The Role of Software Puzzle in Overcoming the Denial of Service Attacks

Advancement of Client Puzzle

Ashish Chinnappa T M
Dept. of CSE ,AMCEC

Aishwarya K
Dept. of CSE,AMCEC

Dheeraj R Makam
Dept. of CSE,AMCEC

Ms. Nirmala S, Associ. Prof
Dept. of CSE,AMCEC

***Abstract*—some of the major threats to cyber security are Denial of service and Distributed Denial of service attacks. In order to overcome these attacks, the present system incorporates the client puzzle which mandates the client to perform various complicated operations before being granted requested services from the server. However, an attacker makes use of various fast puzzle solving software and/or built-in graphics processing unit (GPU) hardware to suppress the potential of client puzzles. Also, the puzzle algorithms generated in advance by the client puzzle are not dynamic and hence an attacker can prepare an implementation to solve the puzzle in advance. In this paper, we study how to prevent these attacks by introducing an advancement of the existing client puzzle called as Software puzzle. In this scheme, once the server receives the request from the client, a puzzle is generated randomly. Thus, an attacker has to put in considerable effort in translating the CPU puzzle software to its functionally equivalent Graphical Processing Unit such that the translations are complex and cannot be done in real time.**

Keywords—Software puzzle, code obfuscation, GPU programming, distributed denial of service (DDoS) and denial of service (DoS).

I. INTRODUCTION

Denial of service is a type of attack where the server is being flooded with a number of requests by the attacker in order to keep the server busy and thereby denying the services requested by the clients. For example, a malicious client sends a large number of unwanted requests to a server. Hence the server spends enormous amount of CPU time in completing the SSL handshakes due to which the server may not have sufficient resources left to handle service requests. To maintain the confidentiality and integrity of the puzzle generated by the server as well as the solution for the puzzle given by the client are encrypted and decrypted respectively using various encryption algorithms like RSA decryption.
Let γ denote the ratio of resource consumption by the client to the resource consumed by the server. This ratio should be greater in order to meet the economic feasibility of the software puzzle scheme. Consider P to be a puzzle function, x to be a randomly chosen puzzle challenge by the server and sends this x to the client. The client solves this puzzle and sends the response (x,y) back to the server for the verification. The server then checks the solution for the correctness and if the solution does not confirms x=P(y) [18], then the client is not being served. The time spent by the client to solve the given puzzle is given by t_c and the time spent by the server to generate the puzzle and verify the solution is given by t_s. Hence $t_c >> t_s$ in order to make use of the server time efficiently and stop the DoS attack by not allowing the attackers to solve many puzzles. But, the attackers can reply to the server with an unsolved puzzle solution $y^{\sim}$ which increases the verification time of the server. Thus, the ratio γ decreases dramatically which results in the decrease of the efficiency of the client puzzle.

The server assumes that the puzzle sent to the client will be solved by it only using CPU. However, it is not so. The client as an attacker can utilize its freely available GPU to solve the puzzle and thereby decreasing the computational cost ratio γ. For example, an attacker may amortize one puzzle-solving task to hundreds of GPU cores if the client puzzle function is parallelizable or the attacker may simultaneously send to the server many requests and ask every GPU core to solve one received puzzle challenge independently if the puzzle function is non-parallelizable. This parallelism strategy can dramatically reduce the total puzzle-solving time and hence increase the attack efficiency. Another disadvantage of the client puzzle is that the puzzle function P(.) can be rewritten in real time by the attacker if it cannot be solved. 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;[12] and (2) an extra real-time environment shall be installed in order to run GPU kernel.

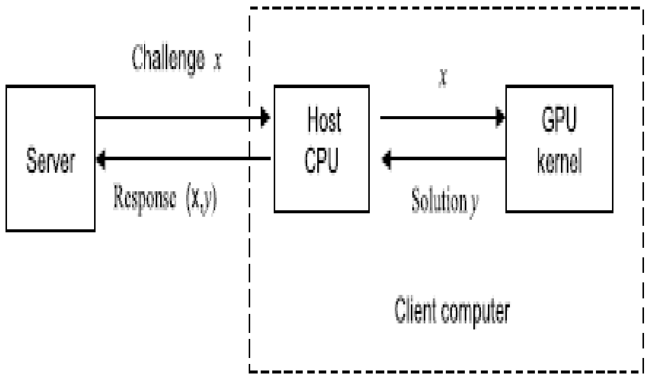

Fig 1.GPU inflated DoS attack against data puzzle

II. SOFTWARE PUZZLE

Focussing on the architectural difference between CPU and GPU, we present advancement to the client puzzle called the software puzzle which defends against the DoS and DDoS attacks. Unlike the client puzzle scheme, the software puzzle scheme generates the puzzle function P(.) dynamically in the form of a software core C upon receiving a client's request. Specifically, by extending DCG technology which produces machine instructions at runtime, the proposed scheme randomly chooses a set of basic functions, assembles them together into the puzzle core C, constructs a software puzzle C0x with the puzzle core C and a random challenge x. If the server aims to defeat high-level attackers who are able to reverse-engineer software [9], it will obfuscate C0x into an enhanced software puzzle. After receiving the software puzzle sent from the server upon request, the client tries to solve this puzzle on its host CPU and sends back the solution of the puzzle to the server. If the client is suspected to be an attacker, he sends the puzzle received from the server to its host GPU. Thereby, overwhelming the server with unwanted requests through its host CPU. However in this case, translating the puzzle function received by the host CPU into equivalent GPU instructions which is highly complex and time consuming. These translations cannot be done well in advance because the software puzzle is generated dynamically and randomly.

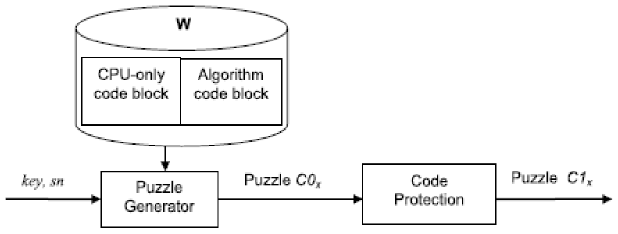

Fig. 2 Diagram of software puzzle generated with secret key.

A. Notations

For ease of reference, important notations used throughout
the paper are listed below.
x: A challenge chosen by server.
m: A message collected from environment.
y: A solution to the puzzle challenge x
(˜ x, ˜ y): A puzzle response returned from client.
P(·): Puzzle algorithm such that x =P(y,m).
C: Puzzle core which is the software implementation of P(·).
C0x: Puzzle which embeds the information of x into C.
C1x: Obfuscated C0x.

That is to say, unlike a data puzzle challenge which includes a challenge data only, a software puzzle challenge includes a dynamically generated software C(·) which including a data puzzle function as a component. Although a software puzzle scheme does not publish the puzzle function in advance, it also follows the Kerckhoffs's Principle because an adversary knows the algorithm for constructing software puzzles, and is able to "reverse-engineer" the software puzzle C1x to know the puzzle function P(·) several minutes later after receiving the software puzzle.

B. Basic GPU-inflated DoS Attack

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[12], 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.

C. Framework of software puzzle

In order to defeat the GPU-inflated DoS attack, we extend data puzzle to software puzzle as shown in Fig. 2. 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 C0x by randomly assembling code blocks extracted from the warehouse; and obfuscating the puzzle C0x for high security puzzle C1x.

D. Code Block Warehouse Construction

The code block warehouse W stores compiled instruction blocks {bi}, e.g., in Java byte code, or C binary code [15]. 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.

• 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 [21]. 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).

III. METHODOLOGY

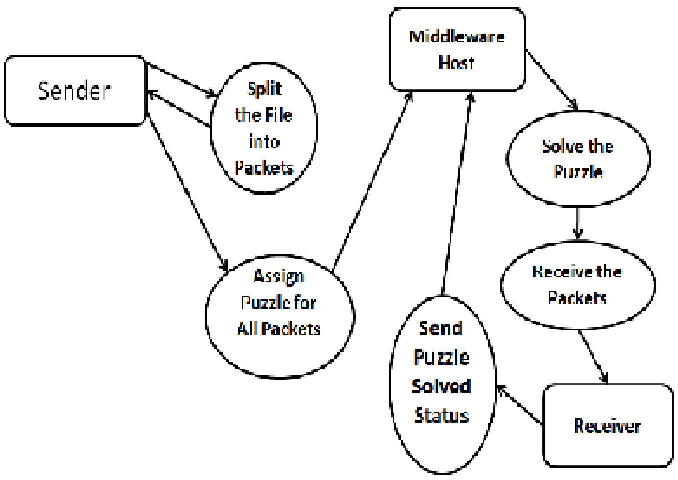
Sender
Split the File into Packets
Middleware Host
Solve the Puzzle
Receive the Packets
Assign Puzzle for All Packets
Send Puzzle Solved Status
Receiver

Fig. 3 Design Modules

A. *Puzzle Generator*

The Puzzle Generator browses the file, splits file into packets, assign puzzles and then sends Packets to the Receiver via Middle ware Host to corresponding end users by providing the IP address.

B. *Middleware Host*

The middleware host is introduced here to redirect packets and puzzle details to corresponding end users. It will maintain all Puzzle flatted user and also the DOS Attackers who misbehaved to sole the corresponding puzzle by giving wrong values.

C. *Remote Receiver (End User)*

The End user can receive the data file and verify the puzzle by entering the puzzle value which is assigned by the Puzzle Generator. If the puzzle value is true then he is considered as a Flatted user or considered as a DOS Attacker. The attacker and flatted user details will send to the Middleware Host instantly.

D. *Attacker*

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 this system, 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 [13].

IV. SOFTWARE PUZZLE PACKING

Once a software puzzle C1x is created at the server side and compiled into the Java class file C1x.class, it will be delivered to the client who requests for services 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, despite not applicable to some mobile browsers without jail breaking the operating system such as iOS. Usually, an Applet is embedded into an HTML page which is embedded with an archive including the software puzzle class C1x.class and a Java class init. Class for activating the puzzle software C1x.class

```
1: <APPLET CODE=‘‘init.class’’ ARCHIVE = ‘‘init.class, C1x.class’’ WIDTH=‘‘200’’ HEIGHT=‘‘40’’>
2: </APPLET>
```

However, not all Applets can be run at the client's browser with the default access policy such that the design for software puzzle varies with the browser's configurations at the client side. In the following, we describe two options for packing software puzzle based on the configuration at the client side [5].

```
1: Read the C1x.class
2: Repeat
3: Randomly choose a small y˜
4: Decrypt C1x.class with key y˜ into class C0x.class
5: Load class C0x.class
6: Invoke C0x.class to obtain m˜ and future x˜=C0˜(y˜, m˜)
7: Until x˜ =x
8: output (x˜, y˜)
```

(Init. class structure for reloading puzzle class on JVM. If a correct solution y is found, C0x.class shall be the same as the original puzzle C0x.class, where $z = x \oplus y$ is calculated in advanced and hard-coded into at the server side)

REFERENCES

[1] J. Larimer. (Oct. 28, 2014). Pushdo SSL DDoS Attacks. [Online]. Available: http://www.iss.net/threats/pushdoSSLDDoS.html
[2] 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
[3] A. Juels and J. Brainard, “Client puzzles: A cryptographic countermeasure against connection depletion attacks,” in Proc. Netw. Distrib. Syst. Secur. Symp., 1999, pp. 151–165.
[4] 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
[5] 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. [Online]. Available: http://hdl.handle.net/2142/17372
[6] 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.
[7] Y. I. Jerschow and M. Mauve, “Non-parallelizable and non-interactive client puzzles from modular square roots,” in Proc. Int. Conf. Availability, Rel. Secur., Aug. 2011, pp. 135–142.
[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. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.5709
[9] W.-C. Feng and E. Kaiser, “The case for public work,” in Proc. IEEE Global Internet Symp., May 2007, pp. 43–48.
[10] D. Keppel, S. J. Eggers, and R. R. Henry, “A case for runtime code generation,” Dept. Comput. Sci. Eng., Univ. Washington, Seattle, WA, USA, Tech. Rep. CSE-91-11-04, 1991.
[11] E. Kaiser and W.-C. Feng, “mod_kaPoW: Mitigating DoS with transparent proof-of-work,” in Proc. ACM CoNEXT Conf., 2007, p. 74.

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

[13] 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.

[14] M. Jakobsson and A. Juels, “Proofs of work and bread pudding protocols,” in Proc. IFIP TC6/TC11 Joint Working Conf. Secure Inf. Netw., Commun. Multimedia Secur., 1999, pp. 258–272.

[15] D. Kahn, The Codebreakers: The Story of Secret Writing, 2nd ed. New York, NY, USA: Scribners, 1996, p. 235.

[16] 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.

[17]] B. Barak et al., “On the (Im)possibility of obfuscating programs,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 2139. Berlin, Germany: Springer-Verlag, 2001, pp. 1–18.

[18] H.-Y. Tsai, Y.-L. Huang, and D. Wagner, “A graph approach to quantitative analysis of control-flow obfuscating transformations,” IEEE Trans. Inf. Forensics Security, vol. 4, no. 2, pp. 257–267, Jun. 2009.

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

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

[21] J. Ansel et al., “Language-independent sandboxing of just-in-time compilation and self-modifying code,” in Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement., 2011, pp. 355–366.

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

[23] T. Lindholm and F. Yellin, The Java Virtual Machine Specification, 2nd ed. Reading, MA, USA: Addison-Wesley, 1999, ch. 9. [Online]. Available: http://docs.oracle.com/javase/specs/jvms/se5.0/html/VMSpecTOC.doc.html

[24] J. Black and P. Rogaway, “Ciphers with arbitrary finite domains,” in Topics in Cryptology (Lecture Notes in Computer Science), vol. 2271. Berlin, Germany: Springer-Verlag, 2002, pp. 114–130.