Using users as temporary CDN nodes: an application-layer DDoS defense method

Zhitao Wu

Hangzhou Dianzi University

Abstract: For application-layer DDoS attacks, we propose a defense method that utilizes user devices as temporary CDN nodes. For instance, when a user accesses a webpage via HTTP/HTTPS protocols, the server segments the webpage or its content (such as HTML, CSS, images, etc.) into smaller data blocks and returns only one block to the user at a time. To retrieve the next data block, the user must complete a task: forwarding the already received data block to N other users. Only after these N users confirm receipt of the data block will the server return the next data block to the original user. This approach reduces the resource demand on the server while increasing the resource requirements for attackers.

Keywords: DDoS, CDN, application-layer.

1. Introduction

For application-layer DDoS attacks, we propose a defense method that utilizes user devices as temporary CDN nodes. For instance, when a user accesses a webpage via HTTP/HTTPS protocols, the server segments the webpage or its content (such as HTML, CSS, images, etc.) into smaller data blocks and returns only one block to the user at a time. To retrieve the next data block, the user must complete a task: forwarding the already received data block to N other users. Only after these N users confirm receipt of the data block will the server return the next data block to the original user. This approach reduces the resource demand on the server while increasing the resource requirements for attackers.

2. DDoS type

DDoS (Distributed Denial of Service) attacks refer to malicious activities in which a large number of distributed devices send massive amounts of traffic or requests to a target server, network, or service, rendering it unable to function properly. Based on the methods of execution, DDoS attacks can be categorized into the following main types.

The first type of DDoS attack is volume-based attacks. These attacks aim to overwhelm the target's network bandwidth or server resources with a large volume of traffic, with the primary goal of exhausting bandwidth. Common types of volume-based attacks include the following. UDP Flood: Attackers send a large number of UDP packets to the target server, forcing it to expend resources processing these unnecessary packets, ultimately leading to resource exhaustion. ICMP Flood / Ping Flood: Attackers send a massive number of ICMP echo requests (ping requests) to the target, consuming the target's bandwidth and computational capacity. DNS Amplification Attack: By leveraging open DNS resolvers, attackers spoof the source IP to appear as the target server's IP and send small query packets. This triggers the DNS servers to generate large response traffic, which is then directed to the target server. NTP Amplification Attack: Exploiting the MONLIST command in the Network Time Protocol (NTP), attackers send small requests that are amplified into significantly larger response traffic, which is then sent to the target. HTTP Flood: Attackers send a high volume of HTTP requests (e.g., GET or POST requests) to the target server, depleting the server's computational resources.

The second type of DDoS attack is protocol attacks. These attacks aim to exhaust the protocol processing capacity of the server or the resources of intermediary devices (e.g., firewalls, load balancers), rendering the system inoperative. The common types of protocol attacks are as follows. SYN Flood: Exploiting the TCP three-way handshake mechanism, attackers send a large number of SYN requests but do not complete the subsequent ACK response, causing the server's resources to be exhausted. ACK Flood: Attackers send a high volume of spoofed TCP ACK packets, consuming the target's resources. TCP Connection Exhaustion: Attackers establish a large number of half-open connections or maintain long-duration connections, occupying the server's connection resources. Ping of Death: Attackers send oversized ICMP packets (exceeding the standard MTU size), which can crash or incapacitate the target system. Fragmentation Attack: Attackers send deliberately fragmented or malformed packets (e.g., Teardrop attacks), causing the target system to crash when attempting to reassemble the packets.

The third type of DDoS attack is the application-layer attack. These attacks target specific application-layer protocols or services, exploiting their vulnerabilities or design characteristics to exhaust system resources. Common types include the following. HTTP Flood: Attackers send a large volume of legitimate HTTP requests (e.g., GET or POST requests), simulating real user behavior to make the attack difficult to detect. Slowloris: Attackers send partial HTTP requests and keep the connection open, occupying the target server's connection pool resources and preventing other legitimate users from accessing the server. DNS Query Flood: Attackers send a large number of legitimate DNS query requests, rendering the target DNS server unable to respond to legitimate requests from other users. SMTP Attack: Attackers send a massive number of forged email requests to a mail server, causing the email service to become unavailable. SSL/TLS Exhaustion Attack: Attackers maliciously send a large number of SSL/TLS handshake requests, consuming the target server's computational resources, as SSL/TLS handshakes are more resource-intensive than regular connections.

3. DDoS defense method based on temporary CDN nodes

3.1 Details

For application-layer DDoS attacks, we propose a defense method that utilizes user devices as temporary CDN nodes. For instance, when a user accesses a webpage using the HTTP/HTTPS protocol, the server splits the webpage or its content (such as HTML, CSS, images, etc.) into smaller data blocks and returns only one block at a time to the user. To request the next data block, the user must complete a task: forwarding the received data block to another N users. Only after those N users confirm receipt of the data block will the server return the next data block to the original user.

Suppose a user with the address ip_0 requests data a, where the size of the data is size(a). Let the size of each transmitted data packet be size(m). Then, the total number of packets required to split data a is $\mathrm{k} = \mathrm{ceil}(\mathrm{size(a)}/\mathrm{size(m)})$. The set of packets resulting from the split of data a is denoted as $\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{k}\}$. We compute the hash values of each packet, forming the hash set $\mathrm{H} = \{\mathrm{h}_1, \ldots, \mathrm{h}_\mathrm{k}\}$. The server can then send packet a_1 directly to the user at address ip_0, or it can instruct other users who have already received packet a_1 (such as $\mathrm{ip}_{\mathrm{other}}$) to forward it to the user at ip_0. Once the user at ip_0 receives packet a_1, they calculate its hash value and return it to the server. The server verifies the hash for correctness; if the hash is incorrect, the packet must be resent using the aforementioned procedure.

When the user with IP address ip_0 receives the data packet a_1, the server sends ip_0 a list of user IPs that require forwarding, denoted as $\mathrm{ip}_{\mathrm{task}} = \{\mathrm{ip}_1, \ldots, \mathrm{ip}_\mathrm{N}\}$. Once these users receive the data packet a_1, they each send a hash value back to the server. If the user ip_0 successfully completes the forwarding task for all N users, the server then begins to send the next data packet, a_2. Following the same process, the transmission of all k data packets in dataset A is completed step by step.

In the case of a Distributed Denial-of-Service (DDoS) attack, a large number of users are not legitimate users but are instead botnet hosts controlled by the attacker. When one of the attacker's botnet hosts (e.g., user ip_0) receives the data packet a_1, it calculates the hash value h_1 of the packet. The server then instructs the botnet host ip_0 to forward the data packet a_1 to N other users, denoted as $\mathrm{ip}_{\mathrm{task}} = \{\mathrm{ip}_1, \ldots, \mathrm{ip}_\mathrm{N}\}$. However, most of the users in $\mathrm{ip}_{\mathrm{task}} = \{\mathrm{ip}_1, \ldots, \mathrm{ip}_\mathrm{N}\}$ are also botnet hosts controlled by the attacker, with only a minority being legitimate users. For the botnet hosts in $\mathrm{ip}_{\mathrm{task}}$, the attacker can bypass the actual transmission of the data packet a_1 and instead directly send the hash value h_1, thereby avoiding internal network consumption within the botnet. For the legitimate users in $\mathrm{ip}_{\mathrm{task}}$, however, the attacker is still required to forward the data packet a_1.

For a server, instead of directly sending data packet a_1 to all users, it only needs to forward the packet to a small subset of users. The remaining users can forward the packet to each other, with the server acting as a coordinator. Whether users genuinely forward data packet a1 to others (rather than forwarding its hash value h_1) has no impact on the server's resource consumption. For an attacker, the more machines under their control in the forwarding list $\mathrm{ip}_{\mathrm{task}} = \{\mathrm{ip}_1, \ldots, \mathrm{ip}_\mathrm{N}\}$, the fewer data packets a_1 they need to forward in reality.

Next, we discuss the second round of forwarding. At this point, all users in $\mathrm{ip}_{\mathrm{task}} = \{\mathrm{ip}_1, \ldots, \mathrm{ip}_\mathrm{N}\}$ are assumed to have received data packet a_1 (if controlled by the attacker, they may have only received the hash value h_1). Let us denote one specific user in $\mathrm{ip}_{\mathrm{task}}$ as ip_i, who is a machine controlled by the attacker. In this case, ip_i only receives the hash value h_1. The server then assigns ip_i a new forwarding task for the second round, requiring it to forward the packet to N users in order to receive the next data packet a_2. The task list assigned to ip_i is denoted as $\mathrm{ip}^2_{\mathrm{task}} = \{\mathrm{ip}^2_1, \ldots, \mathrm{ip}^2_\mathrm{N}\}$, where some of the targets in $\mathrm{ip}^2_{\mathrm{task}}$ are legitimate users. This means that ip_i must send data packet a_1 to these legitimate users. However, since ip_i only possesses the hash value h_1, it cannot complete this task. As a result, for the attacker, they are left with no choice but to forward all data packets accurately. After several rounds of forwarding, all compromised machines (often referred to as "bots") will gradually fail to complete their tasks, rendering them unable to receive subsequent data packets.

Under non-attack scenarios, the verification mechanism can be disabled or its requirements relaxed (e.g., reducing the number of forwarding targets) to improve user experience. Under attack scenarios, the server can dynamically increase the difficulty of forwarding tasks (e.g., increasing the value of N) to impose greater pressure on attackers.

Process 1. Pseudocode of the server

```
Input: A user at address $\mathrm{ip}_0$ requests resource a (with size size(a)); the size of each transmitted data packet is size(m); the list of forwarding users is $\mathrm{ip}_{\mathrm{task}}$
Output: Data packets of resource a are sent to the user at address $\mathrm{ip}_0$
Split resource a into multiple data packets, where the size of each packet is size(m). The total number of packets is calculated as $\mathrm{k} = \mathrm{ceil}(\mathrm{a}/\mathrm{size(m)})$. The set of data packets is denoted as $\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{k}\}$.
Compute the hash values for the set of data packets A, resulting in a hash set $\mathrm{H} = \{\mathrm{h}_1, \ldots, \mathrm{h}_\mathrm{k}\}$.
The server sends the basic information of the data packets to the user at address $\mathrm{ip}_0$, including the size of each packet size(m) and the total number of packets k.
for (each data packet $\mathrm{a}_\mathrm{i} \in \mathrm{A}$):
    The server sends the data packet $\mathrm{a}_\mathrm{i}$ to user $\mathrm{ip}_0$, or instructs another user $\mathrm{ip}_{\mathrm{other}}$ to send the data packet $\mathrm{a}_\mathrm{i}$ to user $\mathrm{ip}_0$.
    Wait for user $\mathrm{ip}_0$ to send back the hash value $\mathrm{hi}_{\mathrm{send}}$ of
```

```
data packet $a_i$.
If($hi_{send} \neq h_i$):
    If(the data packet was sent directly by the server to user $ip_0$):
        The server retransmits the data packet $a_i$. A maximum retransmission count is set; if the count is exceeded, no further transmissions are made.
    If(the data packet was sent by another user $ip_{other}$ to user $ip_0$):
        The server instructs $ip_{other}$ to retransmit the data packet. A maximum retransmission count is set; if the count is exceeded, the server sends the data packet $a_i$ directly to $ip_0$. If the retransmitssion count is exceeded again, no further transmissions are made.
The server then sends a task list $ip_{task}[i][N] = \{ip_1^i, \dots, ip_N^i\}$ to user $ip_0$, requiring user $ip_0$ to send the data packet $a_i$ to any user in the list.
The server waits for each user in the task list $ip_{task}[i][N]$ to send back the received hash values of the data packet, $H = \{h_i^1, \dots, h_i^N\}$.
for j in 1:N:
    If($h_j \neq h_i, h_j \in H$):
        The server requires user $ip_0$ to retransmit the data packet $a_i$ to user $ip_j$. If the retransmission count exceeds the maximum, the server assigns a new task, requiring user $ip_0$ to send the data packet to a new user $ip_{new}$.
        If the number of task switches exceeds the limit, the user is required to take PoW(Prove of Work) task[1]. If this cannot be completed, subsequent data packets $a_{i+1}$ will no longer be sent.
If(the server does not receive the hash values $hi_{send}$ from all users in the task list $ip_{task}[i][N]$ within the timeout period):
    The server requires user $ip_0$ to retransmit the data packet $a_i$ to user $ip_j$. If the retransmission count exceeds the maximum, the server assigns a new task, requiring user $ip_0$ to send the data packet to a new user $ip_{new}$.
    If the number of task switches exceeds the limit, the user is required to take PoW(Prove of Work) task[1]. If this cannot be completed, subsequent data packets $a_{i+1}$ will no longer be sent.
```

Process 2. Pseudocode of the client

```
Input: Server address $ip_{server}$
Output: Received data packet a
Request data packet a from the server at $ip_{server}$.
Retrieve basic information about the data packets: the size of each data packet, size(m); the total number of data packets, k.
for i=1:k
    Receive data packet $a_i$, compute its hash value $hi_{send}$, and send the hash value $hi_{send}$ to the server at $ip_{server}$.
    If(the server responds with a data packet reception error):
        Prepare to re-receive data packet $a_i$. If the maximum number of retransmissions is exceeded, the reception fails.
    Receive the task list $ip_{task}[i][N] = \{ip_1^i, \dots, ip_N^i\}$, and forward data packet $a_i$ to the users in the task list.
    If a task transmission fails, such that the hash value $h_j$ computed by certain users in the list $ip_{task}[i][N]$ for the received data packet $a_i$ differs from the server's hash value $h_i$. Perform the retransmission tasks as requested by the server.
    If(the next data packet $a_{i+1}$ is not received within the specified waiting time):
        break
```

3.2 Resource consumption of the server

We provide a detailed calculation of server resource consumption. Server resource consumption can be categorized into two types. The first type includes shared resources that are identical for all users, such as the website homepage and static content (CSS, JS, images, videos, etc.). These resources are the same for all users and, therefore, the server's load can be minimized by utilizing distributed transmission. The second type includes personalized resources that differ for each user, such as dynamic requests (login operations, CRUD operations, personalized data, etc.). These resources must be processed by the server and cannot be offloaded through forwarding between users to reduce resource consumption.

The optimization of the first type of resources can be achieved by constructing a distributed CDN using user devices. The theoretical optimal solution under extreme conditions is as follows: theoretically, the server only needs to send a single copy of the shared resource. Assume user ip_0 is the first visitor, and the server sends the shared resource to user ip_0. User ip_0 then forwards the content to N other users (users $ip_1, \dots, ip_N$). Subsequently, these N users forward the content to the next layer of N^2 users. Through this chain-based distribution, the server's resource demand becomes: Server resource consumption = 1 copy of shared resources + metadata for forwarding instructions. However, this approach has the following limitations. If the distribution relies entirely on user-to-user forwarding, some users may experience excessive delays, particularly when the number of visitors surges, making the initial layers of users a bottleneck.

To reduce user waiting time, the server can proactively distribute multiple copies of public resources to create multiple "starting points" for a distributed dissemination network. This can be achieved through the following methods: First, the number of proactively sent copies can be dynamically adjusted. Based on the current number of visitors and the speed of distribution, the server can dynamically determine the number of initial copies to send. For example, if the goal is to ensure that public resources reach all users within three layers of forwarding, the server can proactively send a sufficient number of initial copies to accelerate dissemination. Additionally, a multi-origin distribution strategy can be adopted. The server selects several "distribution nodes" (early visitors) and directly sends the complete content to them. These distribution nodes are then responsible for forwarding the content to the next layer of users.

In this distributed model, the server's resource consumption is mainly reflected in the following aspects: First, there is the resource consumption for the initial distribution task. The server proactively sends several copies of the public resources to serve as the starting points for the distribution network, and this task consumes relatively little bandwidth (theoretically much less than the bandwidth required to send content directly to all users). Second, there is the resource consumption for coordination and control. The server is responsible for maintaining a forwarding network, directing each user to forward the content to specific target users. The main consumption here is the metadata overhead (e.g., forwarding instructions and node state management). Lastly, there is the resource consumption for supplementary transmission. If certain users fail to receive the content in time, the server can proactively send additional copies to ensure delivery.

For the second type of resource (user-personalized requests), since these requests must be processed by the server, the server's resource consumption cannot be reduced through distributed distribution. As a balancing mechanism, we can shift more forwarding tasks to visitors at the cost of additional effort on their part.

The analysis of total server resource consumption is as follows. Here, server resource consumption includes bandwidth resources for sending data, as well as CPU and memory resources for processing data. However, instead of measuring specific bandwidth, CPU, or memory usage, we use the amount of data processed as a proxy for resource consumption. For the first type of resource consumption, the theoretical optimal resource consumption can be calculated as follows.

$$\begin{aligned}&\text{Resource consumption (first type of resource)}\\&= 1 \text{ unit of public resource}\\&+ \text{metadata for forwarding instructions}\end{aligned}$$

In practice, by appropriately increasing the number of actively sent copies (assume sending M copies), the server's resource consumption can be expressed as:

$$\begin{aligned}&\text{Resource consumption (first type of resource)}\\&= \mathrm{M} \times \text{public resource}\\&+ \text{metadata for forwarding instructions}\end{aligned}$$

For the consumption of the second type of resource, the server must directly handle each user's dynamic request, making it impossible to reduce this part of the consumption. Assuming the number of users is U and the average resource consumption per user's dynamic request is $\mathrm{D_{req}}$, the server's resource consumption can be expressed as follows:

$$\begin{aligned}&\text{Resource Consumption (Second Type of Resource)}\\&= \mathrm{U} \times \mathrm{D_{req}}\end{aligned}$$

The total resource consumption is the sum of the first type of resource consumption and the second type of resource consumption.

3.3 Countermeasures by attackers

This DDoS defense method employs a strategy resembling a non-cooperative game in game theory between the server, legitimate users, and attackers. Below, we analyze the response strategies of attackers and the handling mechanisms of the server.

First, upon receiving a packet $\mathrm{a_i}$, the attacker must complete the forwarding task $\mathrm{ip_{task}} = \{\mathrm{ip_1.}, \ldots, \mathrm{ip_N}\}$ to obtain the next packet $\mathrm{a_{i+1}}$. If the task list $\mathrm{ip_{task}}$ contains a bot $\mathrm{ip_i}$ controlled by the attacker, the attacker can choose not to send the actual packet $\mathrm{a_i}$ but instead transmit the hash value $\mathrm{hi_{send}}$ of the packet. In the next round of forwarding, the task list for the bot $\mathrm{ip_i}$ becomes $\mathrm{ip_{task}^2} = \{\mathrm{ip_1^2}, \ldots, \mathrm{ip_N^2}\}$, which may include legitimate users. At this point, the bot $\mathrm{ip_i}$ is required to retrieve the actual packet $\mathrm{a_i}$ and forward it to the legitimate user. Otherwise, the bot $\mathrm{ip_i}$ will not be able to obtain the next packet $\mathrm{a_{i+1}}$. This approach increases the forwarding cost for attackers. After several rounds of forwarding, many bots may fail to complete their forwarding tasks and consequently drop out of the DDoS attack.

The attacker may choose not to perform the forwarding task but instead intercept only the first data packet, denoted as $\mathrm{a_1}$, and then disconnect from the server. Subsequently, the attacker can repeatedly request the same data packet $\mathrm{a_1}$ from the server. To counter this, we can impose size limitations on the initial few data packets or gradually increase the size of the data packets sent in each transmission. Furthermore, this approach can force the attacker to shift from an application-layer DDoS attack to a traffic-based DDoS attack. Such traffic-based DDoS attacks exhibit more distinct characteristics (e.g., a large number of connections from the same IP within a short period), making it possible to identify and block these malicious users effectively.

In addition, the following attack scenario also exists: for a legitimate visitor $\mathrm{ip_{normal}}$, after receiving a data packet $\mathrm{a_i}$, it must complete the forwarding task $\mathrm{ip_{task}} = \{\mathrm{ip_1}, \ldots, \mathrm{ip_N}\}$. However, the forwarding task $\mathrm{ip_{task}}$ may include a large number of compromised machines (zombies) controlled by attackers. Even if these zombies receive the data packet $\mathrm{a_i}$ sent by the legitimate visitor $\mathrm{ip_{normal}}$, they may still refuse to send the hash value $\mathrm{hi_{send}}$ of the data packet to the server (pretending they did not receive it). This behavior can obstruct the legitimate visitor $\mathrm{ip}_{\mathrm{normal}}$ from completing the forwarding task, thereby preventing $\mathrm{ip}_{\mathrm{normal}}$ from obtaining the next data packet $\mathrm{a}_{\mathrm{i}+1}$.

We propose a mechanism where legitimate visitors ($\mathrm{ip}_{\mathrm{normal}}$) are redirected to complete a Proof of Work (PoW) task after exceeding the maximum number of retransmission attempts. Once the PoW task is completed, they can still obtain the next data packet, denoted as $\mathrm{a}_{\mathrm{i}+1}$. The PoW task requires visitors to perform a computationally intensive operation before they are allowed to proceed. For example, some validation pages on Google require visitors to execute a piece of JavaScript code before gaining further access. The characteristics of this approach are as follows. Even if the attacker's botnet does not cooperate, legitimate visitors ($\mathrm{ip}_{\mathrm{normal}}$) can still obtain the next data packet ($\mathrm{a}_{\mathrm{i}+1}$) by completing the PoW task. For bots controlled by the attacker (ip_{i}), if they repeatedly fail to receive the data packet (a_{i}), the server may deny further transmission of data packets to them, effectively excluding the malicious bot (ip_{i}) from the system.

4. Traffic-based DDoS

In addition to application-layer DDoS attacks, there are also traffic-based DDoS attacks. These types of attacks overwhelm the target's network bandwidth or server resources through massive volumes of traffic, with the primary goal of exhausting bandwidth. Common types of such attacks include the following. UDP Flood: Attackers send a large number of UDP packets to the target server. The target server must allocate resources to process these unnecessary packets, leading to resource exhaustion. ICMP Flood (ICMP Flood / Ping Flood): Attackers send a large volume of ICMP echo requests (ping requests) to the target, consuming the target's bandwidth and computational capacity. DNS Amplification Attack: This type of attack exploits open DNS resolvers. By spoofing the source IP address to appear as the target server's IP, attackers send small request packets to the DNS resolvers, which then generate massive response traffic and direct it to the target server. NTP Amplification Attack: This attack leverages the MONLIST command in the Network Time Protocol (NTP). Small requests are amplified into significantly larger response traffic and sent to the target. HTTP Flood: Attackers send a large number of HTTP requests (e.g., GET or POST requests) to the target server, depleting its computational resources.

For traffic-based DDoS attacks, the losses faced by website providers include bandwidth costs and server computational resource costs. We argue that a significant portion of the bandwidth cost incurred by website providers is attributable to the bandwidth billing policies of network operators. Some network operators charge website providers for both upstream and downstream bandwidth. During a DDoS attack, website providers may experience a surge in downstream bandwidth (request data from attackers), which is beyond their control and is instead generated by the attackers. As a result, even if website providers choose not to respond to any attack requests or keep their servers shut down, they are still required to pay for the downstream bandwidth costs. In practical applications, website providers can mitigate these costs by opting for fixed-speed bandwidth plans or redirecting attack traffic to blackhole routing to limit downstream bandwidth expenses. Therefore, we suggest that a more reasonable bandwidth billing policy for network operators should involve charging only for upstream bandwidth, as upstream bandwidth is actively generated by website providers. This is analogous to the telephone billing system, where the caller is responsible for the call charges. In fact, some network operators have already adopted this approach of charging exclusively for upstream bandwidth.

If both website providers and ordinary users are charged for upstream bandwidth, the following issues may arise: ordinary users consume very little upstream bandwidth when accessing websites (only for sending requests), while the majority of their network usage involves downstream bandwidth (downloading data provided by websites). This would result in most of the network operators' fees being borne by website providers. Consequently, website providers would carefully evaluate whether responding to a user's request is worthwhile, thereby raising the threshold for information exchange on the network.

To balance network costs between website providers and ordinary users, two approaches can be considered: 1) ordinary users could be charged for both upstream and downstream bandwidth, or 2) ordinary users could be charged only for downstream bandwidth, but with restrictions placed on their upstream bandwidth rate to prevent abuse. Such abuse could otherwise result in Distributed Denial of Service (DDoS) effects that harm website providers. This is also the charging model currently adopted by most network operators.

REFERENCES

[1] K. Sung, S. Hsiao. Mitigating DDoS with PoW and Game Theory, 2019 IEEE International Conference on Big Data (Big Data).