ISSN XXXX XXXX © 2017 IJESC

Research Article Volume 7 Issue No.5

Software Puzzle for GPU Inflated DoS Attack

B.Banu Priya[1], N. Mohan[2]
ME Student[1], Associate Professor[2]
Department of CSE
Kalaignar Karunanidhi Institute of Technology, Coimbatore, India

Abstract:
Denial-of-service (DoS) and distributed DoS (DDoS) are among the major threats to cyber-security, and client puzzle, which demands a client to perform computationally expensive operations before being granted services from a server, is a well-known countermeasure to them. However, an attacker can inflate its capability of DoS/DDoS attacks with fast puzzle solving software and/or built-in graphics processing unit (GPU) hardware to significantly weaken the effectiveness of client puzzles. In this paper, we study how to prevent DoS/DDoS attackers from inflating their puzzle-solving capabilities. To this end, we introduce a new client puzzle referred to as software puzzle. Unlike the existing client puzzle schemes, which publish their puzzle algorithms in advance, a puzzle algorithm in the present software puzzle scheme is randomly generated only after a client request is received at the server side and the algorithm is generated such that: 1) an attacker is unable to prepare an implementation to solve the puzzle in advance and 2) the attacker needs considerable effort in translating a central processing unit puzzle software to its functionally equivalent GPU version such that the translation cannot be done in real time. Moreover, we show how to implement software puzzle in the generic server-browser model.

Keywords: Code obfuscation, Code block warehouse, Denial of Service, Gpu inflated dos attack, Client Puzzle, Virtual Channels.

1. INTRODUCTION

1.1 DENIAL OF SERVICE ATTACK

A Denial of Service (DoS) attack does not steal or damage the server but blocks or prevents access to the server or website. Such DoS attacks target the network bandwidth or connectivity. Such attacks flood the network degrading the service provided to a genuine user who is not able to send or receive response from server. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. There are basically three Types of DOS attacks: Smurf, UDP flood and SYN flood attacks. Despite the significant varieties of attacks there is a common objective amongst all types of DoS attacks. The attackers aim to exhaust the resources of the system that includes cpu cycles, memory, disk space and network bandwidth. The attackers generate too many requests which is feasible since they pay very little or nothing to request a service. Often their cost is only of sending the request on the network. However the attack can vary significantly in many aspects including the target and protocol layer of the network, distribution of attack sources, the strategy employed and the impact. In most puzzle schemes, each puzzle requires an approximately fixed number of cryptographic operations, such as hashing, modular multiplication, or modular exponentiation, to compute the puzzle solution. Thus, the more an attacker wants to overwhelm the server, the more puzzles she has to compute, consequently the more computational resources of her own she needs to consume. The construction and verification of the puzzle are designed to be very efficient to avoid DoS on the puzzle scheme itself.

1.1.1DISTRIBUTED DDOS

A distributed denial-of-service (DDoS) attack occurs when multiple systems flood the bandwidth or resources of a targeted system, usually one or more web servers. Such an attack is often the result of multiple compromised systems (for example, a botnet) flooding the targeted system with traffic. A botnet is a network of zombie computers programmed to receive commands without the owners' knowledge. When a server is overloaded with connections, new connections can no longer be accepted. The major advantages to an attacker of using a distributed denial-of-service attack are that multiple machines can generate more attack traffic than one machine, multiple attack machines are harder to turn off than one attack machine, and that the behaviour of each attack machine can be stealthier, making it harder to track and shut down. These attacker advantages cause challenges for defense mechanisms. For example, merely purchasing more incoming bandwidth than the current volume of the attack might not help, because the attacker might be able to simply add more attack machines. This, after all, will end up completely crashing a website for periods of time.

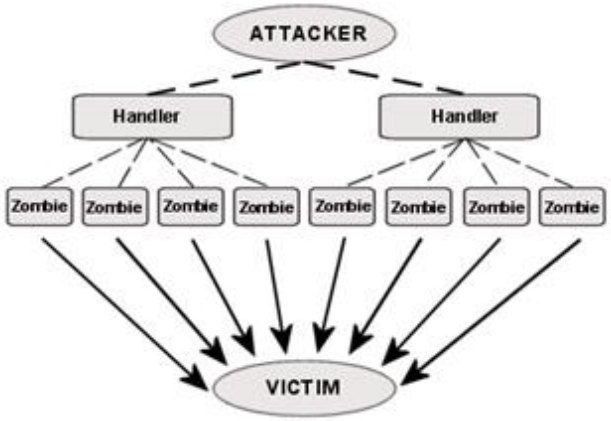

Figure.1.distributed denial of service attack

1.1.2 CODE OBFUSCATION

In software development, manual **obfuscation** is the deliberate act of creating obfuscated code, i.e. source

or machine code that is difficult for humans to understand. Like obfuscation in natural language, it may use needlessly roundabout expressions to compose statements. Programmers may deliberately obfuscate code to conceal its purpose (security through obscurity) or its logic, in order to prevent tampering, deter reverse engineering, or as a puzzle or recreational challenge for someone reading the source code.

1.2 GPU PROGRAMMING

General-purpose computing on graphics processing units (**GPGPU**, rarely **GPGP** or **GP²U**) is the use of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU). The use of multiple video cards in one computer, or large numbers of graphics chips, further parallelizes the already parallel nature of graphics processing. In addition, even a single GPU-CPU framework provides advantages that multiple CPUs on their own do not offer due to the specialization in each chip. Essentially, a GPGPU pipeline is a kind of parallel processing between one or more GPUs and CPUs that analyzes data as if it were in image or other graphic form. While GPUs operate at lower frequencies, they typically have many times the number of cores. Thus, GPUs can operate on pictures and graphical data effectively far faster than a traditional CPU. Migrating data into graphical form and then using the GPU to scan and analyze it can result in profound speedup. In order to elaborate software puzzle, we recap its rival GPU-inflated DoS attack in advance. When a client wants to obtain a service, she sends a request to the server. After receiving the client request, the server responds with a puzzle challenge *x*. If the client is genuine, she will find the puzzle solution *y* directly on the host CPU, and send the response *(x, y)* to the server. However, as shown in Fig. 1, by using the similar mechanism in accelerating calculation with GPU, a malicious user who controls the host will send the challenge *x* to GPU and exploit the GPU resource to accelerate thepuzzle-solving process.

2 .Framework of Software Puzzle

In order to defeat the GPU-inflated DoSattack, we extend data puzzle to software puzzle as shown in Figure. At the server, the software puzzle scheme has a code block warehouse **W** storing various software instruction blocks. Besides, it includes two modules: generating the puzzle $C0_x$by randomly assemblying code blocks extracted from the warehouse; and obfuscating the puzzle $C0_x$ for high security puzzle $C1_x$.

1.2.1 Basic GPU-Inflated DoS Attack

In order to elaborate software puzzle, we recap its rival GPU-inflated DoS attack in advance. When a client wants to obtain a service, she sends a request to the server. After receiving the client request, the server responds with a puzzle challenge *x*. If the client is genuine, she will find the puzzle solution *y* directly on the host CPU, and send the response *(x, y)* to the server. However, as shown in Fig. 1, by using the similar mechanism in accelerating calculation with GPU, a malicious user who controls the host will send the challenge *x* to GPU and exploit the GPU resource to accelerate the puzzle-solving process.

2 .Framework of Software Puzzle

In order to defeat the GPU-inflated DoSattack, we extend data puzzle to software puzzle as shown in Figure. At the server, the software puzzle scheme has a code block warehouse **W** storing various software instruction blocks. Besides, it includes two modules: generating the puzzle $C0_x$by randomly assemblying code blocks extracted from the warehouse; and obfuscating the puzzle $C0_x$ for high security puzzle $C1_x$.

2.1 Code Block Warehouse Construction

The code block warehouse **W** stores compiled instruction blocks $\{b_i\}$, *e.g.*, in Java byte code, or C binary code. The purpose to store compiled codes rather than source codes is to save server's time; otherwise, the server has to take extra time to compile source codes into compiled codes in the process of software puzzle generation. The intuitive requirements for each block are

- In order to assemble the code blocks together, each block has well-defined input parameters and output parameters such that the output from one block can be used as the input of the following blocks.

Table I

Inbound			
Source	Protocol	Port Range	Comments
0.0.0.0/0	TCP	80	Allow inbound HTTP access from anywhere.
0.0.0.0/0	TCP	443	Allow inbound HTTPS access from anywhere.
Outbound			
Destination	Protocol	Port Range	Comments
The ID of the security group for your web application servers	TCP	8080	Allow outbound HTTP access to instances in the specified security group.

The size of each code block is decided by the security parameter κ. Given that the size of software puzzle is constant, if the block size is smaller, there are more blocks on average such that more puzzles can be constructed. Thus smaller block size implies higher security level because an attacker has to spend more effort to figure out a puzzle in question. The shortcoming of small block size is that the server has to spend more time in extracting the basic blocks and assembling the extracted blocks into software puzzle. Preferably, the warehouse stores both Java byte code and the corresponding C binary code. Because the former is applicable to different OS platforms but slow, it is suitable to deliver the software puzzle to the client in the format of Java byte code. In contrast, the later is fast and is used by the server for generating the stored pair *(x,y)*. As a result, this Java-C hybrid scheme ensures that the server has advantage over the client/adversary in terms of resource consumption, as well as the support of cross-platform deployment. In general, code blocks can be classified into two categories: CPU-only instruction block and data puzzle algorithm block.

2.1 Traffic Analysis Code Block

2.1.1 CPU-Only Instruction Block

Unlike CPU, GPU is designed for the predictable graphic processes such as matrix operations, not generic logic processing. As branching operations (e.g., try-catch-

finally,goto) are inherently non-predictable and are non-parallel able, executing them in GPU is slow such that the major merit of GPU cannot be exploited by the attacker; Secondly, some hardware-related operations such as reading hardware input and surfing network, cannot be performed on GPU; Thirdly, the state-of-the-art GPUs do not support dynamic thread generation; Fourthly, the high-speed shared memory is shared by all the GPU thread blocks together such that the size of fast accessible memory available to each thread is small. Therefore, if the puzzle kernel demands large shared memory, the GPU parallelism potential will be restricted seriously, or the threads have to access the global memory at a much slower speed. Therefore, we can exploit the instructions which are different between GPU and CPU as components to design software puzzle. Table I lists some of these instructions.

TABLE II

Instruction	Difference exploited
1.Read local cookie	GPU can not directly read CPU storage
2.Allocate large memory	GPU has much smaller memory than CPU
3.Try-catch	GPU does not support except handling
4.Goto (address)	GPU does not support branch
5.Network interface	GPU cann't support networking function
6.Human-machine interface	GPU cann't support
7.Create new class	GPU does not support dynamic code
8.Create new thread	GPU does not support child thread

2.2 CPU-ONLY INSTRUCTIONS

2.2.1 Data Puzzle Algorithm Block

Similar to the blocks in data puzzle, algorithm blocks perform the mathematical operations only. For example, in an AES round, *ShiftRows* code block outputs a transformed message matrix (or state), which can be used as input of any other operation such as *MixColumn* code block without incurring parameter mismatch errors.

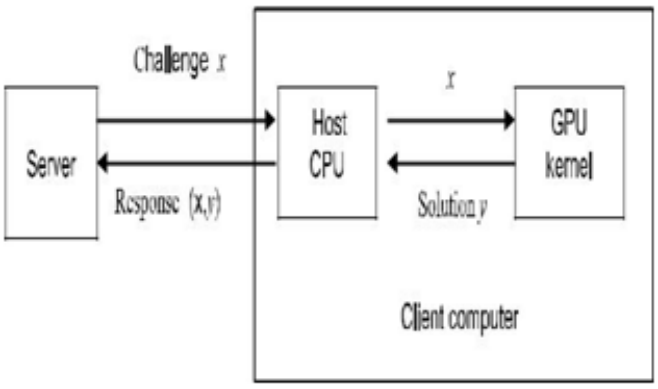

Figure.2. Software puzzle architecture

2.2.2 GPU INFLATED DOS ATTACK

Modern GPUs have many processing cores that can be used for general-purpose computing as well as graphics processing. Additionally, nVidia and AMD, the major GPU vendors, provide convenient programming libraries to use their GPUs for intensive computation applications.

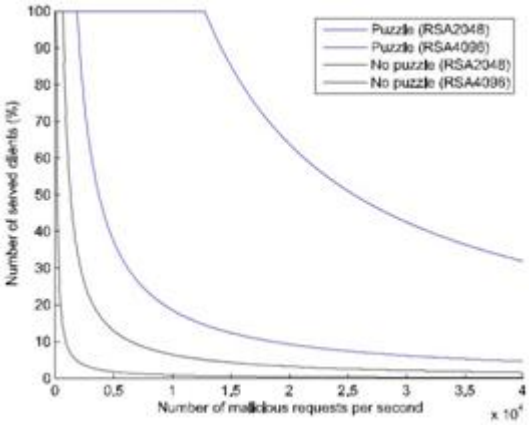

Figure.3.Mallicious Attack

3. GPU INFLATED DOS ATTACK

3.1 GPU BASED THREAD

GPGPUs have been used to improve the speed of various programs such as folding and computational chemistry. Recently, GPUs were also used for finding MD5 chosen-prefix collisions. GPU-based architectures consist of a large number of SIMD engines. Each engine contains a number of thread processors. Each thread processor unit has access to its own general purpose registers as well as access the GPU's memory. The thread dispatcher manages various threads and is invoked by the client on the CPU. Threads in GPUs are not analogous to processor threads on a general purpose computer. A GPU-based thread is a very lightweight thread that can be started with minimum overhead. All GPU-based threads (within a single SIMD engine block) need to execute the same instruction (possibly on different input data) for maximum efficiency. In effect, we cannot use different threads to perform completely distinct computations on different pieces of data. When two (or more) threads running on a thread processor in an SIMD engine need to execute different instructions, the GPU will ensure that only one of the threads executes at any given time. Although GPUs only support this limited notion of parallelism, for the right kinds of processing algorithms GPUs offer large advantages in efficiency.

3.2 GPU PROGRAMMING MODEL

A GPU-based program is usually written in a way that takes advantage of the inherently data parallel programs (such as matrix multiplication, simulations, etc.). The GPU-based platform can run a program such that each thread of the program operates on a distinct block of data. All of such threads can run simultaneously on the stream processors as long as there are enough stream processors on the GPU. In case the data elements are larger than the number of stream processors, the Thread Dispatcher manages the available processors for the threads. For instance, consider the following snippet of code written in AMD's stream computing language: kernel voidsum (float a<>, float b<>, out float c<>){ c = a + b; } The inputs are the streams a and b and the output is the stream c. When this program, along with the stream reading and writing operations (not shown) is run, the GPGPU infrastructure reads the input streams from the CPU's main memory into the GPU's memory. Once loaded, each stream processor will run simultaneously and independently on a slice of the input, corresponding to the two input vectors and produces the result in the output. This program computes the sum of two vectors a,b and stores it in the output vector c. Although each of the threads is working on a different piece of data, they are each effectively performing the same operation.

This results in maximum efficiency in a data parallel algorithm on a GPU.

4. RESULTS AND CONCLUSION

Software puzzle aims to prevent GPU from being used in the puzzle-solving process based on different instruction sets and real-time environments between GPU and CPU. Conversely, an adversary may attempt to deface the software puzzle scheme by simulating the host on GPU, cracking puzzle algorithm, re-producing GPU-version puzzle, or abusing the access priority in puzzle-solving. If an attacker is able to run a CPU simulator over GPU environment, the software puzzle can be executed on GPU directly. However, this simulator-based attack may be impractical in accelerating the puzzle-solving process.

5. REFERENCES

[1]. J. E. Smith and R. Nair (2005), 'Virtual Machines: Versatile Platforms for Systems and Processes'. San Mateo, CA, USA: Morgan Kaufmann,p. 19.

[2]. M. Jakobsson and A. Juels (1999),'Proofs of work and bread pudding proto-cols', in *Proc. IFIP TC6/TC11 Joint Working Conf. Secure Inf. Netw.,Commun. Multimedia Secur.*, pp. 258–272.

[3]. A. Juels and J. Brainard (1999), 'Client puzzles: A cryptographic countermeasure against connection depletion attacks,' in *Proc. Netw. Distrib. Syst.Secur. Symp.*, pp. 151–165.

[4]. E. Kaiser and W.-C. Feng, "mod_kaPoW: Mitigating DoS with trans-parent proof-of-work," in *Proc. ACM CoNEXT Conf.*, 2007, p. 74.

[5]. NVIDIA CUDA. (Apr. 4, 2012). *NVIDIA CUDA C Programming Guide,Version 4.2*. [Online]. Available: http:// developer.download.nvidia.com/

[6]. X. Wang and M. K. Reiter, "Mitigating bandwidth-exhaustion attacks using congestion puzzles," in *Proc. 11th ACM Conf. Comput. Commun.Secur.*, 2004, pp. 257–267.