Enhancement of KaPoW Plugin to Defend Against DDoS Attacks

Farah Samir Barakat
IT Department
Faculty of Computers and Information, Cairo University
Cairo, Egypt

A .Prof. Amira Kotb
IT Department
Faculty of Computers and Information, Cairo University
Cairo, Egypt

***Abstract*—DDoS attack is one of the hardest attacks to detect and mitigate in the computer world. This paper introduces two quantitative models, which use the client puzzling to detect and thwart application DDoS attacks. We simulated the models to use the probabilistic metrics to penalize the malicious users and prevent them from launching a DDoS attack while offering a stable environment to the normal users and decreasing the number of false positives and false negatives.**

Keywords—Application Security; Client Puzzling; DDoS; Metrics; PHP; Puzzle

I. Introduction

Distributed Denial of Service (DDoS) attacks is one of the most rapidly increasing threats to the Internet eco-system. It has been increasing almost exponentially leaving the servers always wanting more bandwidth. Nowadays, DDoS attacks may be more than 100Gbps which is 10 times the size of most internet backbone pipes.

DDoS is DoS taken to a whole new level using diversification, obfuscation and distribution of the attack origin. DDoS is launched using many computers on one or more victims to prevent the legitimate users from accessing the network resources [1,2].

Over the past years, many defenses techniques were introduced to defend against DDoS attacks: Whitelists [3], Blacklists [4], VIP lists [5], Captcha [6]. But they all had some disadvantages e.g. false positives and false negatives.

In this paper, we applied the Client Puzzling approach to defend against the DDoS attack. It’s a Proof of Work (PoW) [7] technique where the client proves that it has done some work, by solving medium to hard puzzles, in return to get the needed resources from the server and to prove its legitimacy [8,9].

II. Related Work

A. *Client Puzzling*

Client Puzzling is a protection technique, characterized by its capability to be integrated into any web application with minimal alterations to the infrastructure and software components. Dwork and Naor were the first people to suggest the use of client puzzles to limit the junk email [10,11]. But unfortunately, client puzzling has its shortcomings for adversaries with parallelization capabilities, or legitimate flash-crowds [8].

B. *Puzzles Difficulty Calculations*

The puzzle difficulty can be determined based on the server load, the client behavior or just fixed difficulty [8]. In cases where the difficulty is based on the server load, the puzzle difficulty increases as the server runs out of resources regardless of their maliciousness. That’s why it is the worst for the legitimate clients. It’s better to determine the puzzle difficulty based on the client’s behavior to penalize the attackers by giving them harder puzzles than the normal clients. Yet, this will require the server to track the client’s behavior by using client identifying information, such as the client’s IP address or the assigned nonce tokens. In the fixed difficulty all the clients are not required to solve a puzzle. However, when the server resources are occupied above a certain threshold, all the clients receive a puzzle with a pre-defined fixed difficulty.

C. *KaPoW*

KaPoW is a PoW based technique, implemented as libraries and can be used by the web applications to enhance the performance of anti-spam techniques such as: Captcha and spam filters [13].

There are two implementations for using KaPoW to protect the web content:

- KaPoW Apache module known as Mod_KaPoW. It is an Apache2 module which is almost transparent for the application. It embeds the puzzling and the solver mechanism in a way that changes the application on-the-fly [14,15].
- KaPoW plugin which is a PHP library that allows the puzzles to be embedded in the HTML tags, solved by JavaScript and verified by a server-side component [16]. Two existing applications for KaPoW plugin are KaPoW webmail filter and KaPoW anti-spam filter.

KaPoW calculates the puzzle difficulty based on multiple metrics. The total score is calculated by summing all the metrics’ scores:

$$\text{Score} = S1 + S2 + \ldots Sn \qquad (1)$$

where *n* is the number of used metrics. The user will receive a puzzle with difficulty (Dc) based on his score. The difficulty is calculated using:

$$Dc = m \times (\text{score})^n \qquad (2)$$

where *m* is an arbitrary empirical constant [16].

D. KaPoW Modules

As any client puzzling system, KaPoW plugin consists of three components: the puzzle issuer and verifier at the server-side and the puzzle solver at the client-side. The issuer generates the puzzle and delivers it to the client. After the client receives the puzzle, the solver generates random solutions to these puzzles until a correct solution is found and sent to the server. Finally, the verifier accepts or rejects the solutions, sent to the server, based on their correctness, legitimacy and freshness [16]. Fig. 1 describes the system architecture of KaPoW Guestbook. The same architecture is followed in the proposed models.

E. KaPoW Guestbook

KaPoW Guestbook [13] is an open source project under GPLv2 License and implemented in PHP. It solves the modified time-lock puzzles using a JavaScript solver which is called via AJAX which allows solving the puzzle in the background.

KaPoW Guestbook can be integrated in any application because of its modularity; which makes it easy to add after the application is already developed; there is no need to make changes in the core modules of the applications.

KaPoW Guestbook's browser side displays the comment form to the user asking him to enter his name, e-mail, comment and IP address. Then the user submits the form, and a new puzzle is requested.

When the server receives a request for a new puzzle, it invokes SpamAssassin to detect if the contents contain any spam data. Then the server checks the blacklists and returns the threat score. The difficulty (Dc) is calculated based on (2). After calculating Dc, the server nonce and the difficulty are returned as a response to the JavaScript. The client tries to solve a hash function using the information sent by the server. Finally, when the answer is found, the client submits it to the server. If the server verifies the answer is correct, it'll accept the new message and will display all the messages, otherwise it will reject the submission and will only display the old messages.

III. PROPOSED MODEL "DDOS_KAPOW"

Client Puzzling has proven its capability and efficiency at defending DoS and spam attacks, that's why we decided to apply it to defend against DDoS attacks.

After examining the KaPoW_Guestbook open source code, some discrepancies were found that prevented the code from running. So, we solved the problem with the SpamAssassin and to save bandwidth, we applied caching on the message content. Also, to make the code run, we replaced the blacklists used by the authors, by "DShield Blacklist" because they didn't exist anymore.

KaPoW Guestbook's original implementation was done using two metrics Spam filter and IP Blacklist to detect the presence of Forum Spam attacks. "DDoS_KaPoW" is an implementation of KaPoW Guestbook, it uses the same architecture but with some enhancements made on the individual modules and the used metrics to adjust them to the setup environment and to defend DDoS attacks instead of spam attacks.

We modified the user interface and the core engine using a Resource Intensive Operation "RIO" (some calculations in the background which makes the post message action does some processing) to simulate the CPU intensive operation. We substituted the spam filter metric by the processor load since it is known as one of the most important factors indicating the presence of a DDoS attack. Since DDoS_KaPoW focuses on the freshness of the client puzzles, we made the nonce Nc random for each request and it is submitted with the answer for verification instead of being constant like in KaPoW Guestbook. The constant Nc in KaPoW Guestbook will allow the attackers to generate multiple requests using the same answer. We added the capability to enable and disable the Client Puzzling which will help us in the evaluation of the models. We also added an internet connectivity check because of the regularly updated services which require an internet connection e.g. blacklist.

Finally, DDoS_KaPoW checks if the user is an attacker by checking the processor load and the IP blacklist. If the processor load is higher than a predefined threshold, the score is increased by 8. If the user's IP address exists in blacklist, the score will increase by 5. At the end, the puzzle difficulty is calculated using (2).

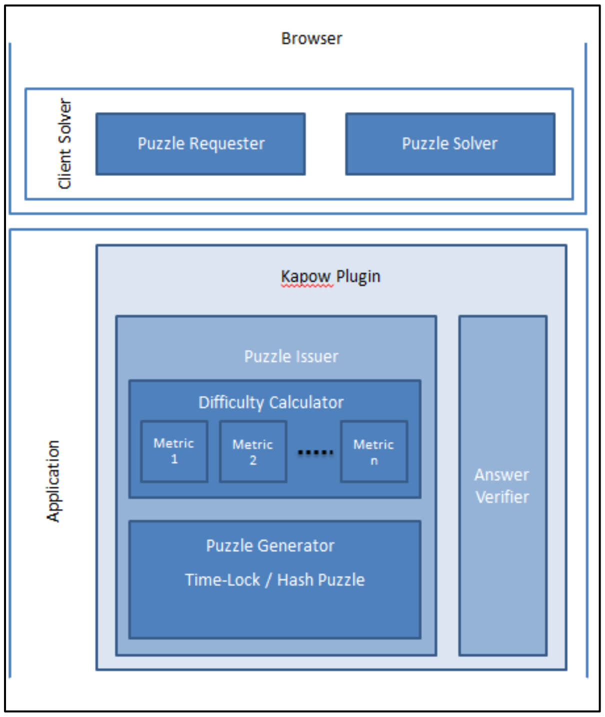

Fig. 1. System Architecture of KaPow Guestbook

IV. PROPOSED MODEL "Z-POW"

This proposed model describes enhancement options to mitigate application DDoS attacks based on the previous work's deficiencies taking into consideration the various dependent and independent variables, the nature of problem at hand and the technological environment limitations. It is called

Zombie Proof of Work or “Z-PoW”. Due to the vulnerabilities found in the old techniques, Z-PoW combines the effectiveness of anti-spam defense and anti-DoS defense to defend against DDoS.

Z-PoW is a mutation of the client puzzling implementation in Mod-KaPoW and KaPoW plugin. It combines the concept of the client’s maliciousness score and the equation needed to calculate a combined score from these metrics taken from the DoS protection in Mod-KaPoW, in a framework similar to that used in KaPoW plugin. However, Z-PoW proposes multiple new metrics to detect DDoS.

A. *Architecture*

Fig. 2 displays the flowchart of Z-PoW’s browser-side. At the beginning, the browser reads the puzzle from the server. After that, the browser generates a possible puzzle solution. If this solution is incorrect, it will try another one. But if the solution is correct, it will read the operation argument and will send it to the server along with the difficulty and the answer. If the operation is not complete, an error message, received from the server, will be displayed.

0 displays the flowchart of Z-PoW’s server side. The server reads the operation which can have three values: “null”, “preview” and “submit”. If it’s “null”, it’ll display all the old operations. On the other hand if it’s “preview”, it will first check whether the client puzzling is switched on or off. When the client puzzling is off, it will only make the difficulty equal to zero. But when the client puzzling is on, the score is initialized by zero. Then the server does several checks to calculate the score based on different metrics. These metrics check if the request is coming from The Onion Router (ToR), is a referrer, is blacklisted, is not permitted in the country, is a proxy, is a user agent or is the processor high, then the score will increase if one or more of the metrics is true by 1, 6, 5, 4, 1, 1 and 8 respectively. When the calculated score is less than the threshold, the difficulty will be equal to zero. But when the calculated score is higher than the threshold, the server will return a puzzle with a calculated difficulty. Finally, when the operation is “submit” and the client puzzling is on, the server will read the IP address, the answer, the difficulty and the operation. The server will also generate a puzzle based on the IP address and the given difficulty. After that, it validates the answer. In case the answer was wrong, an error message is displayed. However when the answer is correct, the operation is executed and the argument is saved.

B. *Attack Identification Metrics*

Based on [4,13,16,19], many factors were identified to help indicate the presence of a DDoS attack or that the user is potentially an attacker. The following factors are used as the puzzle metrics based on their disadvantages and their difficulty of implementation.

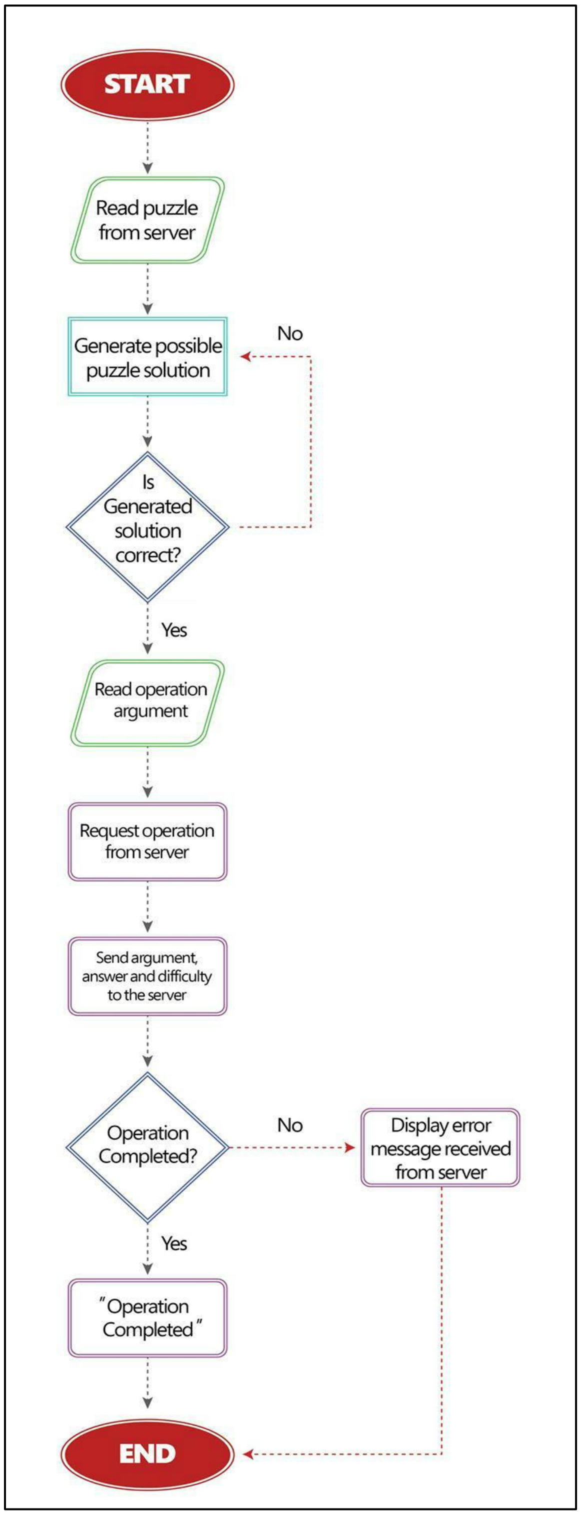
START
Read puzzle from server
Generate possible puzzle solution
No
Is Generated solution correct?
Yes
Read operation argument
Request operation from server
Send argument, answer and difficulty to the server
Operation Completed?
No
Display error message received from server
Yes
"Operation Completed"
END

Fig. 2. Z-PoW's Browser Side

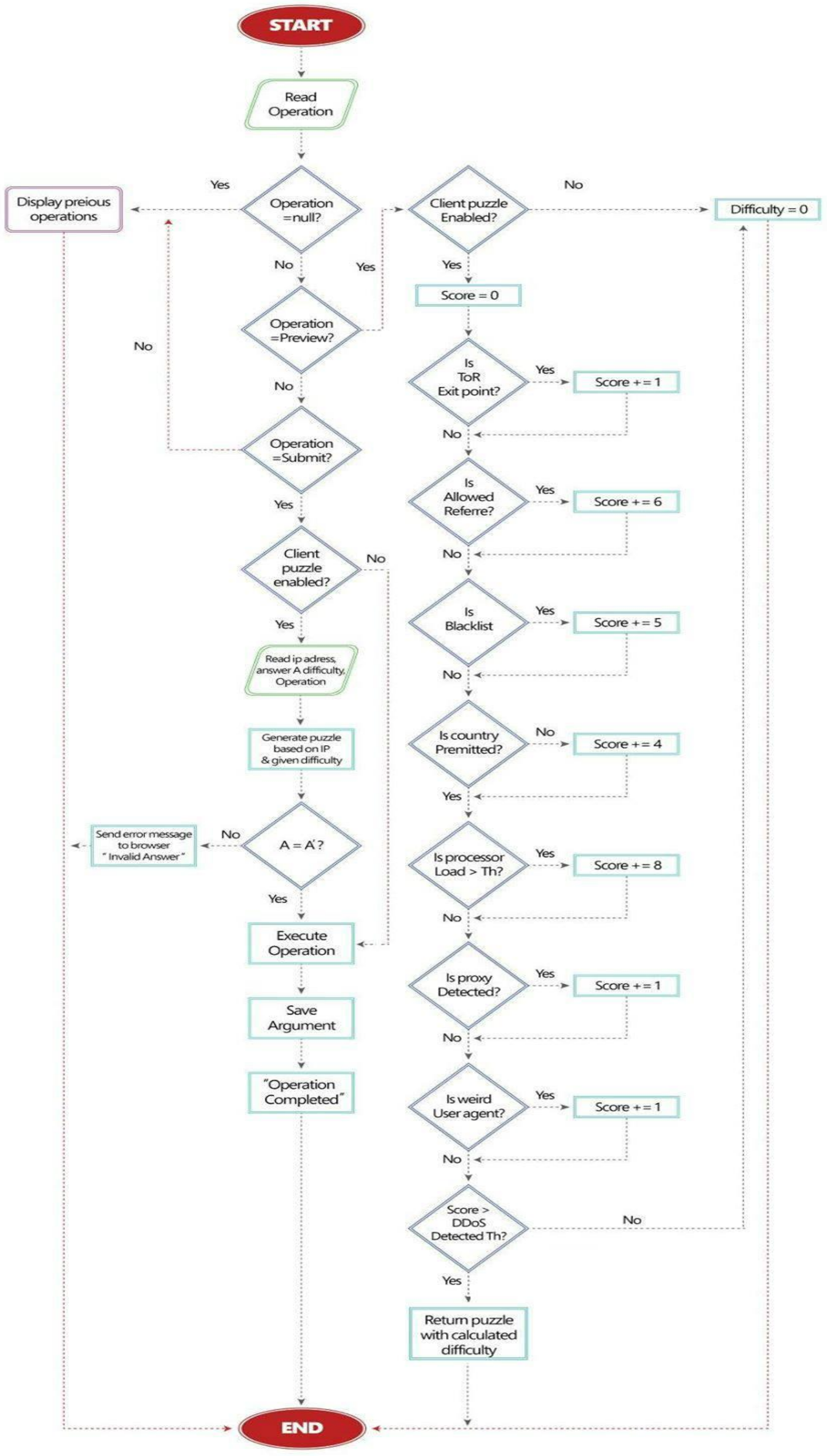

Fig. 3. Z-PoW's Server Side

TABLE I. METRICS SCORE CALCULATION

Metric	Blacklist	Processor	Geographical Location	Referring URL	ToR Network	Proxy Server	Weird User Agent
Overhead (Network Processing)	High	Low	High	Low	High	Low	Low
False Positives	Medium	High	High	Low	High	High	High
False Negatives	High	Low	High	High	High	High	High
Can be bypassed	No	No	Yes	Yes	No	Yes	Yes
Probability of metric occurrence given there is an attack launched	Med	High	Med	Med	Med	Med	Med
Probability of having an attack given that the metric is high	High	Very High	High	Medium	Low	Low	Medium
Impact on normal user	High	High	Low	Low	High	High	Low

Two of them are already used in DDoS_KaPoW: the existence of client's IP address in a blacklist and the increase in the server processing load. Other factors are used like using the client's geographic location to identify the clients who wouldn't normally access the server. Also, the absence of the referring URL indicates that this client is most likely a bot or a malicious user. If the request is originated from a bot then the user agent will probably have a signature which is known as one of the bad user agents. Although, in real life, the normal clients can use anonymizing networks, such as ToR and Proxy for privacy, but still the malicious users can exploit them to launch an attack.

C. Maliciousness Score Calculation

After selecting the metrics used to identify the presence of a DDoS attack, we established some factors to measure the effectiveness of each metric and assign its score. These factors are the processing overhead, false positives, false negatives, how difficult the metric can be bypassed, the probability of its occurrence, the accuracy of the metric and finally the negative impact on the normal clients as shown in . We gave each factor a score based on its variability. At the end, the score and the difficulty are calculated based on equations (1) and (2).

D. New Modules

In order to apply the new metrics, we integrated the proposed models with third party services and libraries like Windows Management Instrumentation (WMI) objects to calculate the processor load. We also integrated Z-PoW with DB-IP database to determine the client's geographic location, a referrer anomaly detector, ToR and Proxy detection libraries and finally a user agent anomaly detector.

E. Attack Simulation

During the implementation of the DDoS attack simulation, there was a problem with the browser automation because of its limitation of maximum number of simultaneous requests to the same domain. We tried many solutions like different browser profiles, different browsers instances using Selenium, different webdrivers using Python-Selenium Library, JavaScript to Python engine and Virtual machine with BeEF. But still all these solutions were neither satisfying nor feasible to solve the problem. At the end, we used a command line standalone JavaScript engine "PhantomJS" to conduct the attack simulation

We simulated the malicious user agents and proxy headers by injecting custom user agent and proxy randomly from the code. We added a module to select randomly from a list of the source IP addresses and feed it to both the simulated source IP header and the proxy header to simulate the clients behind a proxy. Also, we handled the case of unsolved puzzle, such that the operation will be discarded and the user's browser will have to request a new puzzle to solve (Retrying Request).

V. EXPERIMENT SETUP

A. Network Setup

To build the network, we used 5 machines: one machine acting as a server and 4 machines serving as clients (good and malicious). The server machine has 4GB RAM with Windows 8.1. The clients' machines: one has 1GB RAM with Windows 7 Ultimate; one has 3GB RAM with Windows 7 Professional; one has 2GB RAM and Windows 7 Starter and one with 4GB RAM and Windows 7 Ultimate. We built the network using an 8-port 100 Mbps desktop switch and straight through Ethernet cables.

B. Software Setup

On the server machine, we used XAMPP v3.2.1. Also, PHP v5.4.19 and Apache v2.4.4 were used. We used NetBeans IDE and xDebug to run all the models. Finally, to execute the simulation consoles remotely, we mounted the network drives. The server is designed to give priority to malicious users over normal ones. So as suggested in [14], we applied the limitation of accepting 4 clients simultaneously in DDoS_KaPoW and Z-PoW using Multi-Processing Modules "MPM" parameters. Also, we changed "PHP.ini" parameter to control the maximum execution time and adjust the default value from 30 to 80 seconds.

C. Simulation Assumptions

When the good and the malicious requests are sent; we send the good requests from one client machine using 2 consoles; except during experiment 1 and 2, we only use 1 console since the number of the good requests is very small. On the other hand, we send the malicious requests from the other clients' machines through 5 consoles. But when we only send good requests, they are distributed among all the clients' machines using 5 consoles on each. No requests are sent from the machine acting as the server. We used 900 seconds (15 minutes) as a threshold after which any request will be ignored because it's not feasible for an attacker to wait all that long for a single request; it's easier for him to launch a new attack.

VI. Results and Analysis

We have made various tests to measure the efficiency of the proposed models, and we have altered many variables to evaluate them in different environments and capture their performance. These tests aim to reveal the benefits and overheads of using client puzzling to defend against DDoS attacks. We conducted 10 experiments; each experiment consists of 6 tests. TABLE II and TABLE III display the different experiments and tests applied to test the setup and the behavior of the models under different environments and conditions.

While running the experiments, we noticed that a considerable amount of time was spent to process the good requests when the client puzzling is on. This wasn't desirable and affected the aim of the models. This amount of time was caused by the lookup for the ToR network. We removed this metric which saved a lot of time such that the average time taken by the requests during the presence of the ToR metric is triple the average response time during its absence.

A. Client Puzzling on vs off

We can conclude from Fig. 4 and Fig. 5 that the average response time of the good requests during tests ON/V/G and ON/F/G is higher than test OFF/G. This makes perfect sense because this gap represents the time taken to check the user maliciousness; it's the cost of security. We can also observe that the average response time of the good clients during test ON/V/G is almost the same as test ON/F/G with very few tweaks.

From Fig. 6 and Fig. 7, we can conclude that both models have almost the same behavior with very few differences, when only good requests are sent, whether the puzzle difficulty calculation was varied or fixed. Furthermore, the average response time in both models is directly proportional to the total number of requests. In Test OFF/G, the puzzle difficulty remains zero, throughout all the experiments in both models, as the client puzzling is switched off. During test ON/F/G, the difficulty also appears to be zero since the good clients' requests never exceeded the predefined threshold. Based on TABLE IV, in Test ON/V/G, the client will receive a puzzle difficulty with either zero or 131072 in Z-PoW and 32 in DDoS_KaPoW. These numbers '131072' and '32' refer to the difficulty calculated based on equation (2) when there is a high load processing on the server and the score is substituted by the processor load score which is 8 as mentioned in section IV. There are some exceptions in Test ON/V/G where the difficulty is zero like in Z-PoW' experiments 1 & 2 and DDoS_KaPoW experiments 1, 2 ,3 & 4. These exceptions are due to the small number of the sent requests such that it didn't affect the server processor.

TABLE II. Different Tests used

Test	Client Puzzling on/off	Client Puzzling varied / fixed	Good or/and malicious Clients
OFF/G	OFF	-	Only Good
ON/V/G	ON	Varied	Only Good
OFF/GM	OFF	-	Good & Malicious
ON/V/GM	ON	Varied	Good & Malicious
ON/F/G	ON	Fixed	Only Good
ON/F/GM	ON	Fixed	Good & Malicious

Either in Z-PoW or DDoS_KaPoW, all the requests coming from the good clients, with or without the client puzzling, received a response. There weren't any requests dropped even when the total number of requests was increased four times.

B. Varied Puzzle Difficulty Calculation

In reference with Fig. 8 and Fig. 9, in both models during Test OFF/GM, the average response time of the malicious and the good requests are close to each other.

On the other hand, in Test ON/V/GM, the average response time of the malicious requests is way greater than the average response time of the good ones. Sometimes, the average response time of the malicious requests is 25 times the average response time of the good ones. This proves that the client puzzling enhanced the good users' experience and punished the malicious clients by giving them complex puzzles and hence delaying the response of their requests.

TABLE III. Number of Clients during Different Experiments

Exp.#	Number of requests (good clients only)	Number of requests (both good and malicious clients)	
	Good	*Good*	*Bad*
1&2	280	25	255
3&4	560	50	510
5&6	1100	100	1005
7&8	2200	200	2010
9&10	4400	400	4020

Fig. 10 shows that during Test ON/V/GM, Z-PoW's performance is better than DDoS_KaPoW because the average response time of the malicious clients is very high in Z-PoW while it's slightly higher than the average response time of the good ones in DDoS_KaPoW.

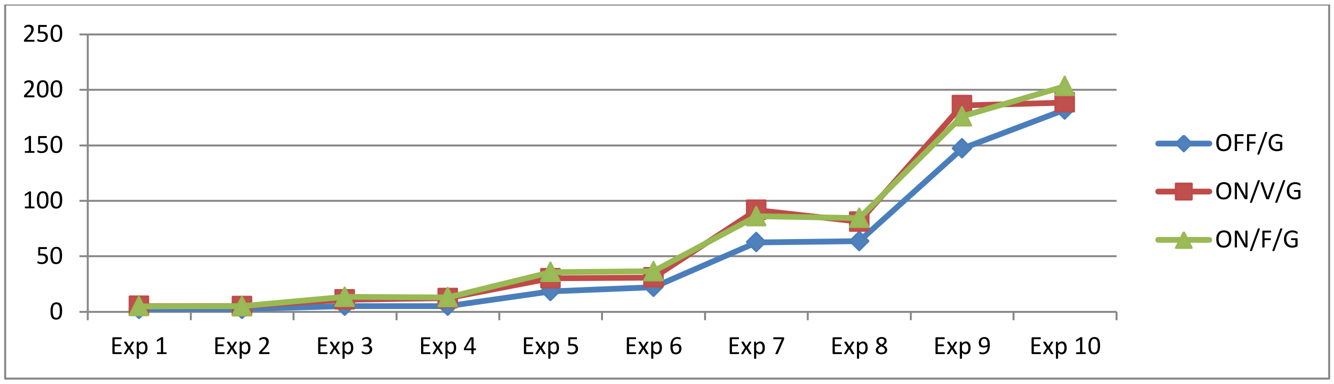

Fig. 4. Z-PoW's Average Response Time during Tests OFF/G, ON/V/G and ON/F/G

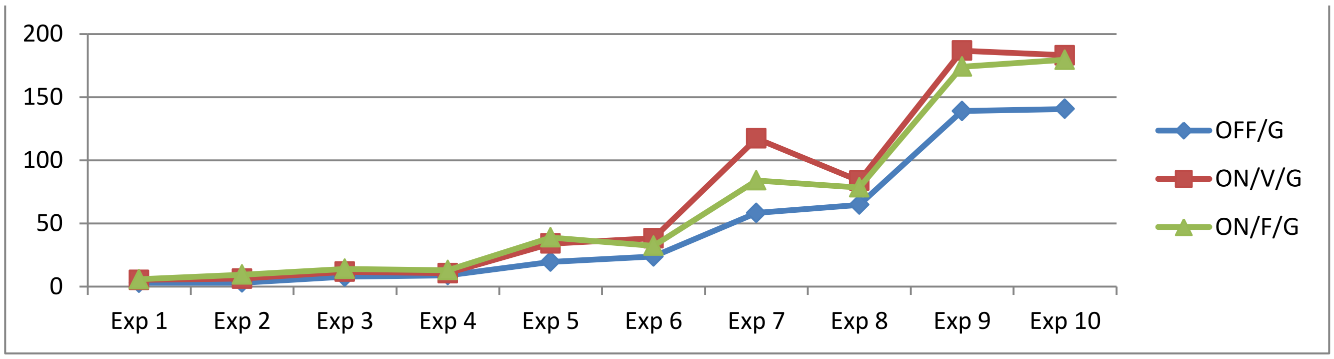

Fig. 5. DDoS_KaPoW's Average Response Time during Tests OFF/G, ON/V/G and ON/F/G

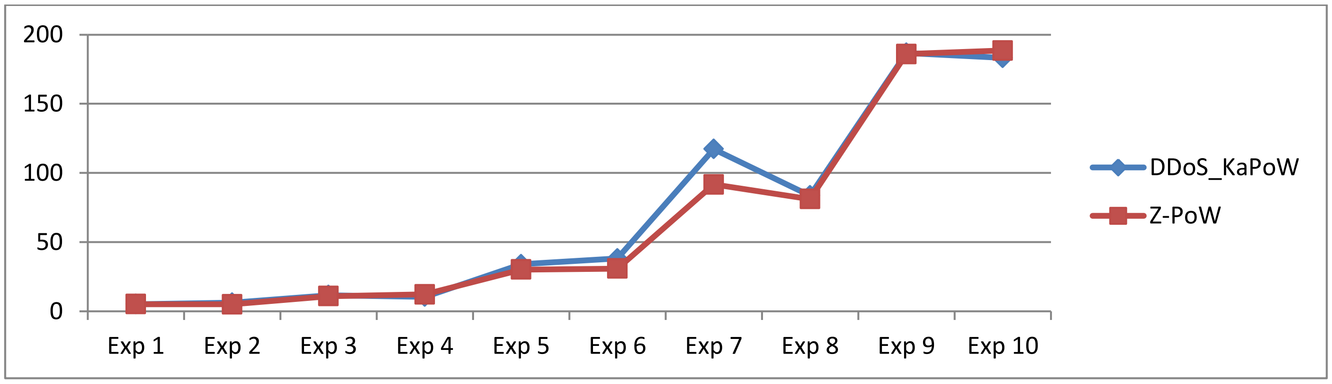

Fig. 6. Average Response Time of Both Models during Test ON/V/G

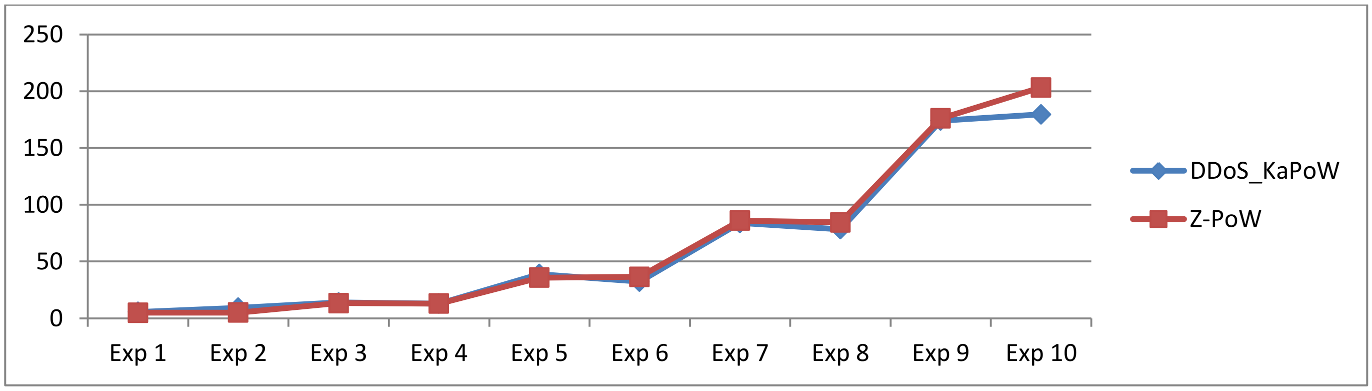

Fig. 7. Average Response Time of Both Models during Test ON/F/G

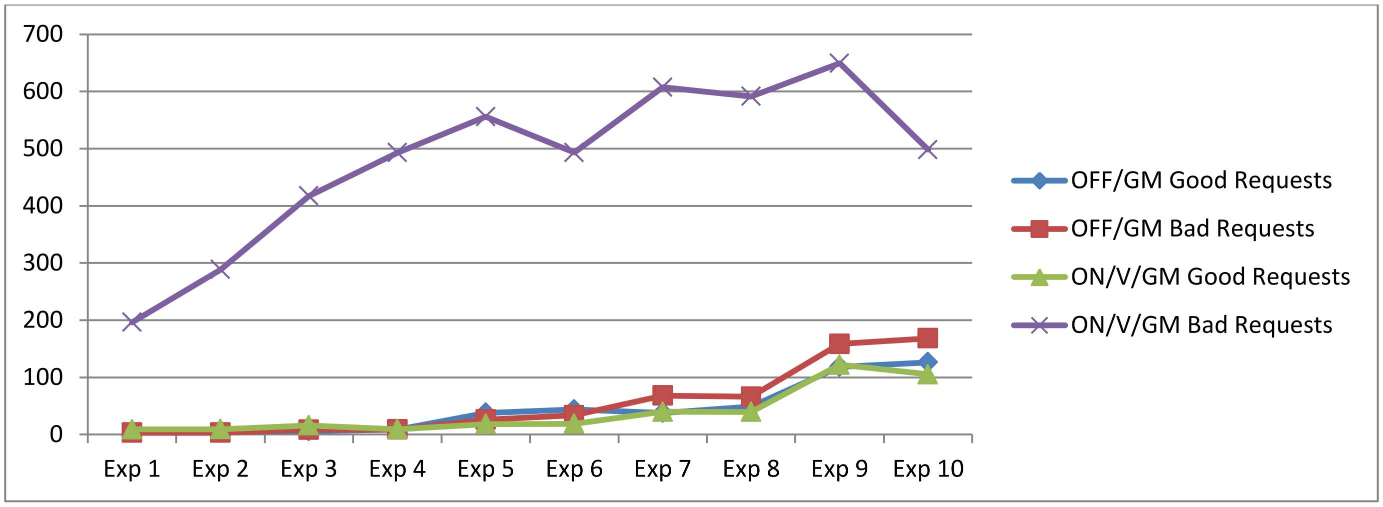

Fig. 8. Z-PoW's Average Response Time during Tests OFF/GM and ON/V/GM

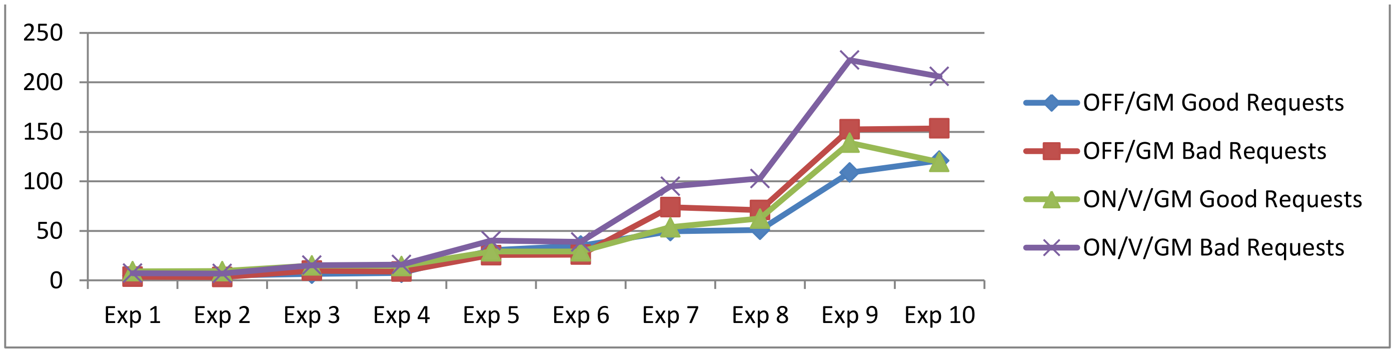

Fig. 9. DDoS_KaPoW's Average Respone Time during Tests OFF/GM and ON/V/GM

TABLE IV. THE MINIMUM AND MAXIMUM DIFFICULTY OF BOTH MODELS DURING TEST ON/V/G

ON/V/G Puzzle Dc	Z-PoW Dc		DDoS_KaPoW Dc	
	Min	Max	Min	Max
Exp. 1	0	0	0	0
Exp. 2	0	0	0	0
Exp. 3	0	131072	0	0
Exp. 4	0	131072	0	0
Exp. 5	0	131072	0	32
Exp. 6	0	131072	0	32
Exp. 7	0	131072	0	32
Exp. 8	0	131072	0	32
Exp. 9	0	131072	0	32
Exp. 10	0	131072	0	32

TABLE V displays the maximum puzzle difficulties calculated during Test ON/V/GM. As observed, the puzzle difficulty of the good requests during Z-PoW remained zero through all the experiments while it reached 32 during DDoS_KaPoW. This proves that increasing the number of the metrics didn't affect the processor load; on the contrary it enhanced the good user's experience. Finally, the puzzle difficulty of the malicious requests in Z-PoW is way higher than the malicious requests in DDoS_KaPoW and that's because Z-PoW uses 6 metrics instead of 2.

In Z-PoW and DDoS_KaPoW, no good nor malicious requests were dropped during any experiment in Test OFF/GM since the client puzzling is switched off. In both models, during test ON/V/GM there weren't any good requests dropped. TABLE VI shows the total number of the malicious requests sent and dropped during Test ON/V/GM for each experiment in both Z-PoW and DDoS_KaPoW.

In Z-PoW, when the client puzzling is on, a considerable amount of the malicious requests was dropped; even sometimes half of the requests were dropped. The number of the requests dropped is directly proportional to the total number of the sent requests. On the other hand, in DDoS_KaPoW, when the client puzzling is on, almost no malicious requests were dropped even when the number of the sent malicious requests was increased. So still the attackers will be able to access the server and dominate it at the end.

C. *Fixed Puzzle Difficulty Calculation*

Based on Fig. 11 and Fig. 12, in both models the average response time of the good clients, when the puzzle difficulty calculation is fixed (Test ON/F/GM), is higher than their average response time when the client puzzling is off (Test OFF/GM). This is the time cost of calculating the maliciousness score. On the other hand, the average response time of the malicious requests in test ON/F/GM is way higher than Test OFF/GM so both models succeeded at fulfilling their aim which is delaying the malicious clients by giving them harder puzzles which take more time to solve.

TABLE V. THE MAXIMUM DIFFICULTY OF BOTH MODELS DURING ON/V/GM

ON/V/GM Max Puzzle Dc	Z-PoW		DDoS_KaPoW	
	Good Requests	Malicious Requests	Good Requests	Malicious Requests
Exp. 1	0	3764768	0	12
Exp. 2	0	3764768	0	12
Exp. 3	0	3764768	0	12
Exp. 4	0	3764768	0	12
Exp. 5	0	3764768	0	32
Exp. 6	0	3764768	0	12
Exp. 7	0	3764768	32	32
Exp. 8	0	3764768	32	12
Exp. 9	0	885780	32	32
Exp. 10	0	3764768	32	12

Fig. 13 displays the average response time of the good and the malicious requests, in Z-PoW and DDoS_KaPoW, when the puzzle difficulty calculation is fixed (Test ON/F/GM). The average response time of the good requests of both models is almost the same with very few changes. Furthermore, the average response time of the malicious requests of Z-PoW is way higher than DDoS_KaPoW's. So, using more metrics helped delaying the malicious users and increasing their average response time.

During test ON/F/GM the minimum puzzle difficulty a user can get is 0 and the maximum puzzle difficulty, based on equation (2), is 500000 in Z-PoW and 50 in DDoS_KaPoW since the score used, after exceeding the predefined threshold, is equal to 10.

Based on TABLE VII, both models succeeded at preventing the attackers from accessing the server. Thanks to using more metrics, Z-PoW succeeded at preventing more malicious users and dropping their requests

D. *DDoS_KaPoW vs Z-PoW vs KaPoW Guestbook*

We simulated KaPoW Guestbook's model like Z-PoW's except that: one machine was acting as the server and only two client machines were used (one acting as the good clients and the other acting as the malicious ones) since it's a forum spam attack. This attack was launched 3 times, each time the number of consoles used by the attacker and the number of the sent requests were changed as shown in TABLE VIII.

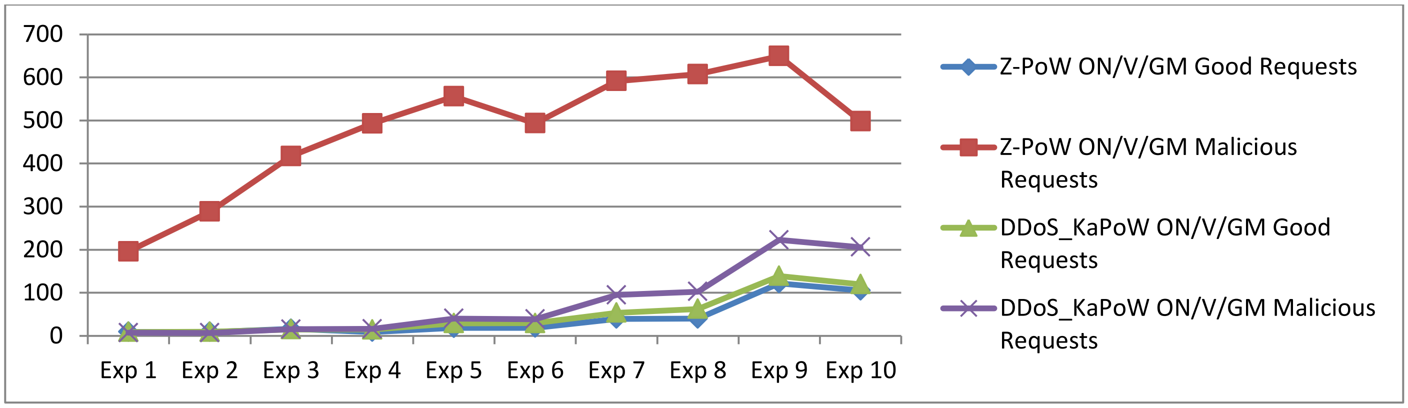

Fig. 10. Average Response Time of Both Models during Test ON/V/GM

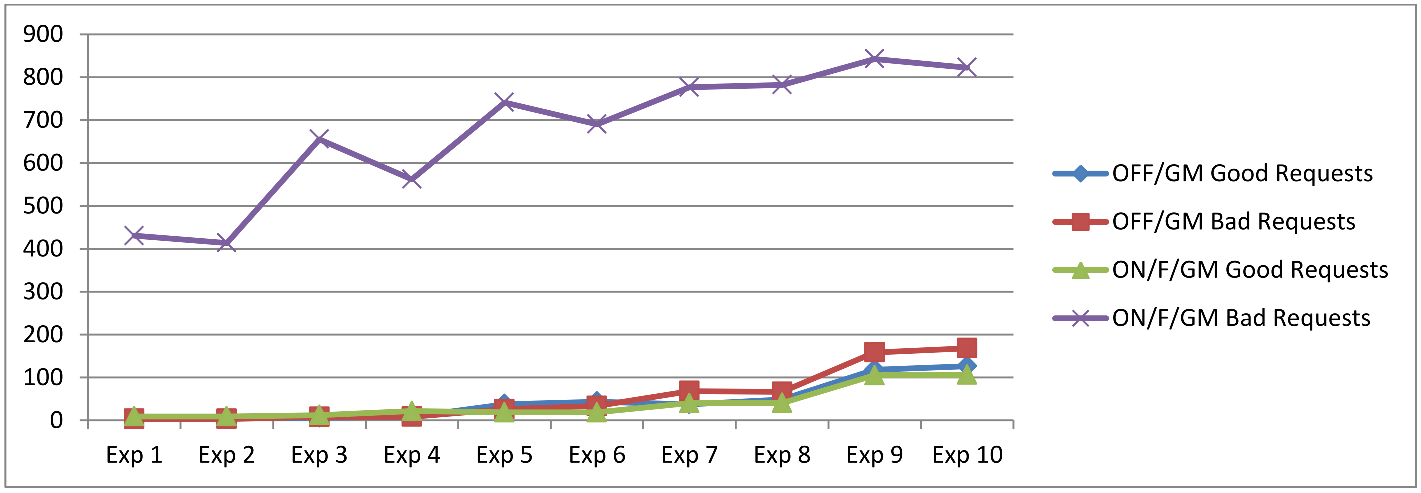

Fig. 11. Z-PoW's Average Response Time during Tests OFF/GM and ON/F/GM

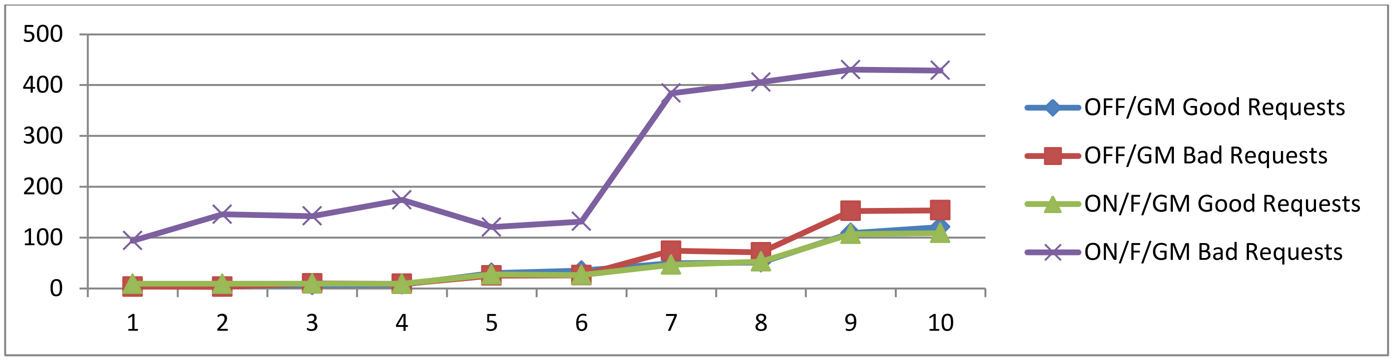

Fig. 12. DDoS_KaPoW's Average Response Time during Tests OFF/GM and ON/F/GM

TABLE VI. NUMBER OF DROPPED MALICIOUS REQUESTS OF BOTH MODELS DURING TEST ON/V/GM

ON/V/GM Malicious Requests	**Total Requests sent**	**Z-PoW Dropped Requests**	**DDoS_KaPoW Dropped Requests**
Exp. 1	255	20	0
Exp. 2	255	43	0
Exp. 3	510	126	0
Exp. 4	510	196	0
Exp. 5	1005	501	0
Exp. 6	1005	430	0
Exp. 7	2010	1045	0
Exp. 8	2010	1094	0
Exp. 9	4020	2418	2
Exp. 10	4020	1628	1

TABLE VII. NUMBER OF DROPPED MALICIOUS REQUESTS OF BOTH MODELS DURING TEST ON/F/GM

ON/F/GM Malicious Requests	**Total Requests sent**	**Z-PoW Dropped Requests**	**DDoS_KaPoW Dropped Requests**
Exp. 1	255	33	0
Exp. 2	255	39	10
Exp. 3	510	243	16
Exp. 4	510	184	9
Exp. 5	1005	704	24
Exp. 6	1005	636	45
Exp. 7	2010	1628	678
Exp. 8	2010	1611	724
Exp. 9	4020	3629	1502
Exp. 10	4020	3448	1539

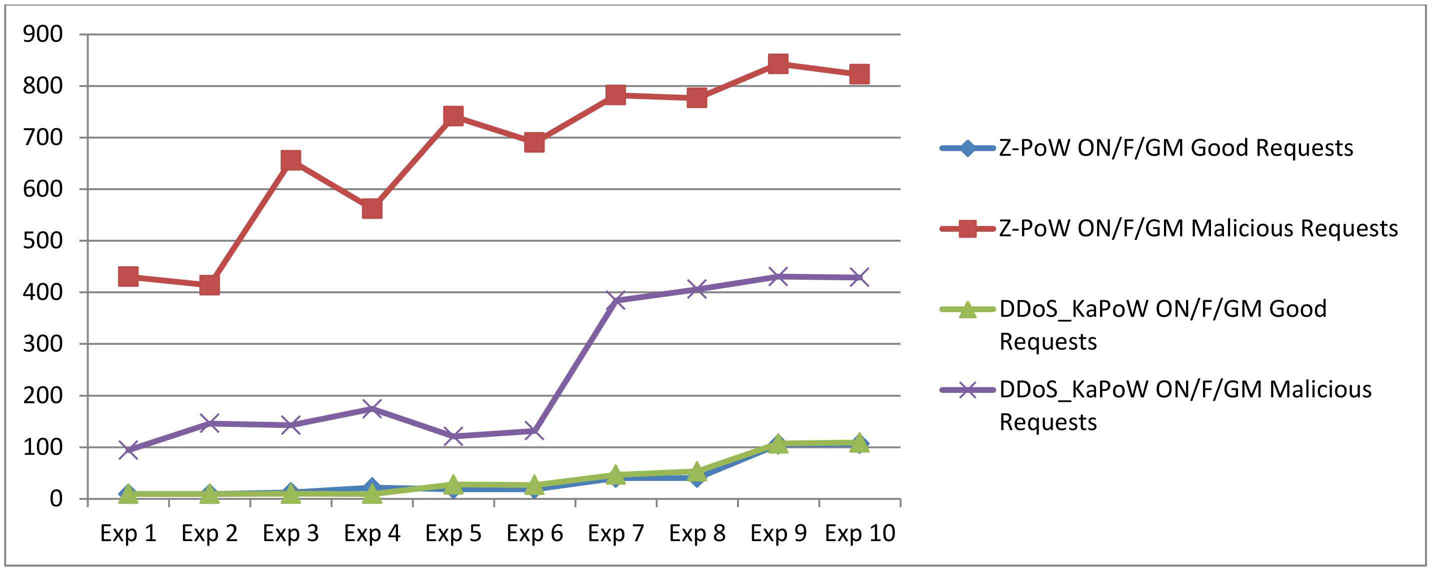

Fig. 13. Average Response Time of Both Models during Tests ON/F/GM

TABLE VIII. PERCENTAGE OF DROPPED REQUESTS IN DDOS_KAPOW, Z-POW AND KAPOW GUESTBOOK DURING TEST ON/V/GM

Exp.#	**DDoS_KaPoW (5 consoles)**		**Z-PoW ON/V/GM (5 consoles)**		**KaPoW Guestbook ON/V/GM (2 consoles)**		**KaPoW Guestbook ON/V/GM (5 consoles)**		**KaPoW Guestbook ON/V/GM (1 console)**	
	Total malicious	**% of Dropped**	**Total malicious**	**% of Dropped**	**Total malicious**	**% of Dropped**	**Total malicious**	**% of Dropped**	**Total malicious**	**% of Dropped**
1	255	0%	255	7.84%	250	0.80%	140	5%	140	10.71%
2	255	0%	255	16.86%	250	3.20%	140	2.85%	140	9.28%
3	510	0%	510	24.70%	510	20.50%	280	10.71%	280	33.57%
4	510	0%	510	38.43%	510	21.56%	280	5.71%	280	20%
5	1005	0%	1005	49.85%	1000	50.90%	550	23.09%	550	52.36%
6	1005	0%	1005	42.78%	1000	50.80%	550	25.81%	550	53.45%
7	2010	0%	2010	51.99%	2000	68.05%	1100	50.63%	1100	65.90%
8	2010	0%	2010	54.42%	2000	66.55%	1100	50%	1100	68.36%
9	4020	0.04%	4020	60.14%	4000	74.50%	2200	70.22%	2200	74.86%
10	4020	0.02%	4020	40.49%	4000	57.37%	2200	51.13%	2200	57.18%

TABLE VIII shows the percentage of the dropped malicious requests in each experiment when the client puzzling is on during the simulation of Z-PoW, DDoS_KaPoW and KaPoW Guestbook. As listed, the client puzzling dropped more malicious requests and defended DDoS attack better when the number of used metrics was increased. Almost both Z-PoW and KaPoW Guestbook have the same behavior with slightly differences which indicates that the client puzzling algorithm has comparable performance in defending against both DoS and DDoS, but it needed more metrics to defend the DDoS attacks.

VII. CONCLUSION

DDoS attacks are still considered a big threat for big companies. Although there is no 100% security but the client puzzling has proven its capability and efficiency to thwart DDoS attack through punishing the malicious clients without affecting the normal clients.

Z-PoW, is like KaPoW Guestbook, can be integrated in any application because of its modularity. It also investigates a lot of metrics to prevent the DDoS attackers from accessing the server. No good requests were dropped by applying the client puzzling which satisfies Z-PoW's goal.

Although the results of the tests with fixed difficulty are better than the tests with varied difficulty; some good clients may accidently be misinterpreted as malicious ones, hence suffer more receiving very hard puzzles.

Unfortunately Z-PoW has some deficiencies. One of them is that some normal users, who are using an automated tool or a plugin to block the referrer in the browser, will be considered as attackers because there won't be a referrer in the URL. Another flaw is the overhead added by the IP-to-country library because of the duplicate cache entries. Finally when a client has to retry a solution for the puzzle, the time taken to get a reply will be calculated from the second request sent, not from the first one.

VIII. FUTURE WORK

In order to make the malicious clients suffer more, the difficulty of their puzzle can be scaled up exponentially while the difficulty for well-behaved clients scales down linearly as suggested in [8]. Or the bad clients can be blocked after multiple spikes.

The look up of the nonce can be enhanced by using the counting bloom filters. The detection of the users coming from a ToR network or behind a proxy could be also enhanced, especially the ToR because it consumes a lot of time which is not effective.

In Z-PoW, we investigated the processor load using a Yes/No check. But in the future, a variable score can be used based on the load which will help detect a DDoS attack earlier.

The attackers can be simulated to be more sophisticated and by using General Processing Unit (GPU) cracking to facilitate solving the puzzles and compare the results with the normal clients.

Finally, Z-PoW can be enhanced by combining a Trust Model with the client puzzling. To cope up with everyday changes, Z-PoW needs to be compatible with HTML5 and IPv6. Also, DShield API changes need to be applied once they are done.

REFERENCES

[1] S. Mansfield-Devine, "DDoS: threats and mitigation," Network Security, pp. 5-12, December 2011.

[2] D. J. Nazario, "DDoS attack evolution," Network Security, 15 July 2008.

[3] T. L. a. D. H. Steven Simpson, "Identifying legitimate clients under distributed denial of service attacks," in Fourth international conference on network and system security, 2010.

[4] D. D. C. a. S. Landau, "The problem isn't attribution, it's multistage attacks," in ACM Re-Architecting the Internet Workshop (ReArch), Philadelphia, 2010.

[5] M. Yoon, "Using whitelisting to mitigate DDoS attack on critical internet sites," IEEE Communications Magazine, July 2010.

[6] M. M. a. J. C. M. Ellie Bursztein, "Text-based CAPTCHA strengths and weaknesses," in Proceedings of the 18th ACM conference on Computer and Communications Security, Chicago, illinois, USA, 2011.

[7] D. L. a. L. Camp, "Proof of work can work," Fifth Workshop pn the Economics of Information Security, 2006.

[8] P. N. a. J. L. Tuomas Aura, "DoS resistant authentications with client puzzles," 8th International workshop on Security Protocols, pp. 170-177, April 2000.

[9] P. M. N. P. S. a. B. W. Liquin Chen, "Security notions and generic constructions for client puzzles," Mitsuri Matsui, editor, Advances in Cryptology - Proceedings ASIACRYPT 2009. LNCS, Springer (2009), vol. 5912, pp. 505-523, 2009.

[10] D. S. A. C. a. H. L. Suriadi Suriadi, "Defending web services against denial of service attacks using client puzzles," in IEEE ICWS "International Conference on Web Services", 2011.

[11] D. D. a. A. Stubblefield, "Using client puzzles to protect TLS," in 10th conference on USINEX Security Symposium, Washington, 2011.

[12] J. J. O. F. R. S. D. J. a. C. A. G. Jeff Green, "Reconstructing hash reversal based proof of work "PoW" schemes," in LEET'11 Proceedings on the 4th USENIX conference on LArge-scale exploits and emergent threats, Boston, 2011.

[13] D. a. W.-C. F. Tien Le, "KaPoW plugins: protecting web applications using reputation-based proof-of-work," WebQuality 2012, April 2012.

[14] E. K. a. W.-C. Feng, "Mod_kaPoW: protecting the web with transparent Proof-of-Work," IEEE INFOCOM, p. 16, 2008.

[15] E. K. a. W.-C. Feng, "Mod_kaPoW: mitigating DoS with transparent proof-of-work," in The 3rd International Conference on Emerging Networking Experiments and Technologies (CONExT), 2007.

[16] E. K. a. W.-C. Feng, "KaPoW webmail: effective disincentives against spam," in CEAS 2010 - 7th Annual Electronic messaging, Antiabuse and Spam conference, Washington, 2010.