International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 02 | Feb-2016 www.irjet.net p-ISSN: 2395-0072

MITIGATION OF DISTRIBUTED DENIAL OF SERVICE ATTACKS BY USING SOFTWARE PUZZLE

Mrs. Suganya K, Mr. Britto Dennis J, Ms. Hamna Farhan P C

PG Scholar, Dept. of.CSE(with specialization in networks)., Dhanalakshmi Srinivasan Engineering College,
Professor, Dept. of Information Technology., Dhanalakshmi Srinivasan Engineering College,Tamilnadu,India
PG Scholar, Dept. of.CSE (with specialization in networks)., Dhanalakshmi Srinivasan Engineering College, Tamilnadu,India

---

Abstract - *Denial of Service (DoS) attack and Distributed Denial of Service (DDoS) attack 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. Client puzzle is a well-known countermeasure, which demands a client to perform computationally expensive operations before being granted services from a server. 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. A new puzzle scheme called software puzzle is introduced to prevent DoS/DDoS attackers from inflating their puzzle-solving capabilities. 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. Software puzzle aims at an attacker is unable to prepare an implementation to solve the puzzle in advance and 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.*

Key Words: *DDoS, client puzzle, software puzzle,*

GPU programming.

1. INTRODUCTION

Internet based technologies have revolutionized the banking industry as well as way people interact with financial institution and one another financially. However, it has raised new questions and dimensions for securing data of the financial institutions as well as the end-users. 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. 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. 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.

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. In the puzzle-solving and verification steps, the client returns a puzzle response (x, y), and 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.*, $\gamma \_ 1$, an attacker cannot start DoS attack efficiently by solving many puzzles. Alternatively, the attacker can merely reply to the server with an arbitrary number $\tilde{y}$ so as to exhaust the server's time for verification. In this case, although $\gamma < 1$ such that defense effect of client puzzle is weakened, the

server time *ts* is still much smaller than the service preparation time (*e.g.*, RSA decryption) or service time (*e.g.*, database process) as the returned answer will be rejected at a high probability. Of course, optimizing the puzzle verification mechanism is very important and doing so will undoubtedly improve the server's performance.

However, an attacker can easily utilize the "free" GPUs or integrated CPU-GPU to inflate his computational capacity. This renders the existing client puzzle schemes ineffective due to the significantly decreased computational cost ratio. 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. As the present browsers such as Microsoft Internet Explorer and Firefox do not explicitly support client puzzle schemes, Kaiser and Feng [2] developed a web-based client puzzle scheme which focuses on transparency and backwards compatibility for incremental deployment. The scheme dynamically embeds client-specific challenges in webpages, transparently delivers server challenges and client responses. However, this scheme is vulnerable to DoS attackers who can implement the puzzle function in real-time. Technically, an attacker can rewrite the puzzle function $P(\cdot)$ with a native language such as C/C++ such that the cost of an attacker is much smaller than that the server expects.3 Even worse, a GPU-inflated DoS attacker can realize the fast software implementation on the many-core GPU hardware and run the software in all the GPU cores simultaneously such that it is easy to defeat the web-based client puzzle scheme

Many researches focus on reduce the effect of DoS/DDoS attacks and add a difficulty to solve the puzzle by an attacker. Software puzzle is proposed to mitigate the effects of DoS and DDoS attacks. Software puzzle aims to deter an adversary from understanding or translating the implementation of a random puzzle function. That is to say, unlike a data puzzle challenge which includes a challenge data only, a software puzzle challenge includes dynamically generated software, which including a data puzzle function as a component. Although a software puzzle scheme does not publish the puzzle function in advance, because an adversary knows the algorithm for constructing software puzzles.

2. SYSTEM MODEL

In this section we consider the existing system design and the proposed system.

2.1 Existing System

Client puzzle schemes which publish a puzzle function in advance, the software puzzle scheme dynamically generates the puzzle function $P(\cdot)$ 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 [10], 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.

Client puzzle schemes assume that the malicious client solves the puzzle using legacy CPU resource only. This assumption is not always true. Presently, the many-core GPU (Graphic Processing Unit) component is almost a standard configuration in modern desktop computers(*e.g.*, ATI FirePro V3750 in Dell T3500), laptop computer (*e.g.*, nVidia Quadro FX 880M in Lenovo Thinkpad W510), and even smartphones (*e.g.*, PowerVR SGX540 in Samsung I9008 Galaxy*TM* S). Therefore, an attacker can easily utilize the "free" GPUs or integrated CPU-GPU to inflate his computational capacity [5]. This renders the existing client puzzle schemes ineffective due to the significantly decreased 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 (*e.g.*, the hash reversal puzzle), 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 (*e.g.* modular square root puzzle [7] and Time-lock puzzle [8]). This parallelism strategy can dramatically reduce the total puzzle-solving time, and hence increase the attack efficiency. Green et al. [6] examined various GPU-inflated DoS attacks, and showed that attackers can use GPUs to inflate their ability to solve typical reversal based puzzles by a factor of more than 600. Moreover, in order to defeat GPU-inflated DoS attack to client puzzles, they proposed to track the individual client behavior through client's IP address [9].

Problems Identified:

- A client has to spend a certain amount of time tc in solving the puzzle (i.e., finding the puzzle solution y)
- The server has to spend time ts in generating the puzzle challenge x and verifying the puzzle solution y

- The existing client puzzle schemes assume that the malicious client solves the puzzle using legacy CPU resource only

2.2 Proposed System

A new type of client puzzle, called software puzzle is defend against GPU-inflated DoS and DDoS attacks. Unlike the existing client puzzle schemes which publish a puzzle function in advance, the software puzzle scheme dynamically generates the puzzle function $P(\cdot)$ in the form of a software core C upon receiving a client's request. 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, it will obfuscate C0x into an enhanced software puzzle. After receiving the software puzzle sent from the server, a client tries to solve the software puzzle on the host CPU, and replies to the server, as the conventional client puzzle scheme does. 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 software puzzle into its functionally equivalent GPU version because GPU and CPU have totally different instruction sets for different application.

Benefits:

- Software puzzle prevent GPU from being used in the puzzle-solving process based on different instruction sets and real-time environments
- Easily deployed as the present client puzzle schemes do.
- Random puzzle with random algorithm at each time.

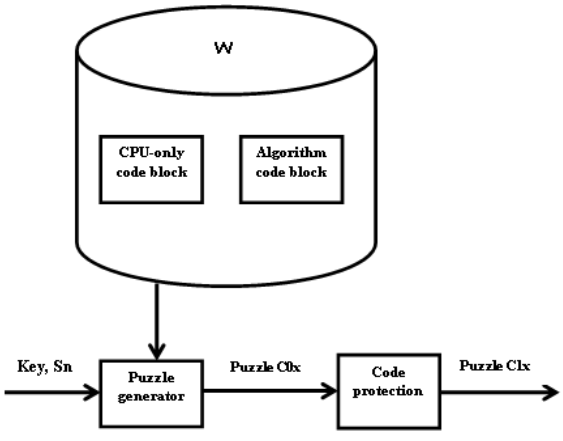

Fig 1: software puzzle generated with secret key and nonce sn.

3. CODE BLOCK WAREHOUSE CONSTRUCTION

The code block warehouse **W** stores compiled instruction blocks. 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.

3.1 CPU-Only Instruction Block

Unlike CPU, GPU is designed for the predictable graphic processing such as matrix operations, not generic logic processing. As branching operations (e.g., try-catch-finally, goto) are inherently non-predictable and are non-parallelable, executing them in GPU is slow such that the major merit of GPU cannot be exploited by the attacker**.**

TABLE I
EXAMPLE CPU-ONLY INSTRUCTIONS

Instruction	Difference exploited
Read local cookie	GPU cannot directly read CPU storage
Allocate large memory	GPU has much smaller memory than CPU
Try-catch	GPU does not support except handling
Goto (address)	GPU does not support branch
Create new class	GPU does not support dynamic class

3.2 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.

4. SOFTWARE PUZZLE GENERATION

In order to construct a software puzzle, the server has to execute three modules: puzzle core generation, puzzle challenge generation, software puzzle encrypting/obfuscating.

4.1 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 j th instruction block $bi\,j$, where $i\,j = H1(y, j\,)$, and $y = H2(key, sn)$, with one-way functions $H1(\cdot)$ and $H2(\cdot)$, key is the server's secret, and sn is a nonce or timestamp. All the chosen blocks are assembled into a puzzle core, denoted as $C(\cdot) = (bi1\ ;\ bi2\ ;\ \cdots\ ;\ bin\ )$.

4.2 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. As the attacker does not know the puzzle core $C(\cdot)$ (or equivalently the puzzle function $P(\cdot)$) in advance, it cannot exploit GPU to solve the puzzle $C0x$ in real time using the basic GPU-inflated DoS attack addressed. Nonetheless, if the puzzle is merely constructed as above, it is possible for an attacker to generate the GPU kernel by mapping the CPU instructions in $C0_x$ to the GPU instructions one by one, i.e., to automatically translate the CPU software puzzle $C0_x$ into its functionally equivalent GPU version.

4.3 Code Protection

Code obfuscation is able to thwart the above translation threat to some extent. Though there are no generic obfuscation techniques which can prevent a patient and advanced hacker from understanding a program in theory [3], results in [4] show that obfuscation does increase the cost of reverse-engineering. Thus, although code obfuscation may be not satisfactory in long-term software defense against hacking, it is suitable for fortifying software puzzles which demand a protection period of several seconds only.

A software puzzle consists of instructs, 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 a popular obfuscation technology, code encryption technology treats software code as data string and encrypts both *operand* and *opCode*. Concretely, given the code $C0x$, the server generates an encrypted puzzle $C1x = E(y, C0x)$, where $E(\cdot)$ is a cipher such as AES, and y is used as the encryption key. In practice, there are many commercial code obfuscation tools for C/C++ software such as VMprotect (*http://vmpsoft.com/*) which can be used to protect the software puzzle from hacking.

5. SECURITY ANALYSIS

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.

Employing Host Simulator on GPU: If an attacker is able to run a CPU simulator over GPU environment, the software puzzle can be executed on GPU directly.

5.1 Deobfuscating Software Code

Generally, dynamic translation can accelerate the attacking speed, but it is not very helpful to the GPU-inflated DoS attacker because

- Dynamic translation is usually a human-machine interactive process. If human interference is required, the DoS attack is very ineffective.

In order to carry on the dynamic translation, the attacker needs a simulation environment for "debugging" the software puzzle. In the translation process, the decryption key $\tilde{y}$ has to be tested by brute force. Because it is impossible to decide whether a tested key is right based on the recovered *opCode* value; the attacker has to run the puzzle $\_C0x$ for every key test to make the decision.

- If the simulation environment is run on CPU host, the host cannot generate the GPU kernel until the solution is found. Therefore, this translation time is longer than the time used to directly solve software puzzle by CPU host. In other words, the GPU is useless for accelerating puzzle-solving in this case.
- If the simulator is run on GPU, the attacker has to face the troubles stated in Subsection V-A besides the trouble existing in the above CPU simulation environment.

Once the translated code has one error, the attacker fails to recover the software puzzle $C0x$ to find the correct response such that he cannot launch DoS attack. Therefore, it is not easy for an attacker to develop a GPU

kernel for solving the original software puzzle by deobfuscating/analyzing software puzzle.

5.2 Abusing Access Priority

All the client puzzle schemes assume that there is no secure channel between the client and the server until puzzle verification completion. Otherwise, the client puzzle scheme is redundant. Thus, an attacker can intercept all the traffic between the client and the server, and start man-in-the-middle attack, say, sending malicious software puzzles to the client browser so as to launch attacks to the clients. However, an access policy should be defined so as to enable the software puzzle to call some special class generation functions. Hence, the attacker may have extra right to create new classes to make troubles to the clients.

Luckily, this "flaw" does not really incur any new threat to the client host. As any new class created from the attacker has the same priority as the original one, *i.e.*, the same as normal class except class generation permission, it cannot access any other extra resources in the host platform. Nonetheless, this class generation permission enables the attacker to deplete the memory resource of the local host by creating infinite number of classes. But this memory DoS attack to local host also exists in the "legal" Applet which requests for a large amount of memory. Hence, the adversary is unable to incur new threat to the host by abusing the extra priority.

6. CONCLUSION

Software puzzle scheme is proposed for defeating GPU-inflated DoS attack. It adopts software protection technologies to ensure challenge data confidentiality and code security for an appropriate time period. Hence, it has different security requirement from the conventional cipher which demands long-term confidentiality only, and code protection which focuses on long-term robustness against reverse-engineering only. Since the software 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.

REFERENCES

[1] Jun. 2009.J. Larimer. (Oct. 28, 2014). *Pushdo SSL DDoS Attacks*. [Online] Available : http:// www .iss.net/threats/pushdoSSLDDoS.html

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

[3] 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.

[4] 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

[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.