ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

IP Blocking For Prevention of Denial-of-Service Attacks Using Dynamic Puzzle

Somshekar Hannikeri, Ram Sankar. G

M.Tech II Year Scholar (CNE), Dept of ISE, The Oxford College of Engineering, Bangalore, Karnataka, India

Assistant Professor, Dept of ISE, The Oxford College of Engineering, Bangalore, Karnataka, India

ABSTRACT: Denial-of-service (DoS) and Distributed DoS (DDoS) are among the major threats to cyber-security and Internet of Things (IoT). Web services architecture are vulnerable to DoS/DDoS attacks where the attacker creates a huge whole sum of traffic and forces it to the server, Which induces significant harm to the victim server. These attacks are typical explicit and intentional attempts to disrupt legitimate user's access to services. Client puzzles are well known defense to these attacks, which demands a client to perform computationally expensive operations before being granted services from a server. However, an attacker can use their GPU to solve the puzzle and weaken the effectiveness of puzzle because existing client puzzle schemes publish their puzzle algorithms in advance. So in the proposed model a puzzle algorithm in the present dynamic puzzle scheme is randomly generated only after a client request is received at the server side and the algorithm is generated such that an attacker is unable to prepare an implementation to solve the puzzle in advance and the instructions in the puzzle can only be solved by the CPU. Further by blocking clients who try to execute the attack by sending wrong solution to the requested puzzle defends the DoS attacks.

KEYWORDS: Denial of service; Distributed denial of service; GPU programming; IP blocking; Dynamic puzzle;

I. INTRODUCTION

DENIAL of Service (DoS) attacks and Distributed DoS (DDoS) attacks attempt to deplete an online service's resources such as network bandwidth, memory and computation power by overwhelming the service with bogus requests. For example, a malicious client sends a large number of garbage requests to an HTTPS bank server. As the server has to spend a lot of CPU time in completing SSL handshakes, it may not have sufficient resources left to handle service requests from its customers, resulting in lost businesses and reputation [1].

DoS and DDoS are effective if attackers spend much less resources than the victim server or are much more powerful than normal users. In the example above, the attacker spends negligible effort in producing a request, but the server has to spend much more computational effort in HTTPS handshake (*e.g.*, for RSA decryption). In this case, conventional cryptographic tools do not enhance the availability of the services; in fact, they may degrade service quality due to expensive cryptographic operations [1].

II. RELATED WORK

The seriousness of the DoS/DDoS problem and their increased frequency has led to the advent of numerous defense mechanisms [6]. DoS/DDoS attacks are targeted on server computation power. Let γ denote the ratio of resource consumption by a client and a server. Obviously, a countermeasure to DoS and DDoS is to increase the ratio γ, *i.e.*, increase the computational cost of the client or decrease that of the server. Client puzzle [7] is a well-known approach to increase the cost of clients as it forces the clients to carry out heavy operations before being granted services. Generally, a client puzzle scheme consists of three steps: puzzle generation, puzzle solving by the client and puzzle verification by the server. Hash-reversal is an important client puzzle scheme which increases a client cost by forcing the client to crack a one-way hash instance. Technically, in the puzzle generation step, given a public puzzle function P derived from one-way functions such as SHA-1 or block cipher AES, a server randomly chooses a puzzle challenge x, and sends x to the client [1]. In the puzzle-solving and verification steps, the client returns a puzzle response (x, y), and

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

if the server confirms $x = P(y)$, the client is able to obtain the service from the server. In this hash-reversal puzzle scheme, a client has to spend a certain amount of time *tc* in solving the puzzle (*i.e.*, finding the puzzle solution *y*), and the server has to spend time *ts* in generating the puzzle challenge *x* and verifying the puzzle solution *y*. Since the server is able to choose the challenge such that *tc »ts* for normal users, *i.e.*, $\gamma \gg 1$, an attacker cannot start DoS attack efficiently by solving many puzzles [1]. A client puzzle can significantly reduce the impact of DoS attack because it enables a server to spend much less time in handling the bulk of malicious requests. Of course, optimizing the puzzle verification mechanism is very important and doing so will undoubtedly improve the server's performance [5].

Obviously, if a puzzle is designed based on client's GPU capability, the GPU-inflation DoS does not work at all. However, we do not recommend to do so because it is troublesome for massive deployment due to (1) not all the clients have GPU-enabled devices; and (2) an extra real-time environment shall be installed in order to run GPU kernel [1]. By exploiting the architectural difference between CPU and GPU, this paper presents a new type of client puzzle, called dynamic puzzle, to defend against GPU-inflated DoS and DDoS attacks. Unlike the existing client puzzle schemes which publish a puzzle function in advance, the dynamic puzzle scheme dynamically generates the puzzle function $P(\cdot)$ in the form of a software core *C* upon receiving a client's request [1]. However, a malicious client may attempt to offload the puzzle-solving task into its GPU. In this case, the malicious client has to translate the CPU dynamic puzzle into its functionally equivalent GPU version because GPU and CPU have totally different instruction sets designed for different applications [1].

Note that this translation cannot be done in advance since the dynamic puzzle is formed dynamically and randomly. As rewriting/translating a dynamic puzzle is time-consuming, which may take even more time than solving the puzzle on the host CPU directly, dynamic puzzle thwarts the GPU-inflated DoS attacks. To demonstrate the applicability of dynamic puzzle, we use Applet to implement dynamic puzzles such that the dynamic puzzle implementation has the same merits as [4] in terms of easy deployment, but overcomes its security weaknesses.

III. PROPOSED ALGORITHM

Client puzzles are divided into two types. If a puzzle is fixed and disclosed in advance, the puzzle is called a data puzzle; otherwise, it is referred to as a dynamic puzzle. Data puzzle aims to enforce the client's computation delay of the inverse function $P^{-1}(x)$ for a random input x; while dynamic puzzle aims to deter an adversary from understanding/ translating the implementation of a random puzzle function $P(\cdot)$ [1]. Although a dynamic puzzle scheme does not publish the puzzle function in advance, an adversary knows the algorithm for constructing dynamic puzzles, and is able to "reverse-engineer" the dynamic puzzle $C1_x$ to know the puzzle function $P(\cdot)$ several minutes later after receiving the dynamic puzzle. 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, then it will find the puzzle solution y directly on the host CPU, and send the response (x, y) to the server.

A. *Framework of Dynamic Puzzle:*

In order to defeat the GPU-inflated DoS attack we extend data puzzle to dynamic puzzle as shown in Fig. 1. At the server, the dynamic 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 assembling code blocks extracted from the warehouse; and obfuscating the puzzle $C0_x$ for high security puzzle $C1_x$.

B. *Code Block Warehouse Construction:*

The code block warehouse W stores compiled instruction blocks {bi }, e.g., in Java bytecode, 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 dynamic puzzle generation.

C. *Dynamic Puzzle Generation:*

In order to construct a dynamic puzzle, the server has to execute three modules: puzzle core generation, puzzle challenge generation, dynamic puzzle encrypting/obfuscating, as shown in Figure 1.

1) ***Puzzle Core Generation***: From the code block warehouse, the server first chooses n code blocks based on hash functions and a secret, e.g., the jth instruction block *bij* , where $ij = H_1(y, j)$, and $y = H_2(key, sn)$, with one-way

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

functions $H_1(\cdot)$ and $H_2(\cdot)$, *key* is the server's secret, and *sn* is a nonce or timestamp [1]. All the chosen blocks are added into a puzzle core, denoted as $C(\cdot)$ = (bi1; bi2; · · · ; bin).

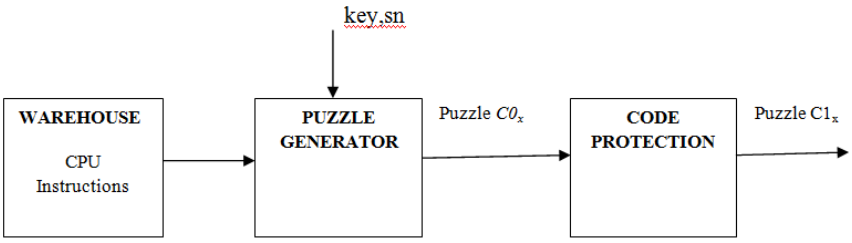

Figure 1. Diagram of dynamic puzzle generated with secret key and nonce sn.

2) ***Puzzle Challenge Generation***: Given some auxiliary input messages such as IP addresses, and in-line constants, the server calculates a message m from public data such as their IP addresses, port numbers and cookies, and produces a challenge x = C(y,m), similar to encrypting plaintext m with key y to produce cipher text x [1]. As the attacker does not know the puzzle core $C(\cdot)$ (or equivalently the puzzle function $P(\cdot)$) in advance, it cannot solve the puzzle $C0_x$ in real time.

3) ***Code Protection***: Although code obfuscation is not satisfactory in long-term software defense against hacking, it is suitable for fortifying dynamic puzzles which demand a protection period of several seconds only [1]. A dynamic puzzle consists of instructions, and each instruction has a form (opCode, [operands]), where opCode indicates which operation (e.g., addition, shift, jump) is, while the operands, varying with opCode, are the parameters (e.g., target address of jump instruction) to complete the operations. As an obfuscation technology, code encryption technology treats software code as data string and encrypts both operand and opCode [1].

D. ***Blocking Clients:***

In all, there are two-layer encryptions. The outer layer is used to encrypt the dynamic puzzle C0x, and the inner layer uses the puzzle software to encrypt the challenge as data puzzle does. Therefore, after receiving C1x, the client has to try ỹ. If and only if ỹ = y, the original dynamic puzzle C0x can be recovered and further used to solve the challenge. Further if the client sends the wrong solution to the server to overload with the false solution to the puzzle then the server adds the clients IP address to its database after verifying the solution sent from the client. The database holds the IP address of the server clients who has sent the wrong solution to the puzzle. When the next time the block client sends the request to the server, the server initial will look into its database for the client's IP address to check whether the client is blocked or not, if the client is not blocked then the server will generate the puzzle and send to it, else it won't send the dynamic puzzle and reject the request of the client.

IV. PSEUDO CODE

Once a dynamic puzzle $C1_x$ is created at the server side and compiled into the Java class file $C1_x$.class, it will be delivered to the client who requests over an insecure channel such as Internet, and run at the client's side. Applet is a suitable delivery means because it can be run in browsers on many platforms such as Windows, Unix, Mac and Linux [3], despite not applicable to some mobile browsers without jail breaking the operating system such as iOS [2].

```
1: <APPLET CODE=‘‘init.class’’
ARCHIVE = ‘‘ini t.class, C1x.class’’
WIDTH=‘‘200’’ HEIGHT=‘‘40’’>
2: </APPLET>
```

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

Usually, an Applet is embedded into an HTML page which is embedded with an archive including the dynamic puzzle class $C1_x$.class and a Java class init.class for activating the puzzle software $C1_x$.class

```
1: Read the C1x.class
2: Repeat
3:   Randomly choose a small ỹ
4:   Decrypt C1x.class with key ỹ into
       class C̃0x.class
5:   Load class C̃0x.class
6:   Invoke C̃0x.class to obtain m̃
       and further x̃ = C̃0(ỹ, m̃)
7: until x̃ = x
8: Output (x̃, ỹ)
```

Figure 2. init.class structure for reloading puzzle class on JVM.

The instructions in $C1_x$.class cannot be directly executed at client's JVM because the dynamic puzzle instructions have to be decryped and then replaced with the decrypted one on the fly. However, a Java class cannot call the new instructions generated by itself. Nonetheless, it is legal in JVM to replace an entire class by reloading a new/recovered version. To this end, the server will generate another class file init.class as in Figure. 2 for managing the puzzle class C1x.class. At the client side, init.class is used to decrypt C1x.class into a temporary class C̃0x.class and reload the class C̃0x.class for one solution trial.

V. SIMULATION RESULTS

In the experiment, an Apache-Tomcat Server 7.0.30 is started to response to client requests on Dell Precision T3600 (Intel Xeon CPU E5-1607, 3.0GHZ, RAM 8GB) installed with Ubuntu 14.04.1 LTS 64 bit. When a client sends a request to the server, a servlet will create the dynamic puzzle [1]. Microsoft Internet Explorer, installed with Java VM 1.7.0_67, is run over Dell T3600.

Experiment Results

SSL/TLS protocol is the most popular on-line transaction protocol, and an SSL/TLS server performs an expensive RSA decryption operation for each client connection request, thus it is vulnerable to DoS attack [1]. Our objective is to protect SSL/TLS server with dynamic puzzle against computational DoS attacks. As a complete SSL/TLS protocol includes many rounds, RSA decryption step to evaluate the defence effectiveness in terms of the server's time cost for simplicity is better [1]. Assume the time to perform one RSA decryption be t0, and the time to generate and verify one dynamic puzzle be ts (Note that t0 > ts , otherwise, dynamic puzzle is useless). Suppose the number of attacker's requests be na, and the number of genuine client requests be nc, the server's computational time required for replying all the requests is $\tau 1 = (na + nc) \times t0$ if there is no dynamic puzzle; otherwise, $\tau 2 = (na + nc) \times ts + nc \times t0$ given that the adversary does not return valid solutions to the puzzles. Thus, dynamic puzzle defence is effective if

$\tau 1 \geqslant \tau 2$, i.e., $na \geq \frac{ts}{t0-ts} nc$ (1)

That is, when the number of malicious requests na is greater than $\frac{ts}{t0-ts} nc$, the genuine clients spend less time in waiting for the services. Hence, a good strategy is to initiate the dynamic puzzle defence if the number of requests is beyond a threshold, otherwise, no defence is required because quality of service is satisfactory for all clients. To demonstrate the effectiveness of dynamic puzzle, let's compare the cost of the participants.

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

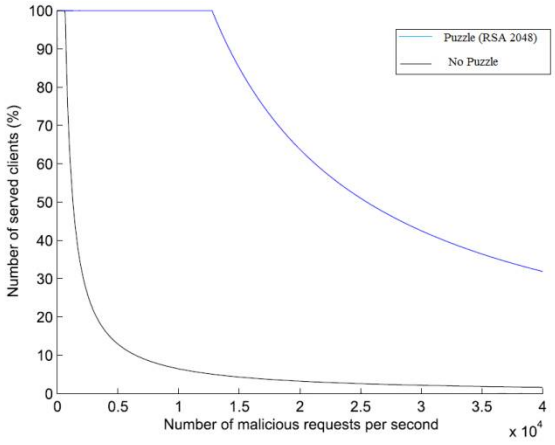

Figure 3. Service capability comparison of server with/without dynamic puzzle for RSA2048.

1) Server Cost: If the server-client system adopts dynamic puzzle, the CPU time spent in the server is

- time t1 for preparing the initial puzzle C0x ;
- time t2 for converting C0x into dynamic puzzle C1x ;
- time t3 for puzzle package generation;
- time t4 for verifying the client answer.

Thus the server time ts = t1+t2+t3+t4 ≈ t1+t2+t3, where the approximation holds because the puzzle verification time t4 is very small. In our experiments, t1 = 1.7 μ s, t2 = 1.5 μ s and t3 = 1.2 μ s on average, or ts ≈ t1 + t2 + t3 = 4.4 μ s in total. On the other hand, it will take the server t0 = 1476μs for performing one RSA2048 decryption with OpenSSL package 1.0.1f [1]. Therefore ts ≪ t0. It means that the dynamic puzzle is a practical defence. Figure. 3 illustrates that the dynamic puzzle can increase the service quality significantly in terms of the percentage of served customers.

VI. Conclusion and Future Work

In this paper, dynamic puzzle scheme is proposed for defeating DoS/DDoS attacks. It offers two layers of protection using two times encryption which provide data confidentiality and code security. Hence, it is different from the security requirement from the conventional cipher which demands long-term confidentiality. Further clients who send the wrong solution to the puzzle are blocked from further sending the requests.

Since the dynamic puzzle may be built upon a data puzzle, it can be integrated with any existing server-side data puzzle scheme, and easily deployed as the present client puzzle schemes do. Further clients using the cloud platform should be identified because they can enhance their puzzle solving capacity.

References

[1] 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, 168-177, January 2015.

[2] J. Bailey. “How to Install Java on an iPhone”, eHow Contributor. [Online]. Available: http://www.ehow.com/how_5659673_install-java-iphone.html#ixzz24jIAyKiM, Oct. 28, 2014.

[3] S. Wang. “How to Create an Applet & C++”.[Online].Available:http://www.ehow.com/how_12074039_createApplet-c.html#ixzz24Lsk0OJQ, Sep.18,2011.

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

[5] T. J. McNevin, J.-M. Park, and R. Marchany, “pTCP: A client puzzle protocol for defending against resource exhaustion denial of service attacks,” Virginia Tech Univ., Dept. Elect. Comput. Eng., Blacksburg, VA, USA, Tech. Rep. TR-ECE-04-10, Oct. 2004.

[6] C. Douligeris and A. Mitrokotsa, “DDoS attacks and defense mechanisms: Classification and state-of-the-art,” Comput. Netw., vol. 44, no. 5, pp. 643–666, 2004.

[7] A. Juels and J. Brainard, “Client puzzles: A cryptographic countermeasure against connection depletion attacks,” in Proc. Netw. Distrib. Syst. Secur. Symp., pp. 151–165, 1999.

ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

BIOGRAPHY

Somshekar Hannikeri is a student of Master of Technology in Computer Network Engineering in the Information Science and Engineering Department, The Oxford College of Engineering, Visvesvaraya Technological University. He received Bachelor of Engineering (B.E) degree in 2014 from Visvesvaraya Technological University, Belgavi, Karnataka, India.

Ram Sankar. G is a assistant professor in the Information Science and Engineering Department, The Oxford College of Engineering, Visvesvaraya Technological University. He is pursing PhD in the area of Soft Computing in Bio Medical. He received Master of Engineering (M.E) degree in Computer Science and Engineering from Anna University, Chennai, Tamil Nadu, India. His research interests are Computer Networks, Cyber Security and Forensic, and Soft Computing.