Firewall Architecture to Prevent DoS Attacks using Rate Limiting and Software Puzzle Techniques

Nithun Chand O
M. Tech. Student (Cyber Security)
Department of Computer Science & Engineering
Nehru College of Engineering and Research Centre Thrissur, Kerala

Dr. S. Subasree
Head of Dept.
Department of Computer Science & Engineering
Nehru College of Engineering and Research Centre Thrissur, Kerala

Girish R
Assistant Professor
Department of Computer Science & Engineering
Nehru College of Engineering and Research Centre Thrissur, Kerala

Abstract

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are major threat to the internet and is a serious cyber-crime. The rapid increase in number of people using internet and the development of technology has given birth to new virus and worms which can exploit our system. Attackers use the latest techniques to perform DoS attacks. There are numerous tools to perform DoS attack from millions of compromised system and can mess up any system or network in short period of time. There are many well-known counter measures available like puzzle based defense mechanism. However an attacker can inflate the ability of DoS attack in solving the puzzle by using cheap and widely available GPUs. Software Puzzles are effective against such resource inflated attacks. But this alone doesn't has a strategy to avoid unfairly delay of time during slower traffic. In this work, we propose a simple firewall architecture to prevent DoS attacks and GPU inflated DoS attacks. The proposed architecture has three stages, (i) attack detection, (ii) rate limiting and (iii) software puzzle. The traffic is analyzed, and if there is a scenario of a heavy traffic or an attack, the rate limiting technique is executed and then software puzzle are given to each and every requests to prevent users from using GPUs to solve the puzzles.

Keywords: Denial of Service, Distributed Denial of Service, GPU programming, Rate Limiting, Client Puzzle, Software Puzzle

I. INTRODUCTION

Denial of service attack (DoS) is an attempt to make a machine or a network unavailable to its users, such as to temporarily or permanently interrupts its services. A distributed denial of service attack (DDoS) is where the attack source has more than one; often millions of unique IP addresses [1]. For example, an attacker sends a large number of spam requests to an e-commerce web server. In such a scenario, the server will have to spend a lot of time processing all these spam requests and it may not have sufficient time to process request from its legitimate customers which eventually results in business loss. Usually the impact of the DoS/DDoS attack would be terrific and often takes down the server permanently. The first documented denial of service attack was on February 7, 2000, when a 15 year old Canadian hacker attacked several e-commerce including Amazon.com and eBay.com [2]. The recent news from cyber world says the world's largest denial of service attack reached 400 Gbps [3]. The studies shows the attacks are getting bigger and common. Denial of service can also be unintentional, that is a sudden increase in popularity of a website may bring the server down, being unable to handle the requests.

The distributed denial of service has two kinds of architectures. The agent handler architecture and the internet relay chat architecture. The agent handler architecture consists of clients, agents and handlers. In the primary step, attacker creates a network of computers by discovering weak systems. Attacker then installs the tools for the attack in all the machines in the network. Machines running these tools are called as the handlers. These machines also install the tools in other weak computers or hosts. In the internet relay chat architecture, a channel is created to connect the clients to agents. An example for such attack is Low Orbit Ion Cannon (LOIC).

II. DEFENSE ARCHITECTURES

A. Client Puzzles

Client puzzle is a well-known technique to defend spam and denial of service attacks. There are different classes of client puzzle schemes proposed as a solution to mitigate denial of service attacks. In the work "Client puzzles: A cryptographic countermeasure against connection depletion attacks", A. Juels and J. Brainard introduced a client puzzle protocol to fight against connection depletion attacks [6]. According to this scheme, when there is no attack sign, it accepts connection normally. Under an attack, it accepts connection selectively. The server gives a unique puzzle as a solvable cryptographic problem. The client must solve and submit the puzzle to server to gain access.

A puzzle auction protocol was proposed by X.Wang and M. Reiter in their work "Defending against denial-of-service attacks with puzzle auctions" [10]. According to this protocol, the client has to bid for service by solving puzzles. Difficulty level of the puzzle can be chosen by client. The protocol has a bidding strategy, such that client is allowed to raise the bid until it wins it.

The client puzzle scheme proposed by Feng *et al* in "The design and implementation of network puzzles" [11] use a challenge response protocol. According to this protocol, server responds to client with a puzzle of current highest level of difficulty.

Different schemes were also proposed in [12, 13, 14] based on the same concept of client puzzles. Cryptographic client puzzles are widely used for several applications like preventing junk emails, metering website usage etc. This is true when using a CPU, but by using a modern GPU, the puzzle can be easily solved since the memory latency of GPU processor and onboard memory is less than that of CPU.

B. Hash Reversal Based Puzzles

Hash reversal based puzzle schemes are important puzzle schemes suggested for DoS attack prevention. In such a scheme, the server or a trusted third party sends a puzzle seed to the client. The client needs to send the puzzle solution and compute the hash

$$p = H(x||s||r)$$

Where, *r* is the set of parameters, whose contents are dependent on puzzle question. The server checks the last *l* bits of *p* for zeros. Server also checks if the client has used the original puzzle it did send or a duplicate one. The value of *l* determines the difficulty level of the puzzle.

J. Green et al proposed an idea of proof of work (POW) mechanism in "Reconstructing Hash Reversal based Proof of Work Schemes" [7]. In this mechanism, a server request client to prove that it has done work previously, it grant resource to client request. Most POW mechanisms are puzzle based. One advantage of hash reversal based puzzles is that the cost paid by the server is less than the client. However, attackers can use GPU to solve such hash reversal based puzzles since the puzzle is parallelizable. This work also comments the hash reversal based scheme only on server side will not be effective, it need client side adaptation also.

C. Time Lock Puzzles

Time lock puzzle scheme was proposed in "Time-lock puzzles and timed-release crypto" by R. L. Rivest, A. Shamir, and D. A. Wagner [8]. The idea was to encrypt a message so that it cannot be decrypted until a predefined time has reached. Time lock puzzles require more time to solve. The puzzle was designed to be non-parallelizable to prevent client from providing more resources in short period of time. A puzzle will be provided to client and the solution of the puzzle will be the key to decrypt the message.

In this approach, for a random challenge *a*, of difficulty *l*, the client will be asked to compute

$$b = a^{2^t} (\mathrm{mod}\ n)$$

Where *n* is the product of any two large prime numbers *p* and q (usual RSA modulus). The client does not know the value of *n* and hence only way to compute the value of *b* is to perform *l* modular squaring of *a*. Due to the non-parallelizable property of time lock puzzles, it has been suggested as a DoS attack prevention mechanism. However, this approach has the problem of making the legitimate clients to wait unnecessarily and the puzzle can be parallelized and solved using GPUs.

III. INTRODUCTION TO GPU

A graphic processing unit (GPU) is an electronic circuit specially designed to perform graphical operations. GPUs are used in personal computers, mobile phones, embedded systems, and gaming consoles. Modern GPUs are very efficient at image processing and manipulating computer graphics. Modern multi core GPU can be used for general purpose computing as well as graphic processing. Additionally the major GPU manufacturers like nVidia and AMD provide convenient programming libraries to use the GPU for extreme compute applications [3].

An example for modern GPU is nVidia GPU. In the nVidia architecture, a GPU has many streaming multiprocessors consisting of many identical cores. For example, the nVidia GeForce GTX 680 has 1536 cores. A processor has a small but very fast memory. The nVidia is programmed with CUDA programming language. CUDA language allows the programmer to define C language functions, or kernels. That is a puzzle function is implemented in GPU as a kernel. When a kernel is loaded into the GPU, it is executed by multiple identical threads in parallel which produces the maximum efficiency.

A. Resource Inflation using GPUs

The modern GPUs are designed to effectively optimize the execution of single threaded programs. But a modern GPU can execute massive data-parallel programs. This is referred to as Single Instruction Multiple Data (SIMD) programming. In the latest nVidia GPU architecture, the GPU has many Streaming Multi Processors (SMs). That is when a kernel like a puzzle function is loaded into the GPU; it is executed in multiple threads in parallel for maximum efficiency. The attackers can use this facility to solve the puzzles and thus to perform a denial of service attacks. Fig.1 shows a resource inflated denial of service attack using a GPU.

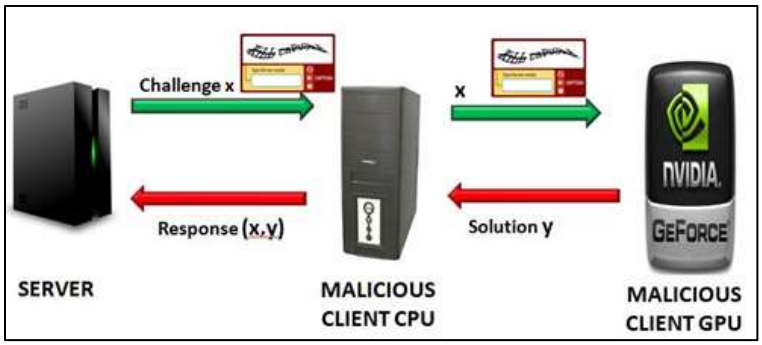

Fig. 1: GPU used to solve client puzzles (GPU image source from [3])

B. Difference between CPU and GPU

A CPU is a general purpose processor for computation whereas a GPU is a special purpose processor optimized for calculation commonly required for computer graphics. GPUs are far better than CPUs in graphic control functions, and are a programmable and powerful computational device in its own rights. Graphic processing can also be done in CPUs, but will not produce the results like a programmed GPU.

Both CPU and GPU software can be implemented using high level language like C. However the low level instruction sets are totally different. Particularly some instruction operations are not supported in GPU software [4]. Table 1 lists some of these instructions.

Table – 1
Example of CPU only instructions

CPU Instructions	*Difference in GPU*
Reading local storage CPU can read the local cookie and storage	*GPU cannot directly read CPU storage*
Allocate large memory CPU can allocate large memory if required by the programs	*GPU has smaller memory than CPU*
Try-Catch CPU Support exception handling	*GPU does not support exception handling*
Goto (address) CPU support branch instructions	*GPU does not support branch instructions*
Network Interface CPU support all network activities	*GPU can't support networking functions*
Human-Machine Interface CPU spport human machine interface	*GPU can't support Human Machine Interface*
Create new class CPU supports dynamic coding, creating a new class	*GPU does not support dynamic coding*

C. Modern GPU Architecture

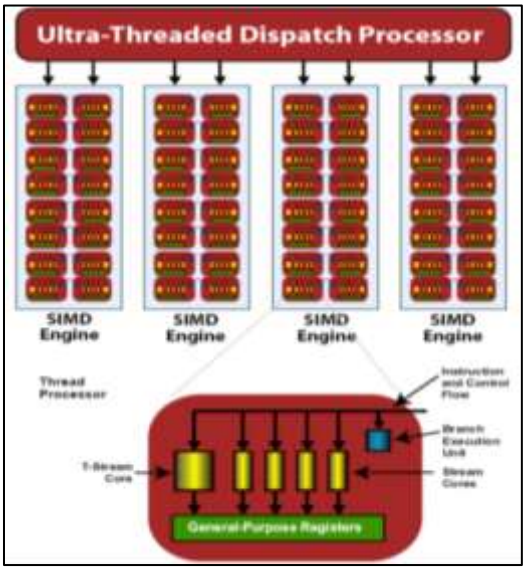

Fig. 2: GPGPU Architecture from [15]

Modern GPUs has the capability of processing large amount of data in parallel. There has been significant increase in the efficiency at executing data parallel, for graphical as well as non-graphical computing. This paradigm is known as General Purpose Graphic Processing Unit (GPGPU). The architecture of the AMD GPU is represented in fig. 2.

GPU has a large number of SIMD engines. For example AMD HD4850 GPU has over 800 thread processors. Each thread processor has access to its own general purpose registers and can also access the GPU memory.

Unlike the CPUs, a GPU based thread is very light weight and can be started with the minimum overhead. All the threads in the GPU execute the same instruction for maximum efficiency. This way legitimate clients and attackers can solve the puzzle schemes using a GPU.

D. Performance of CPU and GPU

In older days GPU were used to accelerate graphical displays only. However in the last few years the performance of the GPU increased at a rapid pace due to the property of parallelism inherent.

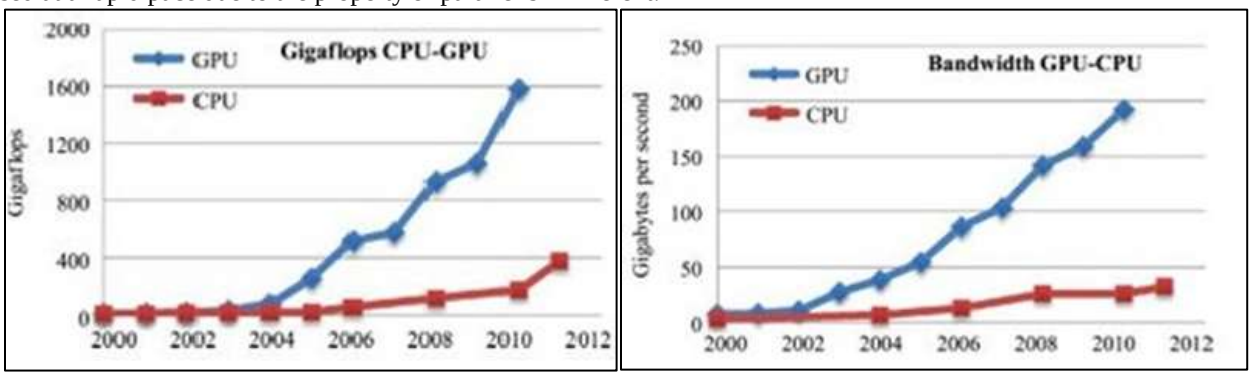

Fig. 3: Performance of CPU and GPU [16]

Researches show that this will continue in the future and the design of the computer architecture will focus more on GPUs. Fig.3 shows the performance comparison of CPU and GPU in gigaflops and bandwidth the last few years [16]

IV. SYSTEM MODEL

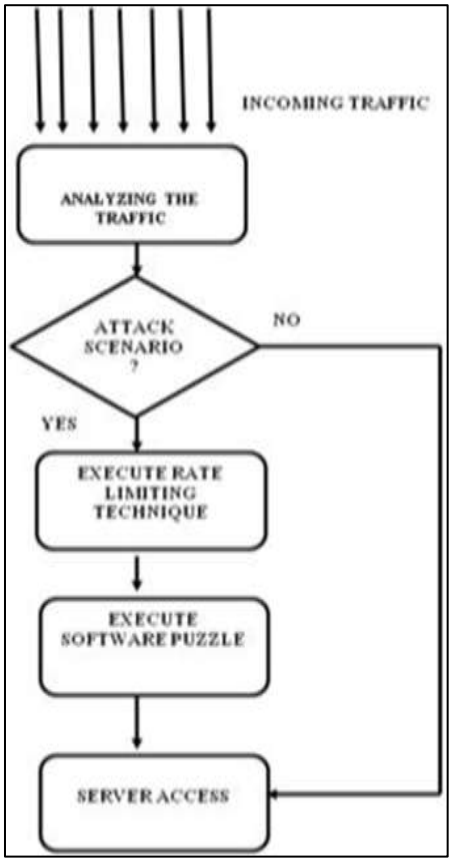

Fig. 4: Proposed System Architecture

In our proposed model, we focus on traffic limiting and preventing gpu inflated denial of service attacks. We propose a three tier strategy to prevent the denial of service attacks. A counter is set at the server side for analyzing the traffic. A threshold is set as the server capacity and it varies with the capacity of the server. If the traffic flow exceeds the threshold value, we assume it as an attack and rate limiting is executed. Round robin scheduling is used to rate limit the traffic and the software puzzle is executed to mitigate gpu inflated dos attacks. Fig. 4 shows the proposed architecture.

V. ANALYZING THE TRAFFIC

Capacity threshold monitoring is a process to monitor each and every action in the server like bandwidth usage, disk usage etc. and we are notified when the limit is reached. By setting a threshold for the server capacity, the performance can be improved by increasing the bandwidth, disk space etc. In our proposed model, once the traffic meets the threshold, we consider it as an attack and the rate limiting is executed. All traffic within the server threshold is given server access.

VI. RATE LIMITING

Rate Limiting also called as bandwidth throttling is an intentional slowing of the internet service to regulate traffic and avoid congestion. There are different techniques for the rate limiting and load balancing in servers. The rate limiting algorithms work on the principle that in which situation the work load is assigned. In our model, we use round robin scheduling for rate limiting.

A. Round Robin Scheduling Algorithm (RR)

Round Robin scheduling is one of the simplest methods for distributing client requests across the servers. Round robin load balancer forwards a client request to each server in turn. Assuming there are three servers (servers A,B and C), request 1 would go to server A, request 2 goes to server B, request 3 goes to server C, and finally request 4 goes to server A thus completing the round robin cycle. It can be modifies based on location, time periods etc. That is requests in time period X goes to one server A, and requests in time period Y goes to server B etc. Round robin treats all servers equally, regardless of connections or response time.

VII. SOFTWARE PUZZLE

Software Puzzle is the final step in the process of defending the gpu inflated denial of service. A normal client puzzle as in the existing client schemes is fixed and disclosed in advance. Such puzzles are called data puzzle, or it is referred as software puzzle. A data puzzle has a challenge data only. An example is graphical captcha. A software puzzle has both data puzzle and dynamically generated software.

A. Software Puzzle Architecture

Fig.5 shows the software puzzle architecture [9]. It consists of a warehouse which contains the different kinds of blocks. Random code blocks are extracted from it and puzzle is created.

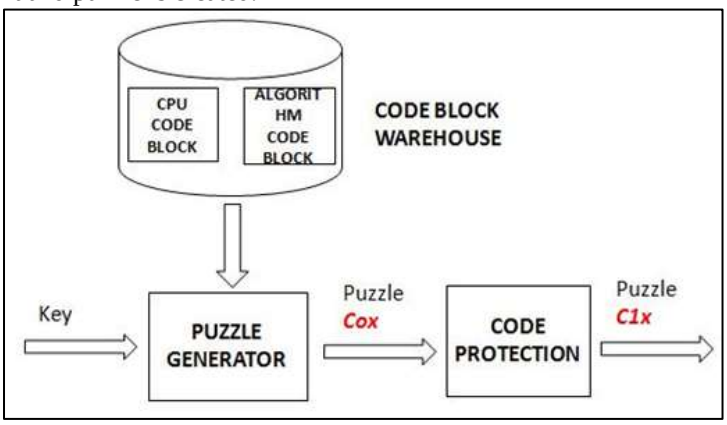

Fig. 5: Software puzzle architecture

Instruction codes are stored as java bytes or C binary code in the warehouse. CPU only instructions that cannot be executed in GPUs are stored in the code block.

B. Software Puzzle Generation

Software puzzle generation has three steps. (i) Puzzle Core generation, (ii) Puzzle Challenge generation, and (iii) Protecting the code

In the puzzle core generation step, the code block warehouse chooses a random code block based on a hash function and secret key. The chosen blocks are gathered into a puzzle core.

In the puzzle challenge generation, the server access a message from the public data like ip address, port numbers or local storage and produces a challenge. This is similar to encrypting a plain text using these parameters like ip address. Attacker cannot force GPU to solve this puzzle.

The code can be protected by an encryption technique. There are many code obfuscation tools available like VM protect which can be used to protect the software puzzle from hacking.

VIII. ADDITIONAL SECURITY FEATURES

Our firewall can be implemented in large scale and small scale networks. For highly secured transactions security mechanisms like OTP (One Time Password) can be added along with the software puzzle.

In this architecture the server computes a secret key to encrypt the puzzle, and the key is send to the client mobile number. Client has to use this secret key to decrypt the message. This is a time consuming process and hence is optional. If secret keys are not used, the puzzle will be encrypted using the code only.

IX. SIMULATION

The simulation was performed in Network Simulator NS2. The threshold of the server capacity was given as 5 and the result was analyzed. Number of clients was entered manually.

First, the number of requests was given within the threshold and the traffic was observed to be normal. The result is shown in the figure 6.

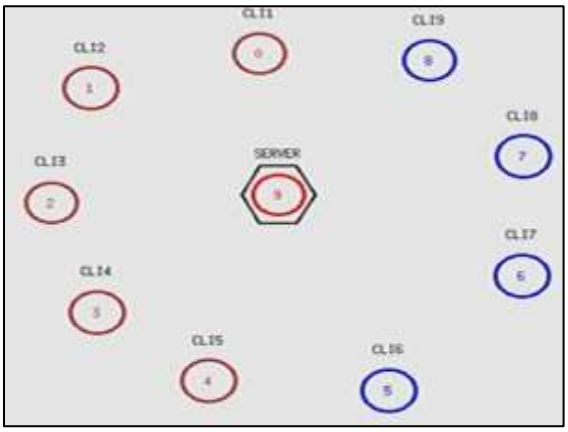

Fig. 6: All requests communicating with server

When the requests were increased above the threshold, the requests were clustered and serviced according to round robin scheduling. The result is shown in figure7.

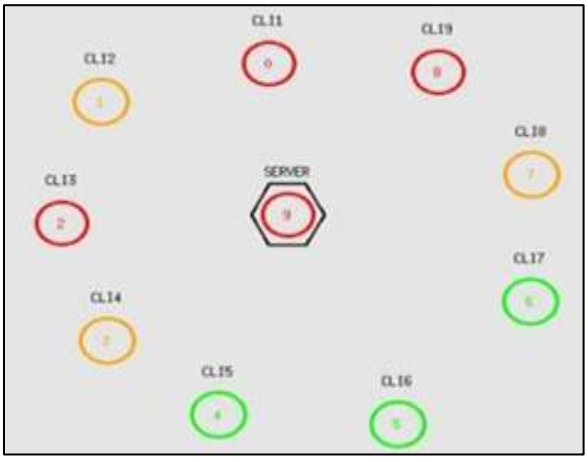

Fig. 7: Requests are clustered and serviced

The puzzle was integrated along with this and executed in the command window (Linux terminal). But we also executed the whole software puzzle and the additional security features in java.

A Server and three clients are created. A secure connection is established with the client and server using the localhost. Client request the server for server data. On receiving the request from client1, server access the public data like IP address and compute three secret keys.

The puzzle is encrypted and sent to the requested client. Client has to decrypt the puzzle using the keys (in case there is a messaging server involved) and the physical address and IP address. The file is received upon proper decryption or the connection is closed. This is shown in figure 8.

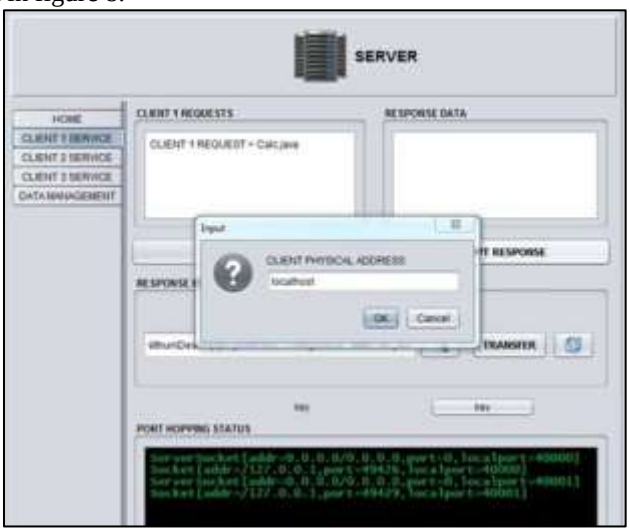

Fig. 8(a): Server accessing the physical address of client

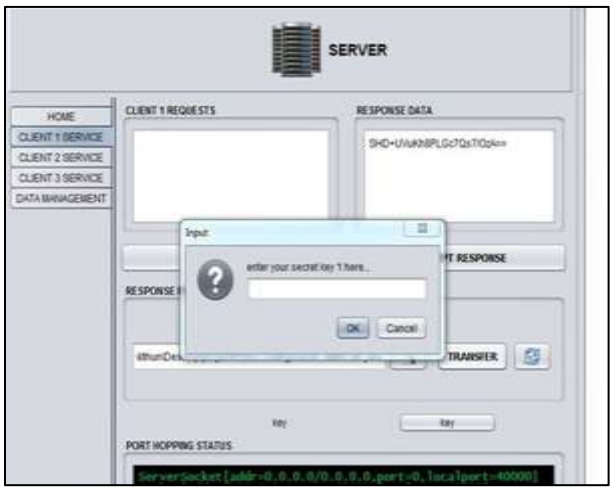

Fig. 8(b): Server computing the secret keys

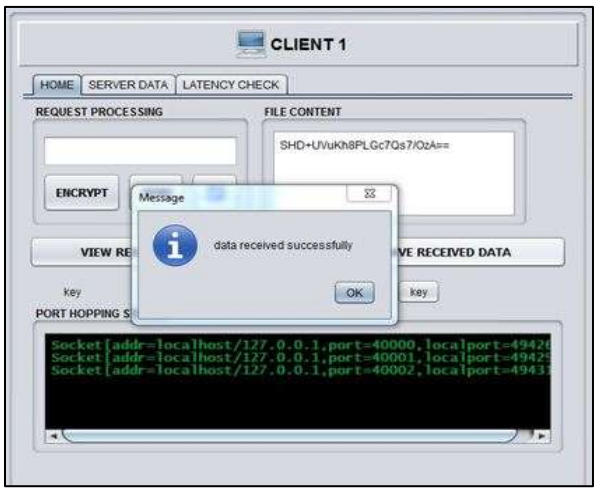

Fig. 8(c): Client receives file on successful decryption

X. CONCLUSION

In this work, we have implemented a firewall architecture to defend denial of service attacks and GPU inflated denial of service attack. This architecture combines three best technologies and produces an effective security architecture. This idea focus on defending DoS but this idea can also be extended into the cloud computing security. For example, during the time of cloud debugging or backup etc when the server is busy, this system can be run which will stop DoS attacks that time.

Another advantage of the system is this can be added along with any other firewall architecture and security systems. This architecture is flexible; it can be used for small networks like college servers to huge banking servers. In the rate limiting technique, we have selected the round robin scheduling but it can also be used with other technologies depending on our needs. This system is a solution to unnecessary waiting of legitimate clients due to the conventional client puzzle system.

REFERENCES

[1] Wikipedia, Denial-of-Service attack. Available at https://en.wikipedia.org/wiki/Denial-of-service_attack
[2] Techworld, Available: www.techworld.com/news/security
[3] NVIDIA CUDA. (Apr. 4, 2012). NVIDIA CUDA C Programming Guide, Version 4.2. Available: www. developer.download.nvidia.com
[4] R. Shankesi, O. Fatemieh, and C. A. Gunter, "Resource inflation threats to denial of service countermeasures," Dept. Comput. Sci., UIUC, Champaign, IL, USA, Tech. Rep., Oct. 2010

[5] K. Iwai, N. Nishikawa, and T. Kurokawa, "Acceleration of AES encryption on CUDA GPU," Int. J. Netw. Comput., vol. 2, no. 1, pp. 131–145, 2012
[6] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," Proc. Netw. Distrib. Syst. Secur. Symp., 1999, pp. 151–165
[7] J. Green, J. Juen, O. Fatemieh, R. Shankesi, D. Jin, and C. A. Gunter, "Reconstructing Hash Reversal based Proof of Work Schemes," in Proc. 4th USENIX Workshop Large-Scale Exploits Emergent Threats, 2011
[8] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," Dept. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep. MIT/LCS/TR-684, Feb. 1996
[9] Yongdong Wu, Zhigang Zhao, Feng Bao, and Robert H. Deng, "Software Puzzle: A Countermeasure to Resource- Inflated Denial-of-Service Attacks", in IEEE Transactions On Information Forensics And Security, vol. 10, no. 1, Jan 2015
[10] X. Wang and M. Reiter, "Defending against denial-of-service attacks with puzzle auctions," in Proc. IEEE Security and Privacy '03, pages 78–92, 2003
[11] W. Feng, E. Kaiser, W. Feng, and A. Luu, "The design and implementation of network puzzles," in Proc. INFOCOM'05, 2005
[12] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," ACM Trans. Inter. Tech., 5(2):299–327, 2005
[13] C. Dwork, A. Goldberg, and M. Naor, "On memory-bound functions for fighting spam," 2003
[14] 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
[15] AMD Stream Computing User Guide. http://ati.amd.com/technology/streamcomputing
[16] J.Y. Chen. Gpu technology trends and future requirements. In Electron Devices Meeting (IEDM), 2009 IEEE International, pages 1-6, 2009