle-based mechanisms and bandwidth-based mechanisms. On puzzle-based mechanisms, we focus on the use of multi-core processors and cloud computing for resource inflation. On bandwidth-based mechanisms, we focus on the possibility of misbehavior in the MAC layer and transport layer, as well as the possibility of using multiple interfaces by the attacker.

5.1 Hash-reversal Puzzle-based Mechanisms

We describe and analyze two ways by which one can speed-up the solving of hash-reversal puzzle schemes, namely the use of multi-core processors and cloud computing. Although the attacks provided in this section provide a cause for concern, they may not pose threats as serious as those by GPUs (see Section 4).

Multi-Core Processors A simple way to speed up solving the hash-reversal puzzle scheme is to simultaneously utilize the cores in a multi-core processor. As the hash-reversal puzzle scheme is *inherently* parallelizable, the attacker can solve a hash-reversal puzzle of level l by splitting the search into n threads, where n is the number of multiple processors available to the system (recall that this requires the attacker to search the solution space of about 2^l). In effect, the time taken to solve the puzzle is reduced from t on a single threaded implementation to $(t/n) + c$ on an n-core processor, where c is a small synchronization overhead.

In an analysis in [35], a factor of 38x speed-up is provided between a resource constrained Nokia 6620 and a PC with a Xeon 3.2GHz processor. However, it appears that the puzzle solving algorithm on the Xeon processor was not parallelized. We study the benefits of running multiple threads for solving hash-reversal puzzles by implementing it on a quad-core Intel Q6600 processor. Figure 4 summarizes our results. Given the recent proliferation of multi-core processors, an attacker controlling a botnet can utilize this to his advantage, causing a small amount of resource inflation.

Cloud Computing Another way for the attacker to achieve resource inflation is by using cloud computing. Various services such as Google's App Engine [23] and Amazon's Elastic Compute Cloud [11] allow users to host their web-service infrastructure on the cloud computing infrastructure. A key feature of these services is that they allow scalable implementations at a very competitive price. The attacker can effectively "hire" a cloud-based computation for the short duration of an attack and have a multiplier effect on the damage that he can cause, compared to what he could have achieved only with his existing botnet.

The attacker can use a cloud-computing based implementation to outsource the puzzle solving, so that an individual bot in a botnet can query the cloud-based puzzle solver to get the solutions without using the limited resources on the bot (Figure 5). In effect, the throttling that the puzzle-scheme is supposed to achieve on the attacker is now no longer as effective.

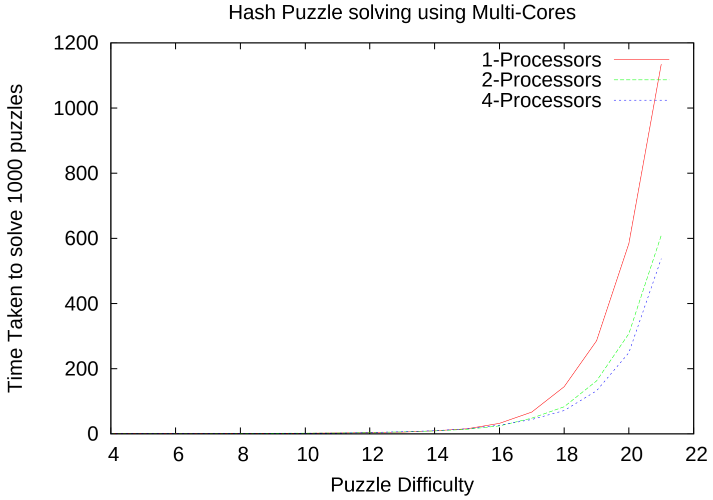


Figure 4: Time taken to solve SHA-1 based hash-reversal puzzles on a quad-core Intel Q6600 processor (summed over 1000 runs).

For this to be effective, however, the attacker must be able implement the cloud-based puzzle solver with low cost. Google's App Engine allows a user to host web-based applications written in Python or Java. In fact, it allows free hosting for applications along with enough bandwidth and CPU resource to allow around 5 million page-views a month. By utilizing this free hosting service, given an account, the attacker can implement the puzzle solving code as a web-based form which returns the results to the puzzle parameters given in the form.

We implemented the hash-reversal based puzzle scheme on Google's App Engine. The performance results are shown in Figure 6a. This test was run with 50 concurrent users (each asking 50 puzzles serially, for a total of 2500 requests per puzzle difficulty). We measured the number of puzzles that we're able to solve per second. As shown in the figure, the performance rate was very good. Figure 6b shows the factor of inflation that the attacker using the cloud would get as compared against a stock PC that can compute 2^{20} hashes per second. This means that an attacker using a bot, can solve more puzzles than (s)he could using the cpu on the bot alone. The trade-off of course is that that attacker needs to expend more bandwidth to get the results. This may not be a problem for an attacker who uses exploited machines to launch an attack, whereas a legitimate client may not always be willing to expend extra bandwidth to get the same advantage.

It is worth noting that we can parallelize solving multiple puzzles with the cloud based implementation. In effect, we can parallelize much better than a multi-core processor, by sending multiple puzzles (in the order of 100s to 1000s) to different applications in the cloud at the same time. An attacker with enough bandwidth can achieve a reasonable amount of resource inflation at almost no computational (or economic) cost.

Furthermore, the restrictions with respect to Google's (free) App Engine, do not apply to Amazon's EC2 implementation (which is payment-based). An attacker willing to spend a modest amount will be able to run his un-interpreted executable puzzle-solving code and will also have a reasonable scaling power. We believe this would be a serious and realistic attack vector for a determined and resourceful attacker. However, this must be evaluated more thoroughly to measure the cost per benefit ratio.

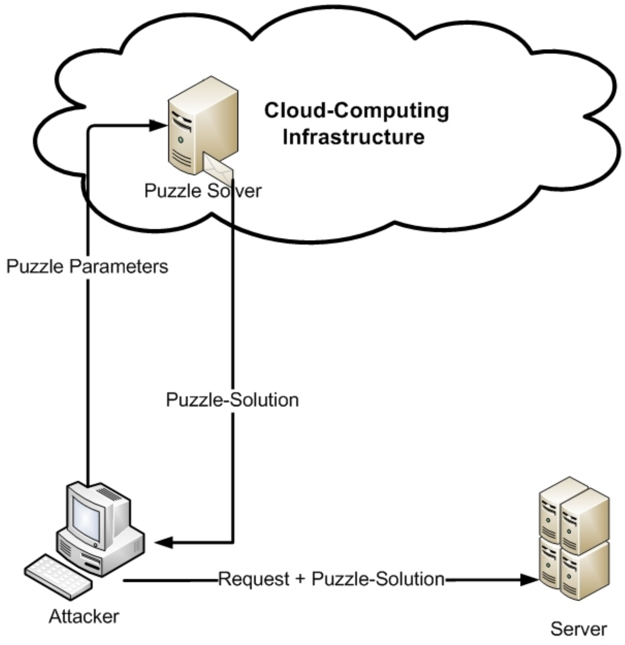
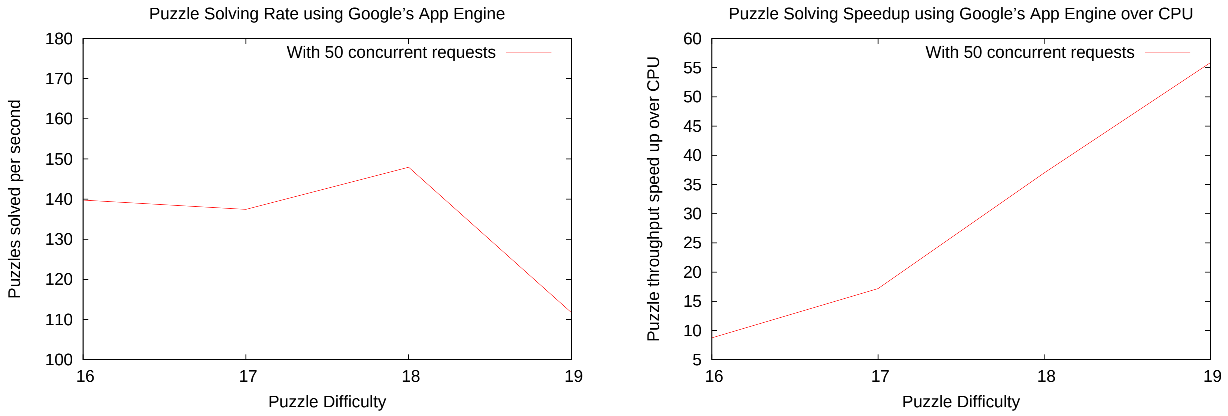


Figure 5: Using a Cloud-based puzzle-solver to attack a server

5.2 Bandwidth-based Mechanisms

As it was introduced in section 2, in bandwidth-based mechanisms clients use bandwidth as a payment to get service. The idea is that attackers are using all of the bandwidth available to them (or the maximum bandwidth they can afford to use without being detected by other mechanisms) to execute an attack, whereas legitimate clients are using only the resources they require to accomplish their less-demanding objectives. Hence legitimate clients have bandwidth to spare to get service. Below, we enumerate a few instances where by thinking out of the box an attacker can inflate the bandwidth payments of the hosts under its command and discuss the resulting ramifications.

Transport Layer Misbehavior Bandwidth-based currency schemes have often been criticized for the potential increase of traffic they can cause. This could result in congestions towards the path to the server, which would only aggravate the DoS attack situation. As a result, leading bandwidth-based currency schemes have considered congestion control as a key design element. One of the challenges with increasing bandwidth on a congestion controlled channel is that the attackers may deliberately ignore the congestion control mechanism. This results in further congesting the network, and more importantly, building an advantage over legitimate clients that slow down due to congestion. This phenomenon has been explored in [10, 22, 29] and is shown to be a real threat in today's internet. Bandwidth auctions [40] allow clients to build credit by sending dummy bytes to an accounting system and the server takes requests from clients that have built the most credit. The dummy bytes are sent over a congestion controlled channel such as TCP, and therefore are vulnerable to greedy attackers that disregard the slowdown rules. The authors do not consider the possibility and ramifications of ignoring the congestion control mechanism by the attackers. The Adaptive Selective Verification bandwidth scheme [27] employs a crude congestion control mechanism in which if a client does not receive either service or a drop acknowledgement for a number of consecutive requests, it would infer significant congestion at the network and back off. This scheme is also vulnerable to attackers that do not respect this mechanism and therefore are able to build an advantage over well-behaved clients. Another possibility is for the attackers to abuse the congestion control mechanism by creating artificial congestions that cause legitimate clients to back-off so that any bottleneck link becomes available



(a)

(b)

Figure 6: (a) Hash puzzle solving rate using Google's App Engine (Each client sent 50 puzzles serially. There were 50 separate client-threads in parallel). (b)Hash puzzle solving speedup using Google's App Engine over a CPU (capable of 2^{20} hashes per second.

for attacker(s). To further illustrate this vulnerability, we report results from prior work on low-rate TCP-targeted DoS attacks [29] and also perform simulations to show the consequences of ignoring congestion back-off rules.

Kuzmanovic *et.al.* [29] introduce periodic on-off "square wave" attacks that consist of short, maliciously-chosen duration bursts that repeat with a fixed, maliciously chosen, and slow timescale frequency. The attacker's traffic is aimed at creating congestion at a link shared with one (or more) of the legitimate contender(s) for a service. The attack aims to create a round-trip time (RTT) timescale burst that is sufficient to induce enough packet losses that the legitimate TCP flow enters a timeout and attempts to send a new packet after a retransmission timeout (RTO). If the period of DoS flow approximates the RTO of the TCP flow, the TCP flow will continually incur loss and fails to exit timeout state. If the DoS period is near but outside the RTO range, significant, but not complete throughput degradation will occur. In the reported experiments, when the attacker, the legitimate client, and the TCP receiver are on three different LANs in a campus network, a 909 Kbps average rate attack consisting of 100ms bursts at the legitimate client's RTO timescale reduces the client's throughput from 9.8 Mbps to 800 Kbps. In a slightly different setting where the attacker is in a remote location, but the legitimate client and the TCP receiver are in two different LANs in the same campus network, a similar attack profile reduces the victim's throughput from 9.8 Mbps to 1.2 Mbps. Therefore, the remaining capacity of the bottleneck link becomes available for attacker(s) to send traffic towards the server.

In the above mentioned low-rate TCP-targeted attacks, the attacker is assumed to be interested in sending low-rate traffic to stay undetected by detection mechanisms for throttling non-responsive flows such as RED-PD [32]. Given the facts that (1) such mechanisms are not yet widely deployed in the internet, (2) still the majority of routers use the FIFO queuing discipline, and (3) the nature of bandwidth-based currency schemes where some aggressiveness on the part of clients is required, an attacker may easily choose to ignore end-to-end congestion control back-off mechanisms. We perform simple ns-2 simulations to verify this intuition. Our setting consists of one legitimate node and one attacker, both trying to upload data (bandwidth payment) to a server S at the rate of 1.5 Mbps. The two clients share a 1 Mbps bottleneck link that deploys FIFO queuing. The legitimate client runs TCP, whereas the malicious attacker is assumed to run a non-responsive transport, which we simply approximate by UDP. When the attacker is not sending, the legitimate client can upload data at the rate of 964 Kbps. In presence of attacker, the legitimate client's rate drops to about 1

Kbps, while the attacker uploads at the rate of 1 Mbps. We experimented with the case where the attacker also runs TCP instead of UDP. In this case, each of the senders achieve the rate of 482 Kbps.

Greedy MAC Layer Behavior A greedy host can deliberately misuse the MAC protocol to gain bandwidth at the expense of other stations. This behavior is hidden and independent from upper layers and cannot be detected by mechanisms designed for those layers. This problem has been studied in [30, 37, 36]. For example in the context of IEEE 802.11 [2], authors of [37, 36] identify various misbehavior techniques that can result in significant bandwidth availability to the greedy hosts. The misbehavior techniques include (1) selectively scrambling frames (*i.e.* CTS, ACK, or DACK) sent by other stations in order to increase their contention windows, (2) manipulating protocol parameters, *e.g.* increasing the duration value in the RTS or DATA frames in order to cause them to adopt false NAVs [15] or reducing the backoff time. The simulation and implementation study in [37] shows that a misbehaving station can achieve throughput advantages of up to 9 times. In addition, since the throughput benefit is obtained at the expense of well-behaved stations in the same LAN, this could significantly reduce the bandwidth payment availability of legitimate nodes in the same LAN, if they happen to be contending for the same service. In this case, the misbehaving attacker can gain advantages of up to 100x compared to a competing well-behaved host in its LAN.

Using Multiple Network Interfaces Laptops sales surpassed desktop sales in the US in 2008, and are expected to repeat the same success worldwide in 2009 (`www.idc.com`). Sales of smart phones have enjoyed double digit growth in the past few years worldwide and the trend is expected to accelerate. Laptop and smart phones are usually equipped with multiple network interfaces. Ordinarily, users would choose one of the interfaces on a device for communication at a time. However, these devices can be configured to communicate using both interfaces with little effort. For example a compromised laptop in an academic department can be configured to use the ethernet and wi-fi interfaces at the same time. If this machine is compromised and is used to perform a DoS attack on a server that deploys bandwidth currency defense, it can easily build an advantage over competing hosts. We tested the upload speeds on a simple laptop in an academic department. Depending on the time of day and the interface that we consider to be primary, we recorded inflation factors between 1.5 and 2.5 to various servers across North America. This inflation factor may not seem significant *prima facie*, but considering the existing disparities in connection speed, this would severely aggravate the situation. For example consider the disparity between a dial up user (typical upload 30kbps) and a high speed cable user (typical upload speed 2000kbps). This yields to a disparity of 67x. If the cable user can use the wireless connection in parallel, an inflation factor of 1.5 would result in a disparity of 100x, which is (arguably) too high.

6 Discussion and Conclusions

In this paper we considered the threats from a range of ‘thinking out of the box’ attacks against various currency-based DoS countermeasures. In particular, we introduced the concept of resource inflation attacks on currency-based DoS countermeasures in which the attackers find ways to inflate their ownership of the resource (payment) required for getting service. As a case study, we illustrated a series of resource inflation attacks on existing DoS mechanisms. For instance, our experiments with resource inflation attacks on puzzle-based schemes showed that an attacker can use multi-core processors, cloud computing, and GPUs to inflate its resource payments. In particular, the resource inflation attack using GPUs proved to create a formidable inflation factor of up to 600x using inexpensive GPUs. Our results underscore the importance of analyzing such attack possibilities to determine whether the valid clients should match the inflation technique, ignore it because its impact is small, or change to a different scheme. Furthermore, since the puzzle schemes considered are used in a diverse set of security applications (besides DoS defense), further research in strengthening such schemes is required.

As mentioned in Section 4.4, there exist schemes that may provide greater resilience to resource inflation attacks. Serial puzzles, such as Rivest *et al.*'s work [38], are inherently resilient to parallelization using multi-core, cloud computing, or GPUs. However, the parameters in this scheme have to be chosen so that the attacker does not get an unfair advantage due to bulk-solving (especially using GPUs). Doshi *et al.* [19] propose another puzzle scheme that claims resistance to parallelization. Their scheme requires the client to solve a number of memory-bound puzzles in a breadth first manner to ensure that the solution steps in every instance eventually match a given checksum. It would be difficult to parallelize this on a cloud-based system because the puzzle-solver has to ensure that the solution it is taking is in agreement. Another promising puzzle scheme that is analyzed to have a measure of protection is presented in [25]. The authors argue that their puzzle scheme (using current state of the art eigensolvers) does not have fixed scale-up in the number of processors. In some sense, the client will only be able to scale-up to a certain number of processors and will not be able to improve the solving time beyond that.

In addition to further studies of inflation-resistant currencies, a few other directions seem interesting for future work. First, performing extensive experiments on other types of puzzle-based and bandwidth-based schemes would be of significant value. Such an analysis, for example, would result in exact numbers for inflation factors of attacks on memory-based puzzle schemes. Second, by collecting accurate data on the distribution of computing and bandwidth resources of a large body of hosts, one can build a mathematical model for threat analysis on various currency-based mechanisms. For example, the resource disparity factor can be defined as the fraction of the resource ownership of the 90-th percentile over the 10-th percentile. This fraction can be used along with the inflation factor for building a mathematical model for the threat analysis. Finally, there seems to be a need for designing a puzzle scheme which is not amenable to bulk data-parallel computations on a GPU. This might involve, for instance, developing a puzzle whose solution requires different computational operations based on the content of the puzzle itself. Such a puzzle scheme should also ensure that clients solving different puzzles of the same difficulty should require predictable and similar computational resources. Creating such a puzzle scheme, which is cheap for the server to generate and verify, is an open problem.

References

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

[2] IEEE 802.11 Wireless Local Area Networks. *http://www.ieee802.org/11/.*

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

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

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

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

[7] Telegraph floored by DDoS attack. *The Register (theregister.co.uk)*, 05/22/2007.

[8] FBI busts alleged DDoS mafia. *Security Focus (securityfocus.com)*, 08/26/2004.

[9] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Trans. Inter. Tech.*, 5(2):299–327, 2005.

[10] A. Akella, S. Seshan, R. Karp, S. Shenker, and C. Papadimitriou. Selfish behavior and stability of the internet:: a game-theoretic analysis of tcp. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 117–130, New York, NY, USA, 2002. ACM.

[11] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[12] AMD Stream Computing. www.amd.com/stream.

[13] AMD Stream Computing User Guide. http://ati.amd.com/technology/streamcomputing/.

[14] A. Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[15] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: real vulnerabilities and practical solutions. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 2–2, Berkeley, CA, USA, 2003. USENIX Association.

[16] M. Bevand. Md5 chosen-prefix collisions on gpus. *Black Hat USA*, 2009.

[17] N. Borisov. Computational puzzles as sybil defenses. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 171–176, Washington, DC, USA, 2006. IEEE Computer Society.

[18] D. Dean and A. Stubblefield. Using client puzzles to protect tls. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 1–1, Berkeley, CA, USA, 2001. USENIX Association.

[19] S. Doshi, F. Monrose, and A. D. Rubin. Efficient memory bound puzzles using pattern databases. In *ACNS*, pages 98–113, 2006.

[20] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. 2003.

[21] W. Feng, W. Feng, and A. Luu. The design and implementation of network puzzles. In *In Proc. Annual Joint Conf. of IEEE Computer and Communications Societies (INFOCOM)*, pages 2372–2382, 2005.

[22] Z. Gao and N. Ansari. Differentiating malicious ddos attack traffic from normal tcp flows by proactive tests. *Communications Letters, IEEE*, 10(11):793–795, November 2006.

[23] Google App Engine Home Page. http://code.google.com/appengine/.

[24] C. A. Gunter, S. Khanna, K. Tan, and S. S. Venkatesh. DoS protection for reliably authenticated broadcast. In *NDSS'04: Network and Distributed System Security Symposium*. The Internet Society, 2004.

[25] H. Hlavacs, W. Gansterer, H. Schabauer, J. Zottl, M. Petraschek, T. Hoeher, and O. Jung. Enhancing zrtp by using computational puzzles. *Journal of Universal Computer Science*, 14(5):693–716, 2008.

[26] A. Juels and J. G. Brainard. 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.

[27] S. Khanna, S. S. Venkatesh, O. Fatemieh, F. Khan, and C. A. Gunter. Adaptive selective verification. In *INFOCOM '08: IEEE Conference on Computer Communications*, Phoenix, AZ, April 2008. IEEE.

[28] P. Kocher, J. J. E, and B. Jun. Differential power analysis. pages 388–397. Springer-Verlag, 1999.

[29] A. Kuzmanovic and E. W. Knightly. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, New York, NY, USA, 2003. ACM.

[30] P. Kyasanur and N. H. Vaidya. Selfish mac layer misbehavior in wireless networks. *IEEE Transactions on Mobile Computing*, 4(5):502–516, 2005.

[31] B. Laurie and R. Clayton. "proof-of-work" proves not to work. In *IN WEAS 04*, 2004.

[32] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *ICNP '01: Proceedings of the Ninth International Conference on Network Protocols*, page 192, Washington, DC, USA, 2001. IEEE Computer Society.

[33] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frentz. Mitigating distributed denial of service attacks with dynamic resource pricing. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, page 411, Washington, DC, USA, 2001. IEEE Computer Society.

[34] nVidia Compute Unified Device Architecture. www.nvidia.com/cuda.

[35] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. 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*, pages 289–300, New York, NY, USA, 2007. ACM.

[36] M. Raya, I. Aad, J.-P. Hubaux, and A. El Fawal. DOMINO: Detecting MAC layer greedy behavior in IEEE 802.11 hotspots. *IEEE Transactions on Mobile Computing*, 5(12), 2006.

[37] M. Raya, J.-P. Hubaux, and I. Aad. Domino: a system to detect greedy behavior in ieee 802.11 hotspots. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 84–97, New York, NY, USA, 2004. ACM.

[38] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.

[39] M. Sherr, M. B. Greenwald, C. A. Gunter, S. Khanna, and S. S. Venkatesh. Mitigating DoS attacks through selective bin verification. In *IEEE ICNP Workshop on Secure Network Protocols (NPSec)*, pages 7– 12, 2005.

[40] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS defense by offense. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 303–314, New York, NY, USA, 2006. ACM Press.

[41] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 78, Washington, DC, USA, 2003. IEEE Computer Society.

[42] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for dos resistance. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 246–256, New York, NY, USA, 2004. ACM.