CAPoW: Context-Aware AI-Assisted Proof of Work Based DDoS Defense

Trisha Chakraborty, Shaswata Mitra and Sudip Mittal
Department of Computer Science & Engineering, Mississippi State University, Starkville, MS, U.S.A.

Keywords: Distibuted Denial of Service Attacks, Proof of Work, Artificial Intelligence.

Abstract: Critical servers can be secured against distributed denial of service (DDoS) attacks using proof of work (PoW) systems assisted by an Artificial Intelligence (AI) that learns *contextual* network request patterns. In this work, we introduce CAPOW, a *context-aware* anti-DDoS framework that injects latency *adaptively* during communication by utilizing context-aware PoW puzzles. In CAPOW, a security professional can define relevant request context attributes which can be learned by the AI system. These contextual attributes can include information about the user request, such as IP address, time, flow-level information, etc., and are utilized to generate a contextual score for incoming requests that influence the hardness of a PoW puzzle. These puzzles need to be solved by a user before the server begins to process their request. Solving puzzles slows down the volume of incoming adversarial requests. Additionally, the framework compels the adversary to incur a cost per request, hence making it expensive for an adversary to prolong a DDoS attack. We include the theoretical foundations of the CAPOW framework along with a description of its implementation and evaluation.

1 INTRODUCTION

An organization protects its critical servers from distributed denial of service (DDoS), which may contain valuable information, such as intellectual property, trade secrets, employee personally identifiable information (PII), etc. To launch a volumetric DDoS attack, the malicious users send a flood of requests to these servers. As a result, requests from legitimate users either experience delays or their requests are dropped. For more than two decades, DDoS attacks have been a prominent issue and even today it is far from being solved as these attacks are cheaper to launch than to defend, especially with the rise of DoS-as-a-Service (Orlowski, 2016).

Proof of work (PoW) system works by requiring incoming requests to expend resources solving an *computational puzzles* to prove one's legitimacy. The general system consists of two parts: *prover* and *verifier*. The prover finds the solution to the computational puzzles, when solved, sends the solution to the verifier. In a simple networked client-server environment, the user-side contains the prover component, and the server-side contains the verifier components. Researchers have proposed PoW-based solutions for DDoS which makes the attack expensive to launch (Aura et al., 2000; Waters et al., 2004; Mankins et al., 2001). Although, these solutions suffer from a lack of intuition on how to set puzzle difficulty and adaptability in different settings.

In this paper, we develop a defensive framework that emphasizes learning the normal activity patterns of legitimate users. The idea behind the framework is to penalize the users that *deviates* from normal activity patterns by issuing them *hard* puzzles and at the same time issuing *easy* puzzles to users who follow the pattern. We leverage a *context-aware AI model* that can learn these normal activity patterns by contextual information. The term *context* within the scope of legitimate activity patterns can be defined as request attributes, such as IP address, time, flow-level information, etc. When the context is *IP address*, network activity is considered deviated if the source IP address is part of a known blocked IP list. Whereas, when the context is *time*, network activity is considered deviated if it arrives at an unusual time compared to the normal activity pattern. Security professionals can select relevant request context attributes which can be learned by the AI models. The concept of *context-aware AI models* is derived from context-aware computing introduced by Dey et. al (Dey, 2001).

We introduce CAPOW framework, a *context-aware* AI-assisted PoW system that improves the defensive posture of critical servers against DDoS attacks. Our framework utilizes context-aware AI mod-

Chakraborty, T., Mitra, S. and Mittal, S.
CAPoW: Context-Aware AI-Assisted Proof of Work Based DDoS Defense.
DOI: 10.5220/0012069000003555
In *Proceedings of the 20th International Conference on Security and Cryptography (SECRYPT 2023)*, pages 62-72
ISBN: 978-989-758-666-8; ISSN: 2184-7711

els that learn the expected context pattern from server-side activity-logs. The activity-logs are stored and managed by the server which contains user activity (IP address, timestamp, flow-level data, etc). The deviation from the learned pattern is then leveraged to generate a *contextual score* for incoming requests which tune the difficulty level of the PoW puzzle to be solved. *The underlying defensive strategy ensures that the rate of incoming malicious users is throttled by adaptively introducing latency through PoW puzzles and compelling malicious users to expend more resources to complete an attack.* The main contributions of this paper are as follows.

Contribution 1: We introduce CAPOW, an anti-DDoS framework that injects latency adaptively, i.e., the framework ensures that malicious users incur higher latency than legitimate users based on the deviation in context pattern. We discuss the process of context score calculation from deviation in Section 3.2.

Contribution 2: We propose a policy component that is created by security personnel to incorporate server-specific security demands. We provide intuition for policy construction in Section 3.3.

Contribution 3: We discuss an instance of CAPOW implementation and perform a preliminary evaluation to illustrate the effectiveness of CAPOW. The implementation details are discussed in Section 4. The code is released on GitHuB (Chakraborty, 2022).

The rest of the paper is structured as follows. In Section 2, we discuss our threat model and attack definitions. We discuss the theoretical foundation of CAPOW in Section 3 and CAPOW implementation and preliminary evaluation in Section 4. Section 5 provides a discussion on common questions and limitations of the PoW framework. We discuss related works of the PoW system and DoS defense in Section 6, followed by the conclusion in Section 7.

2 THREAT MODEL

In this section, we present a series of assumptions associated with the adversary's abilities. An adversary $\mathbb{A}$ initiates a DDoS attack by sending a flood of requests to the server. The adversary's intention is to overwhelm the server's computational resources and disrupt legitimate user communication with the server. *Note that the server disregards any requests that do not solve the assigned puzzle and such requests are dropped.* In Section 5, we underline common questions related to the use of proof of work as a DDoS defense. The assumptions described below are similar to previous literature on DDoS defense using proof of work (Juels and Brainard, 1999) and in some sense, we consider a stronger adversary.

Assumption 1. Adversary $\mathbb{A}$ can eavesdrop on the communication channel of the server. $\mathbb{A}$ cannot modify any user request and cannot read any request payload data.

Assume a secure network communication channel is used by the user to send request packets to the server. The user performs encryption on the payload data, including the puzzle solution, and sends the packet to the server. When an adversary eavesdrops on the channel, they can read the source and destination IP of the packet, but they cannot read the encrypted payload consisting of the puzzle parameters. Additionally, the adversary cannot flip bits of the packet and pollute the puzzle solution included in the payload. Hence, we assume that the adversary has no knowledge of the puzzle parameters used by a user nor can it deny service to a user who has correctly solved the puzzle. In Section 4, we utilize assumption 1 to claim that the adversary cannot reverse engineer the base AI models to receive easier PoW puzzles by mere eavesdropping on the communication channel .

Assumption 2. Adversary $\mathbb{A}$ can appear as legitimate users by spoofing user identifiers, such as IP addresses, and deceive a subset of underlying AI models.

CAPOW uses AI models to learn normal network activity patterns and the deviation from the pattern is directly proportional to the difficulty of PoW puzzles to be solved by the user. $\mathbb{A}$ can spoof a legitimate user identifier (e.g. IP address, user token, etc.) and send requests to the server. An intelligent adversary would send probe packets to the server using a set of spoofed users and only utilize user identifiers that require easier puzzles to be solved. This way, the adversary is able to deceive the AI model and reduce the latency introduced. In Section 4, we discuss that due to the usage of more than one AI model, sending probe packets becomes costly for an adversary to deceive multiple base AI models.

Assumption 3. Adversary $\mathbb{A}$ cannot pollute the training data of the AI models.

The AI model used by CAPOW learns normal activity patterns and calculates a deviation which directly influences the hardness of the puzzle. Hence, it is essential that the AI learns normal activity patterns from an unpolluted activity-log to maximize the effectiveness of CAPOW. In Section 4.2, we describe the training process of a context-aware AI model where a security professional is deployed to select secure data to train the base AI models.

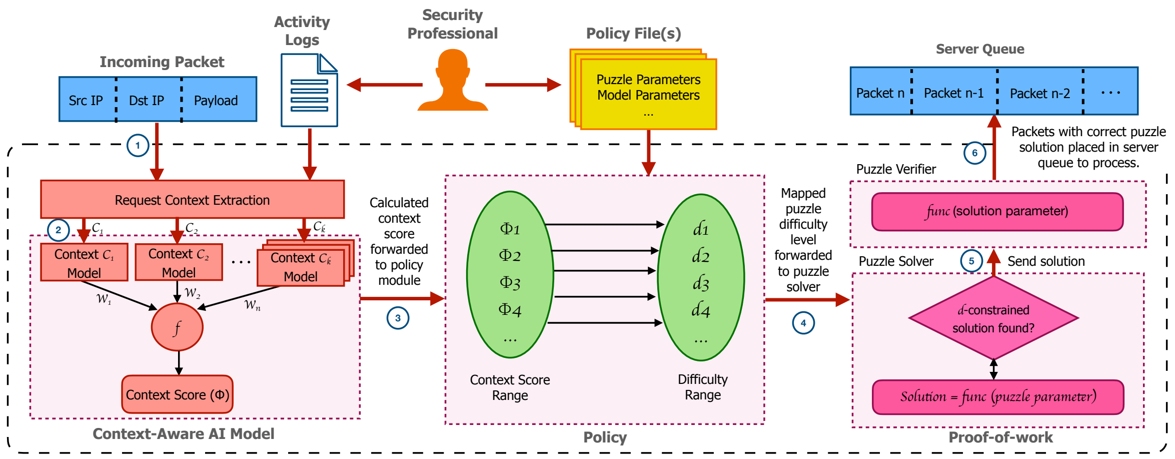


Figure 1: The figure illustrates the architecture of CAPoW framework. CAPoW consists of four core components: request context extractor, context-aware AI model, policy, and proof of work. The AI model learns context patterns from previous activity-logs selected by security personnel and calculates a context score based on the deviation of the incoming packet. The calculated score is mapped to the PoW puzzle difficulty level as defined by the security professional in policy files. The proof of work component performs evaluations to find the constrained solution. The request with a correct solution is placed on the server queue to process.

3 CAPoW ARCHITECTURAL DESIGN AND THEORETICAL FOUNDATIONS

In this section, we describe the high-level architecture of the core components and their inner workings that molds the CAPoW framework. As shown in Figure 1, CAPoW consists of four core components: *request context extractor, context-aware AI models, policy*, and *proof-of-work*.

The AI models learn the normal activity pattern from previous activity-logs. When an incoming request packet is seen, first the context attributes are extracted from the new request packet (see Section 3.1). Then, the deviation between the learned normal context pattern and new request contexts is computed to calculate *context score*. We elaborate on AI model training and score calculation in Section 3.2. The policy component of CAPoW provides security professionals with certain abilities that strengthen the effectiveness of CAPoW in various security settings (see Section 3.3). The context score influences the difficulty of the PoW puzzle. In Section 3.4, we discuss the proof-of-work component and how the PoW puzzles can curtail the volume of malicious user requests by adaptively introducing latency.

Data Flow. In Figure 1, the flow of data between different components of CAPoW is described below. (1) When a new incoming packet is seen, the request packet is forwarded to the request context extractor. (2) The extracted request context attributes are passed to context-aware AI models which learned expected context patterns from activity logs. The context score generated by individual AI models is combined using a function f to produce the final context score (Φ). (3) The context score is forwarded to the policy component which sets certain parameters, such as it maps the context score to a puzzle difficulty level. (4) The difficulty level is passed to the puzzle solver which solves a puzzle of the defined difficulty level using a function *func*. (5) The computed solution is sent to the verifier. (6) When the solution is correct, the request packet is placed on the server queue for processing.

3.1 Context Extraction from Request Packet

The concept of context-aware computing was introduced by Dey et. al (Dey, 2001), where the proposed mechanism improved human-to-computer interaction by delivering contextually relevant data. In the paper, the author proposed an abstract definition of *context*, which is a piece of information that clarifies the characteristics of an entity. When a system contains contextual data about a situation or entity, the system can take context-aware decisions which improve the overall quality of any general decision-making.

In a security setting, a certain request is deemed suspicious if the associated request attributes deviate from the usual network activity pattern. For instance, a request packet of payload size 65500 bytes is considered suspicious due to deviation when the expected normal payload size pattern is in the order of a few hundred bytes. To this end, we define *context* of a request packet as request attributes, such as source IP address, time of arrival, port address, time to live (TTL), and other flow-level attributes. The context attributes to be extracted are selected by security personnel via the policy component. The list of selected context attributes is reformed periodically to update the defensive posture of the organization deployed. When a new request packet is seen, the request context extractor component extracts the selected context attributes from the request packet and feeds it to the context-aware AI models.

3.2 Context-Aware AI Model

The framework component consumes activity-logs supplied by security personnel as input to generate a context-aware AI model. The model is generated by considering a set of request packets from the activity-log $\lambda = \{\lambda_0, \lambda_1, \lambda_2, ..., \lambda_i\}$. Each request packet λ_i consists of a set of request context attributes,

$$\mathbb{C}_{\lambda_i} = \{\mathbb{C}_{0\lambda_i}, \mathbb{C}_{1\lambda_i}, \mathbb{C}_{2\lambda_i}, ..., \mathbb{C}_{k\lambda_i}\} \quad (1)$$

where k is the number of request context attributes. $\mathbb{C}_k$ is represented as n-dimensional vector. When an n-dimensional vector of a single context for λ requests is projected in Euclidean space, such relative positioning produces a cluster. For k context attributes, k clusters are generated. The clusters represent the normal activity pattern. To evaluate a new incoming request, the request context extractor from Section 3.1, feeds the context attributes which are then projected in Euclidean space. The deviation $\Delta(p,q)$ of context $\mathbb{C}_k$ is calculated as the Euclidean distance between the corresponding normal activity cluster and the new request projection,

$$\Delta(p,q) = \sqrt{\sum_{j=1}^{n}(q_j - p_j)^2} \quad (2)$$

where p is the projected single context attribute of the new request and q is the center of a normal cluster of the same context. Consequently, the context score Φ for $\mathbb{C}_k$ is calculated as,

$$\Phi(\mathbb{C}_k) = \left(\frac{\Delta(p,q)}{\Delta_{max}}\right) \times I \quad (3)$$

where Δ_{max} is the maximum possible deviation for $\mathbb{C}_k$. The score is in the range of $[0, I]$, where $I \in \mathbb{Z}^+$. In Section 4.2, we discuss the implementation of context-aware AI models.

3.3 Policy

The policy component is a rule-based strategy that facilitates the adaptive security guarantees of CAPoW. The rules are set in policy files that determine certain CAPoW characteristics. These characteristics include context-aware AI model specifications, such as, which activity-logs are supplied to train the AI models, which context attributes hold more significance over the others, etc. Additionally, these parameters include proof-of-work components specifications, such as, what is the rule to translate context score to puzzle difficulty, which variant of PoW puzzle to be used, etc. Hence, it is evident that policy construction is a non-trivial task and requires consideration of various facets of the deployed server to bolster the effectiveness of CAPoW in different security settings. To perform the convoluted task of policy designing, *security professionals* are deployed to design server-specific policies.

Intuition for AI Model Parameters. From Section 3.1, a request packet consists of several context attributes. The significance of some contexts holds more importance over others depending on the type of attack defense. For instance, payload size is an important context attribute to protect against large payload DDoS attacks (Ozdel et al., 2022), but less important to defend against volumetric DDoS attacks. The policy includes the weight associated with context attributes to provide an attack-specific defense. Additionally, a policy includes the source of data to train the AI models to avoid model data pollution attacks (Assumption 3).

Intuition for Proof-of-Work Parameters. The context score produced by the context-aware AI model is translated to the PoW difficulty level. The policy includes the rules to translate context scores to puzzle difficulty. In Section 4.3, we implemented three rules to show that the translation leads to adaptive latency injected. As stated by Green et. al (Green et al., 2011), amongst groups of users, the CPU capacity of each device can vary 10x times, whereas memory capacity may only vary 4x times. Hence, when a memory-bound PoW puzzle is used, it is less likely for the adversary to have an edge over a legitimate user as the discrepancy in memory power as the resource is less compared to CPU-bound puzzles. The policy includes the means to set variants of puzzles depending on the expected user base.

3.4 Proof of Work

Classical proof of work systems (Dwork and Naor, 1992; Waters et al., 2004; Aura et al., 2000) consists of two main components – *prover* and *verifier*. The prover provides verifiable evidence of expanding computational resources by solving puzzles as assigned by the server. On the other hand, the verifier validates whether the solved puzzle yielded the desired solution. When PoW systems are used as DoS defense (Aura et al., 2000; Wood et al., 2015; Parno et al., 2007), a user commits some computation resources (CPU cycle, bandwidth, etc.) and *burns* one of these resources for solving the PoW puzzle to prove their legitimacy.

In CAPoW, when a user deviates from a normal activity pattern, the PoW component issues a PoW puzzle to request proof of legitimacy. The difficulty level of the PoW puzzle is a function of the context score. The rule to translate to context score to difficulty level is defined under the policy component (Section 3.3). PoW solver uses a function *func* to solve the assigned difficulty puzzle (see Figure 1). In general terms, this function injects two types of cost: (1) direct cost of *resource burning* (Gupta et al., 2020), and (2) indirect cost of *latency*. The notion of resource burning cost represents the resource consumption of a user, where the resource could be computational power, memory, network bandwidth, or human capital (Gupta et al., 2020). This cost directly impacts the ability of the adversary to conduct a DDoS attack as every request requires the adversary to spend real-life resources. The notion of *latency* cost captures the delay in time introduced in communication due to the act of puzzle solving. This cost indirectly impacts the adversarial intent by throttling the rate of adversarial requests reaching the server queue. Both costs ultimately cripple the adversarial capability to prolong an ongoing DDoS attack.

4 CAPoW IMPLEMENTATION, FRAMEWORK INSTANCE DEPLOYMENT, AND EVALUATION

In this section, we present a deployment of the CAPoW framework by implementing a single instance of each core component: context extractor, context-aware AI models, policy, and proof-of-work. Our code is written in Python 3.7 and available online in Github (Chakraborty, 2022). First, the context extractor instance extracts selected request context attributes. Second, the extracted contexts are relayed to context-aware AI model instances where each base AI model is generated using server-side activity-logs. Then, the trained AI models calculate the deviation of selected contexts to produce a context score. Third, we provide three policy designs that map the context score to the difficulty of the PoW puzzle. Finally, we implemented a hash-based PoW puzzle instance which, over repeated trials, finds the constrained solution of the assigned difficulty level. The costs inflicted due to our puzzle instance are CPU cycles (resource burning) and time spent (latency). For the purposes of validating our contribution via evaluation, we consider that the main cost injected is latency which, when injected, slows down the rate of adversarial requests.

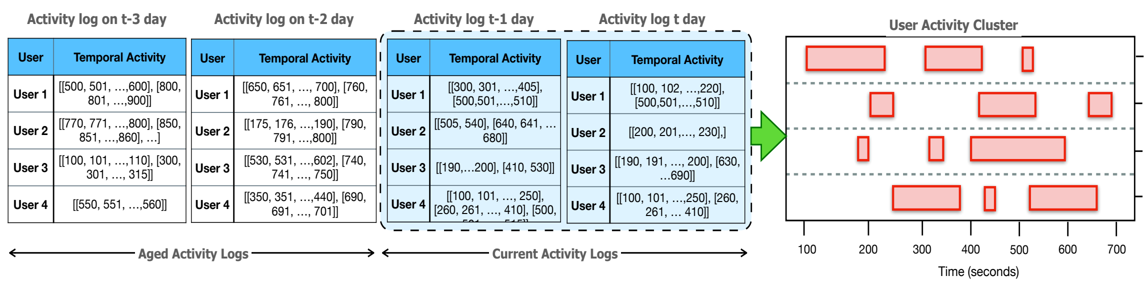
Now, we will describe our preliminary evaluation setup. We split the CIC-IDS2017 dataset (of New Brunswick, 2017) into test and train files where day 1 to day 4 (Monday - Thursday) is used to train the models and day 5 (Friday) is used to evaluate CAPoW. From day 1 to day 5, we deleted the attack traffic to learn the normal activity pattern. Consider five users sending requests to the server $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4$, and $\mathcal{U}_5$. We fixed four user identifiers from day 5 to map the four above-mentioned users. Let the fifth user $\mathcal{U}_5$, be mapped to the user identifier that performs DoS on the day 6. Since, the user identifier in CIC-IDS2017 is IP address, let the mapped IP of user $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4$, and $\mathcal{U}_5$ is represented by `104.20.30.120`, `83.66.160.22`, `37.59.195.0`, `104.16.84.55`, and `205.174.165.73` respectively. Through our evaluation scenario, we provided evidence that CAPoW injects latency adaptively based on the calculated context score of user $\mathcal{U}_5$ which slows down the adversarial requests and makes it expensive for an adversary to prolong a DDoS attack.

4.1 Context Extraction Instance

The context extraction instance consumes the request packet and extracts context attributes from the request packet. For our implementation, we select three context attributes: (1) IP address, (2) temporal activity, and (3) flow-level data. For evaluation, we used feature attributes of the CIC-IDS2017 dataset to serve as context attributes. The source IP feature becomes the IP address context, the timestamp feature becomes the temporal activity context, and the remaining features become the flow-level context.

4.2 Context-Aware AI Model Instance

We propose an ensemble learner that consists of dedicated base AI models to learn individual contextual patterns. The base AI model receives the context attributes from the context extractor as inputs. The model that (1) learns the IP address pattern is called dynamic attribute-based reputation (DAbR), (2) learns the temporal activity pattern is called tem-



Activity log on t-3 day

User	Temporal Activity
User 1	[[500, 501, ...,600], [800, 801, ...,900]]
User 2	[[770, 771, ...,800], [850, 851, ...,860], ...]
User 3	[[100, 101, ...,110], [300, 301, ..., 315]]
User 4	[[550, 551, ...,560]]

Activity log on t-2 day

User	Temporal Activity
User 1	[[650, 651, ..., 700], [760, 761, ..., 800]]
User 2	[[175, 176, ...,190], [790, 791, ...,800]]
User 3	[[530, 531, ...,602], [740, 741, ..., 750]]
User 4	[[350, 351, ...,440], [690, 691, ..., 701]]

Activity log t-1 day

User	Temporal Activity
User 1	[[300, 301, ...,405], [500,501,...,510]]
User 2	[[505, 540], [640, 641, ... 680]]
User 3	[[190,...200], [410, 530]]
User 4	[[100, 101, ..., 250], [260, 261, ..., 410], [500, 501, ..., 515]]

Activity log t day

User	Temporal Activity
User 1	[[100, 102, ...,220], [500,501,...,510]]
User 2	[[200, 201,..., 230],]
User 3	[[190, 191, ..., 200], [630, ...690]]
User 4	[[100, 101, ...,250], [260, 261, ... 410]]

Figure 2: The figure shows that selected activity-logs (left) are used to generate a temporal activity model (TAM) (right). The illustration shows that out of four activity logs, currently only two activity logs are used to form the model (blue box). The remaining activity-logs are aged in an attempt to keep the model up-to-date.

poral activity model (TAM), and (3) learns the flow-level data pattern is called flow-level model (FLOW). Each model computes a context score in the range between $[0,10]$. Context scores from three AI models are combined using the argmax function. Next, we discuss three base models where the subsections are divided into model generation, context score calculation, and evaluation.

Dynamic Attribute-Based Reputation (DAbR): We utilize DAbR (Renjan et al., 2018) as the base AI model that learns context patterns for IP attributes. The AI model is generated by projecting malicious IP attributes from the Cisco Talos dataset (Talos, 2022) into Euclidean space. The dataset contains a list of malicious IP addresses and IP-related attributes. The red dots in Figure 3(A) represent the projected malicious IP attributes that form a cluster in Euclidean space. When a new request is evaluated, the IP attributes of the new request are projected in Euclidean space and a deviation is calculated as Euclidean distance to the malicious cluster center. The distance calculated produces the context score for DAbR (α). The multi-colored stars represent $\mathcal{U}_1$, $\mathcal{U}_2$, $\mathcal{U}_3$, $\mathcal{U}_4$, and $\mathcal{U}_5$. User $\mathcal{U}_1$, $\mathcal{U}_2$, $\mathcal{U}_3$, $\mathcal{U}_4$, and $\mathcal{U}_5$ receives 2.87, 1.16, 3.15, 2.18, and 2.98 reputation score respectively.

Temporal Activity Model (TAM): We propose a temporal activity model (TAM) that learns the pattern of user request activity based on time of arrival from activity-logs. The model is generated using previous t-days server activity-logs. The selected activity-logs can be either previous t consecutive days, or t specific days (as defined in the policy). The temporal model can be updated by *aging* the older activity models (see Figure 2). The red rectangular blocks in Figure 3(B) represent an activity cluster per user. The term *active* in practice can represent a user session or concurrent requests. When a user request $\mathcal{U}$ arrives at the server, the server finds the corresponding user activity cluster ($\mathcal{U}_{CLS}$) formed by the temporal activity model. The user activity cluster ($\mathcal{U}_{CLS}$) is a list of time intervals that represents the user's historical activity times. The deviation in time is calculated as the distance between the two nearest clusters. From CIC-IDS2017 dataset, the cluster formed for user $\mathcal{U}_1$ shows that the user was active between $130-140$ minutes, $160-170$ minutes, $600-670$ minutes, and $720-760$ minutes. When user $\mathcal{U}_1$ arrived at time 700 minutes on day 6, the two nearest clusters are $600-670$ and $720-760$ (see Figure 3(B)). This deviation is called Δ_{local} which is the distance between the two nearest clusters. Finally, the context score for TAM is calculated as,

$$\beta = \frac{\Delta_{local}}{\Delta_{max}} \times 10 \tag{4}$$

where, Δ_{max} represents the maximum deviation possible which in our implementation is 720 minutes. Note that no cluster is found for $\mathcal{U}_5$, hence the context score calculates is the highest in range.

Flow-level Model (FLOW): Flow-level Model (FLOW) learns network flow context patterns from activity-logs. The network flow attributes of a request packet are flow-related data, such as TTL, flow duration, payload size, protocol, etc. To generate the model, the n-dimensional flow attribute vectors are projected in Euclidean space. In Figure 3(C), the green dots represent projected network flow attributes of legitimate requests, and the red dots represent projected network flow attributes of malicious requests. When a new request is seen, its flow-level attributes are projected and the Euclidean distance to malicious and legitimate clusters is computed. The context score is calculated as,

$$\gamma = \frac{\Delta_{l,m}}{\Delta_{max}} \times 10 \tag{5}$$

where, $\Delta_{l,m}$ is the deviation from malicious and legitimate clusters and Δ_{max} is the maximum deviation possible in flow-level context.

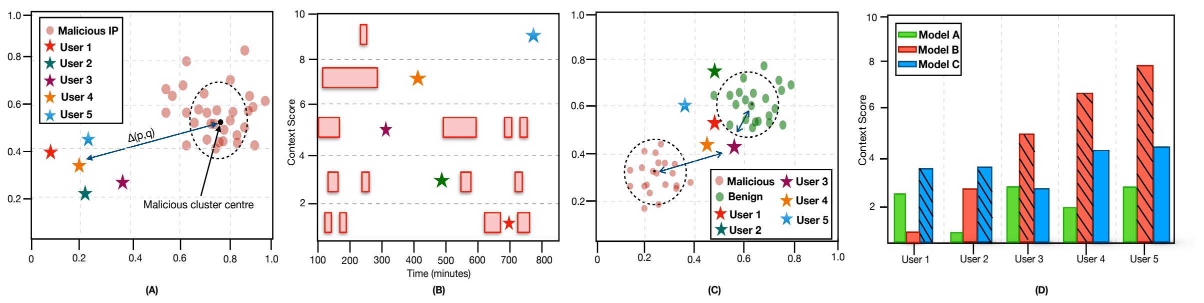


Figure 3: The figure contains four sub-figures. (A) Representation of trained DAbR in the 2-D plot. The red dot cluster represents malicious IP attributes. (B) Representation of trained TAM. The stars represent the current time of arrival. (C) Representation of FLOW. The green cluster represents legitimate flow-level attributes and the red cluster represents malicious ones. (D) Represents the calculated context score after combining scores from Model A is DAbR, Model B is TAM, and Model C is FLOW.

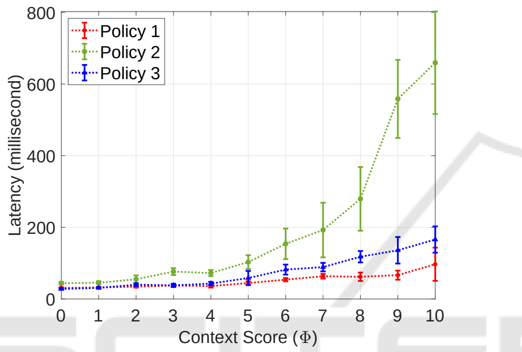


Figure 4: An evaluation of our three implemented policies. The median of 30 trials is reported for each reputation score.

4.3 Policy Component Instance

We constructed three policy instances, *policy* 1, *policy* 2, and *policy* 3. These policies only set the mapping function between context scores to the PoW puzzle difficulty level. Context score is directly proportional to the difficulty of the PoW puzzle, such as the increase in contextual deviation leads to a higher difficulty puzzle and more latency injected.

Policies 1 and 2: Linear Mapping. Assume a linear map function. Policy 1 maps $f(\Phi) \rightarrow d$, where $\Phi \in [0,10]$ is the range of context score and $d \in [0,10]$ is the difficulty levels of the PoW puzzle. Similar to policy 1, policy 2 maps $f(\Phi) \rightarrow d$, where $\Phi \in [0,10]$ and $d \in [10,20]$. Note that, the error bar in Figure 4 shows the discrepancy in time to solve d-level PoW puzzle. As discussed in Section 3.3, this discrepancy in time to solve can be avoided by using memory-bound PoW puzzles.

Policy 3: Error Range Mapping. For policy 3, we incorporated the error ε of the context-aware AI model. Assume a linear map function. Policy 3 maps $f(\Phi) \rightarrow d$, where $\Phi \in [0,10]$ and $d \in [0,10]$. The final difficulty level assigned is a difficulty value chosen at random in the interval $[\lceil d_i - \varepsilon \rceil, \lceil d_i + \varepsilon \rceil]$, where $\varepsilon = 0.2$. Figure 4 shows that as contextual deviation increases, the amount of injected latency increases.

4.4 PoW Instance – Hash Function

We discuss two sub-components of CAPOW that mimic proof-of-work system: *puzzle solver*, and *puzzle verifier*.

Puzzle Solver. The puzzle solver takes user identifiers as input, such as the timestamp of the arrival of the request packet (t), and the user IP address (u). Additionally, the solver takes a server seed value (ρ) to protect against pre-computational attacks. To this, a n-bit string is added, which the client modifies upon each hash function evaluation. We call this string *nonce* denoted by η.

The user evaluates this input until it finds an output string Y where $Y = H(u||t||\rho||\eta)$ with d leading zeroes, where d is the difficulty level assigned to the request packet. The puzzle solver is a user-end component that is installed either in the browser (Le et al., 2012) or kernel-level. After solving, the user sends the nonce back to the server for verification.

Puzzle Verifier. Puzzle verification is a server-side component that performs straightforward verification of the puzzle solution by performing one hash evaluation, i.e., $Y' = H(u||t||\rho||\eta)$. If the sent η value leads to the desired number of leading 0's, then the solution is verified.

Summary of CAPOW implementation Evaluation. The context scores produced by DAbR, TAM, and FLOW models are combined to produce the final context score (Φ). As discussed in Section 3.3, some contexts might be more relevant than others to provide an attack-specific defense. We denote weight w as the significance of each context in the final context score.

The weights for each AI model are fixed through the policy instance as discussed in Section 4.3.

$$\Phi = \arg\max(w_1\alpha, w_2\beta, w_3\gamma) \quad (6)$$

where w_1, w_2, and w_3 represent weights associated with DAbR, TAM, and FLOW respectively. Figure 3(D) illustrates the combined context score where w_1, w_2, and w_3 is set to 1. User $\mathcal{U}_1$ and $\mathcal{U}_2$ show that the final context score is decided by FLOW model. Similarly, $\mathcal{U}_3, \mathcal{U}_4$, and $\mathcal{U}_5$ the final score is decided by TAM model. Using policy 2, user $\mathcal{U}_5$ incurs $\approx$ 300ms latency for a context score of 8, which is the highest latency amongst other users introduced by CAPOW.

Notably, the evaluation performed using a simulated dataset might not reflect the worst case efficiency of CAPOW as in practice, user $\mathcal{U}_5$ might not have deviated in a temporal activity context. In this section, we discuss that the cost of deceiving multiple AI models is expensive for the adversary. In our implementation, user $\mathcal{U}_5$ has to deceive three AI models to receive an easy PoW puzzle by receiving lower context scores. User $\mathcal{U}_5$ can receive a lower context score for DAbR by trivially spoofing the IP address (Assumption 2). To deceive TAM, the user can engineer the requests around the same time as noticed during eavesdropping (Assumption 1). As reading or tracking flow-level data embedded in request payload data while eavesdropping is not possible (Assumption 1), the only way to deceive FLOW is by sending multiple probe packets to land on a low context score. This is an extensive approach as security personnel may select new contexts to improve the defensive posture of the organization periodically. Therefore, deceiving all AI models becomes expensive for the adversary. To validate contribution 3, we designed and evaluated an implementation instance on CAPOW and provided policy designs to validate contribution 2. Finally, CAPOW ensures that malicious users incur higher latency than legitimate users based on the deviation in context pattern. Hence, we validate contribution 1 (see Section 1).

5 DISCUSSION AND LIMITATION

In this section, we discuss some concerns regarding the use of proof of work in non-blockchain networks. In our framework, we assume that user behavior is defined as a range, instead of a binary demarcation of legitimate and malicious behavior. This assumption provides a “metaphorical knob” to tune the latency introduced as dictated by the range of behavior. One may claim that a straightforward solution to DDoS should be to detect an abnormally behaved user packet and drop it. There exist state-of-the-art intrusion detection systems (IDS) and packet classification algorithms (Doriguzzi-Corin et al., 2020; Ndibwile et al., 2015a; Yuan et al., 2017) that work by identifying malicious signatures. However, the classification task becomes challenging in scenarios where packets are encrypted or when an evolving adversary carefully crafts the packet to appear legitimate. Differentiating legitimate and malicious user is challenging since the clientele often enjoys a large degree of anonymity, and face little-to-no admission control for using a service. This paper aims to build and design a DDoS defense framework that adaptively introduces latency as complementary to a range of user behavior.

Building an effective PoW framework for DDoS defense is a non-trivial task due to the following reasons: (1) implementation of user and server processes that can be integrated at browser-level or OS-level; (2) building and training an appropriate AI model specific to our context; and (3) evaluate the framework against faithful to real-world DDoS attacks. Through our work, we provide (1) design principles and preliminary implementation of client and server processes integrable at the browser-level, (2) we utilized three AI models for detecting deviation as defined in our context, and (3) we provide preliminary evaluation using CICIDS-2017 dataset. We see our preliminary evaluation as a step towards extensive real-time extensive evaluation.

Here, we provide justification to common questions regarding CAPOW framework.

Question 1: Can an adversary DDoS the CAPOW framework itself by sending exorbitant amount of request?

Answer: It is possible that the adversary arbitrarily sends numerous requests to CAPOW, without intending to solve the puzzle. In Section 2, we stated that the CAPOW disregards any users that do not solve the assigned puzzle. Hence, it is not possible for an adversary to DDoS the framework itself.

Question 2: Can the adversary drive up the puzzle difficulty for the legitimate user through spoofing?

Answer: The puzzle difficulty is issued as complimentary to the deviation in context. The deviation in context is calculated through AI models. As stated in Section 2 assumption 3, the models are trained using unpolluted activity logs. Hence, it is not possible for the adversary to affect the puzzle difficulty for legitimate users.

6 RELATED WORKS

In this section, we discuss the overview of proof-of-work (PoW) literature in DDoS. Relevant to our work, we will also discuss the current advances in AI-assisted cybersecurity.

6.1 Classical Proof-of-Work

Dwork et. al (Dwork and Naor, 1992) coined the term proof-of-work (PoW) when they proposed the use of cryptographic hash functions (also known as client puzzles) to combat unsolicited bulk emails (junk emails). Following that, Franklin et. al (Franklin and Malkhi, 1997) proposed a lightweight website metering scheme in 1997 to prevent fraudulent web server owners from inflating their website's popularity. In 1999, Jakobsson et. al (Jakobsson and Juels, 1999) proposed MicroMinting (originally proposed by Rivest et. al (Rivest and Shamir, 1996) as a digital payment scheme) as a candidate problem that can reuse the computational effort of solving the POW puzzle. Later that year, Laurie et. al (Laurie and Clayton, 2004) proposed that proof of work does not work in a spam setting.

6.2 Proof-of-Work as DoS Defense

Similar to spam emails, in DDoS, it is significantly cheaper for the attacking party to launch a DDoS attack than to defend an infrastructure with the defending party. According to Arbor network, launching a DoS attack costs an average of $66 per attack and can cause damage to the victim of around $500 per minute (Lynch, 2017). Aura et. al (Aura et al., 2000) proposed the first client puzzle authentication protocol for a DoS resilient system. Mankins et. al (Mankins et al., 2001) investigated methods for tuning the amount of resource consumption to access server resources based on client behavior, where the costs imposed can be either monetary or computational. In a similar vein, Wang and Reiter (Wang and Reiter, 2003) investigate how clients can bid on puzzles through auctions. Ndibwile et. al (Ndibwile et al., 2015b) proposed web traffic authentication as a replacement for CAPTCHA-based defenses. Wu et. al (Wu et al., 2015) proposed a software puzzle framework that disqualifies the adversary's ability to gain an advantage by using a GPU to solve puzzles. A framework was put forth by Dean et. al (Dean and Stubblefield, 2001) to reduce DoS in TLS servers. A DoS variant was introduced by Wood et. al (Wood et al., 2015). Certain PoW defenses against DoS are layer-specific. The network layer of the proof-of-work system used by Parno et. al (Parno et al., 2007) prioritizes users who use more CPU time to solve puzzles. The Heimdall architecture, which can detect any change in network flow in routers, was introduced by Chen et. al (Chen et al., 2010). When a change in network flow is identified for any new connection, a puzzle is generated and sent to the new user. The difficulty of the computational challenges used in the context of DoS attacks on the transport layer was recently assessed using game theory by Noureddine et. al (Noureddine et al., 2019). Walfish et. al (Walfish et al., 2006) propose an alternative resource called communication capacity as a defense against application-layer flood attacks. Other research has concentrated on incorporating PoW puzzles into practical browsing experiences (Le et al., 2012; Kaiser and Feng, 2008; Pandey and Pandu Rangan, 2011; Chakraborty et al., 2022b; Chakraborty et al., 2022a).

6.3 Automated DoS Defense

In this section, we revisit the literature on ensemble learning techniques for network traffic classification problems. Ensemble learning is a branch of supervised ML technique that aggregates the learning of multiple known or new ML algorithms to improve overall prediction accuracy (Polikar, 2006). In ensemble learning literature, each ML model is referred to as base learners (Mienye and Sun, 2022). Each base learner is trained to become an expert in the local area of the total feature space. Gaikwad et al. (Gaikwad and Thool, 2015) proposed a bagging ensemble approach using REPTree (Elomaa and Kaariainen, 2001) base learner to improve classification over weaker AI models. Gupta et al. (Gupta et al., 2022) suggested an IDS system that uses ensemble learning to address a class imbalance problem. The ensemble learner uses three base learners, (1) the deep neural network (Gupta et al., 2022) classifies normal and suspicious traffic, (2) extreme gradient boosting (Chen and Guestrin, 2016) is used to identify major attacks, (3) random forest (Breiman, 2001) is used to classify minor attacks. Zhou et al. (zho, 2020) proposed a feature selection process using ensemble learning in two stages. The first stage involves feature reduction using the heuristic method CFS and the bat algorithm (Yang, 2010). The second stage involves aggregating C4.5 and Random Forest (RF) algorithms. Jabbar et al. (Jabbar et al., 2017) suggested an ensemble classifier that uses Alternating Decision Tree (ADTree) and the k-Nearest Neighbor algorithm (kNN) as base AI models. Paulauskas and Auskalnis (Paulauskas and Auskalnis, 2017) proposed an ensemble learner that employs four base classifiers: J48, C5.0, Naive Bayes, and Partial Decision List (PART) to improve classification results over individual AI models.

7 CONCLUSION AND FUTURE WORK

In this paper, we design and evaluate CAPOW a context-aware AI-assisted PoW framework that protects critical servers against DDoS. The underlying defensive strategy involves adaptively introducing latency on malicious users. To achieve this functionality, our framework employs an AI model that takes the context attributes from the incoming user request packet as input. The AI model computes deviation from normal activity patterns to output a context score. This score influences the difficulty level of a PoW puzzle that injects latency adaptively during communication. CAPOW ensures that the ability of a malicious user to prolong the attack is constrained by adaptively introducing latency through PoW puzzles and compelling malicious users to expend more resources to complete an attack.

For future work, different design variants of CAPOW can be configured to combat different DDoS attack types. It will be interesting to see when the PoW component is replaced by the proof of stake (PoS) component to circumvent the inherent pitfalls of the former. Additionally, an alternate design can include an enhanced human-in-loop strategy which provides control of the framework to the security personnel deploying the framework.

ACKNOWLEDGEMENT

We would like to thank Dr. Milan Parmar for his review and suggestions to improve this paper. This work was supported in part by the National Science Foundation (NSF) grant CNS-2210300.

REFERENCES

(2020). Building an efficient intrusion detection system based on feature selection and ensemble classifier. *Computer Networks*, 174:107247.

Aura, T., Nikander, P., and Leiwo, J. (2000). Dos resistant authentication with client puzzles. In *Revised Papers from the 8th International Workshop on Security Protocols*, page 170177, Berlin, Heidelberg. Springer-Verlag.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Chakraborty, S. M. . T. (2022). Github. Online Website. https://github.com/trishac97/CAPoW.

Chakraborty, T., Mitra, S., Mittal, S., and Young, M. (2022a). Ai_adaptive_pow: An ai assisted proof of work (pow) framework for ddos defense. *Software Impacts*, 13:100335.

Chakraborty, T., Mitra, S., Mittal, S., and Young, M. (2022b). A policy driven ai-assisted pow framework. *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, pages 37–38.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. KDD '16. Association for Computing Machinery.

Chen, Y., Ku, W., Sakai, K., and DeCruze, C. (2010). A novel ddos attack defending framework with minimized bilateral damages. In *2010 7th IEEE Consumer Communications and Networking Conference*, pages 1–5.

Dean, D. and Stubblefield, A. (2001). Using client puzzles to protect TLS. In *10th USENIX Security Symposium (USENIX Security 01)*, Washington, D.C. USENIX Association.

Dey, A. K. (2001). Understanding and using context. page 47.

Doriguzzi-Corin, R., Millar, S., Scott-Hayward, S., Martinez-del Rincon, J., and Siracusa, D. (2020). LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Transactions on Network and Service Management*, 17(2):876–889.

Dwork, C. and Naor, M. (1992). Pricing via processing or combatting junk mail. CRYPTO '92, page 139147, Berlin, Heidelberg. Springer-Verlag.

Elomaa, T. and Kaariainen, M. (2001). An analysis of reduced error pruning. *Journal of Artificial Intelligence Research*, 15:163–187.

Franklin, M. K. and Malkhi, D. (1997). Auditable metering with lightweight security. In *Proceedings of the First International Conference on Financial Cryptography*, FC '97, page 151160, Berlin, Heidelberg. Springer-Verlag.

Gaikwad, D. and Thool, R. C. (2015). Intrusion detection system using bagging ensemble method of machine learning. In *2015 International Conference on Computing Communication Control and Automation*, pages 291–295.

Green, J., Juen, J., Fatemieh, O., Shankesi, R., Jin, D., and Gunter, C. A. (2011). Reconstructing hash reversal based proof of work schemes. USA. USENIX Association.

Gupta, D., Saia, J., and Young, M. (2020). Resource burning for permissionless systems. In *International Colloquium on Structural Information and Communication Complexity*, pages 19–44. Springer.

Gupta, N., Jindal, V., and Bedi, P. (2022). Cse-ids: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems. *Computers & Security*, 112:102499.

Jabbar, M. A., Aluvalu, R., and Reddy, S. S. S. (2017). Cluster based ensemble classification for intrusion detection system. New York, NY, USA. Association for Computing Machinery.

Jakobsson, M. and Juels, A. (1999). Proofs of work and bread pudding protocols (extended abstract).

Juels, A. and Brainard, J. (1999). Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, pages 151–165.

Kaiser, E. and Feng, W.-c. (2008). mod kapow: Protecting the web with transparent proof-of-work. In *IEEE INFOCOM Workshops 2008*, pages 1–6. IEEE.

Laurie, B. and Clayton, R. (2004). Proof-of-work" proves not to work.

Le, T., Dua, A., and Feng, W.-c. (2012). kapow plugins: protecting web applications using reputation-based proof-of-work. In *2nd Joint WICOW/AIRWeb Workshop on Web Quality*, pages 60–63.

Lynch, V. (2017). Everything you ever wanted to know about dos/ddos attacks. https://www.thesslstore.com/blog/everything-you-ever-wanted-to-know-about-dosddos-attacks/.

Mankins, D., Krishnan, R., Boyd, C., Zao, J., and Frentz, M. (2001). Mitigating distributed denial of service attacks with dynamic resource pricing. In *Proceedings of the Seventeenth Annual Computer Security Applications Conference*, pages 411–421. IEEE.

Mienye, I. D. and Sun, Y. (2022). A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *IEEE Access*, 10:99129–99149.

Ndibwile, J. D., Govardhan, A., Okada, K., and Kadobayashi, Y. (2015a). Web server protection against application layer DDoS attacks using machine learning and traffic authentication. In *Proceedings of the 39th IEEE Annual Computer Software and Applications Conference*, pages 261–267.

Ndibwile, J. D., Govardhan, A., Okada, K., and Kadobayashi, Y. (2015b). Web server protection against application layer ddos attacks using machine learning and traffic authentication. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 3, pages 261–267.

Noureddine, M. A., Fawaz, A. M., Hsu, A., Guldner, C., Vijay, S., Başar, T., and Sanders, W. H. (2019). Revisiting client puzzles for state exhaustion attacks resilience. In *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 617–629.

of New Brunswick, U. (2017). Intrusion detection evaluation dataset (cic-ids2017). Online Website. https://www.unb.ca/cic/datasets/ids-2017.html.

Orlowski, A. (2016). Meet ddosaas: Distributed denial of service-as-a-service. Online Website. https://www.theregister.com/2016/09/12/denial_of_service_as_a_service/.

Ozdel, S., Damla Ates, P., Ates, C., Koca, M., and Anarm, E. (2022). Network anomaly detection with payload-based analysis. In *2022 30th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4.

Pandey, A. K. and Pandu Rangan, C. (2011). Mitigating denial of service attack using proof of work and token bucket algorithm. In *IEEE Technology Students' Symposium*, pages 43–47.

Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B., and Hu, Y.-C. (2007). Portcullis: Protecting connection setup from denial-of-capability attacks. SIGCOMM '07, page 289300, New York, NY, USA. Association for Computing Machinery.

Paulauskas, N. and Auskalnis, J. (2017). Analysis of data pre-processing influence on intrusion detection using nsl-kdd dataset. In *2017 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–5.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45.

Renjan, A., Joshi, K. P., Narayanan, S. N., and Joshi, A. (2018). Dabr: Dynamic attribute-based reputation scoring for malicious ip address detection. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 64–69. IEEE.

Rivest, R. L. and Shamir, A. (1996). Payword and micromint: Two simple micropayment schemes. In *Security Protocols Workshop*.

Talos, C. (2022). Talos threat source. https://www.talosintelligence.com/.

Walfish, M., Vutukuru, M., Balakrishnan, H., Karger, D., and Shenker, S. (2006). DDoS Defense by Offense. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 303–314.

Wang, X. and Reiter, M. K. (2003). Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 78–92.

Waters, B., Juels, A., Halderman, A., and Felten, E. (2004). New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 246–256.

Wood, P., Gutierrez, C., and Bagchi, S. (2015). Denial of service elusion (dose): Keeping clients connected for less. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, pages 94–103.

Wu, Y., Zhao, Z., Bao, F., and Deng, R. H. (2015). Software puzzle: A countermeasure to resource-inflated denial-of-service attacks. *IEEE Transactions on Information Forensics and Security*, 10(1):168–177.

Yang, X.-S. (2010). A new metaheuristic bat-inspired algorithm.

Yuan, X., Li, C., and Li, X. (2017). DeepDefense: Identifying DDoS attack via deep learning. In *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8.