REQSMINER: Automated Discovery of CDN Forwarding Request Inconsistencies and DoS Attacks with Grammar-based Fuzzing

Linkai Zheng †, Xiang Li †, Chuhan Wang †,
Run Guo †, Haixin Duan †§✉, Jianjun Chen †‡✉, Chao Zhang †‡, Kaiwen Shen †

†*Tsinghua University*, ‡*Zhongguancun Laboratory*, §*Quan Cheng Laboratory*

***Abstract*—Content Delivery Networks (CDNs) are ubiquitous middleboxes designed to enhance the performance of hosted websites and shield them from various attacks. Numerous notable studies show that CDNs modify a client's request when forwarding it to the original server. Multiple inconsistencies in this forwarding operation have been found to potentially result in security vulnerabilities like DoS attacks. Nonetheless, existing research lacks a systematic approach to studying CDN forwarding request inconsistencies.**

In this work, we present REQSMINER, an innovative fuzzing framework developed to discover previously unexamined inconsistencies in CDN forwarding requests. The framework uses techniques derived from reinforcement learning to generate valid test cases, even with minimal feedback, and incorporates real field values into the grammar-based fuzzer. With the help of REQSMINER, we comprehensively test 22 major CDN providers and uncover a wealth of hitherto unstudied CDN forwarding request inconsistencies. Moreover, the application of specialized analyzers enables REQSMINER to extend its capabilities, evolving into a framework capable of detecting specific types of attacks. By extension, our work further identifies three novel types of HTTP amplification DoS attacks and uncovers *74* new potential DoS vulnerabilities with an amplification factor that can reach up to *2,000* generally, and even *1,920,000* under specific conditions. The vulnerabilities detected were responsibly disclosed to the affected CDN vendors, and mitigation suggestions were proposed. Our work contributes to fortifying CDN security, thereby enhancing their resilience against malicious attacks and preventing misuse.

I. INTRODUCTION

Content Delivery Networks (CDNs) are broadly adopted by various companies and organizations to improve the performance and availability of their online content [39]. Generally, CDNs function by caching copies of content on edge servers geographically distributed all over the world. When a user requests content from a website hosted on a CDN, the request is sent to a CDN edge server that provides the lowest latency rather than the origin server.

✉ Corresponding authors:{duanhx, jianjun}@tsinghua.edu.cn.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24031
www.ndss-symposium.org

Prior research. A wealth of significant research reveals that CDNs frequently alter client requests as they relay them to the original server, thereby unveiling operational inconsistencies. These inconsistencies could potentially lead to security vulnerabilities, including the Host of Trouble (*HoT*), HTTP Request Smuggling (*HRS*), Denial of Service (*DoS*), and Cache Poisoned Denial of Service (*CPDoS*) [11], [25], [32], [50]. Concurrently, innovative attacks that exploit these inconsistencies pose substantial security threats to HTTP servers and CDNs, with HTTP/2 amplification attacks [25] and RangeAmp attacks [32] being notable examples.

Despite the widespread nature of these issues, no research to date has examined this particular class of problems from the vantage point of inconsistency in CDN forwarding requests. The majority of prior studies have relied on manual analysis or have concentrated on a single type of inconsistency [32]. As such, there is a pressing need for a systematic and extensible methodology to uncover hitherto unknown inconsistencies in CDN forwarding requests, to prevent overlooking potential security vulnerabilities.

Implementing systematic and efficient fuzzing on CDN forwarding request inconsistencies, however, poses considerable challenges: *(i)* Lax grammar. As generated templates, unbounded ABNF rules make it difficult for the fuzzer to provide valid values; *(ii)* High costs. Studying a commercial service like CDN is expensive due to massive test cases. *(iii)* Black box. Due to trade secrets, the source code of CDNs is not available to the public, causing less feedback.

Our study. In this study, we present an automated fuzzing framework—REQSMINER—which includes modules for *Rule Generator* and *Grammar-based Fuzzing*. This framework is designed to overcome existing challenges and to identify unknown inconsistencies in CDN forwarding requests. To address the first challenge, we extract valid values from RFC documents and actual HTTP traffic, storing these as field values. REQSMINER uses a *Field Values-assisted Rule Fusioner* to merge these with ABNF rules, thereby creating an ABNF grammar tree to serve as a generation template, which enhances the effectiveness of test cases. To tackle the remaining challenges, we draw inspiration from reinforcement learning, incorporating the Upper Confidence Bounds Applied to Trees with Weighted Randomization (UCT-Rand) algorithm to optimize the fuzzer. Even with limited feedback, the *UCT-based Request Generator* can efficiently generate a plethora of

valid test cases.

Utilizing REQSMINER, we systematically test 22 widely used CDN providers (e.g., Cloudflare, Akamai, CloudFront, and others) and discover a multitude of previously unexamined CDN forwarding request inconsistencies. We categorize these into three primary groups and briefly analyze whether these inconsistencies stem from the CDNs' security mechanisms and whether they can be exploited by attackers to initiate DoS, HRS, or CPDoS attacks.

As a practical security practice, we extend REQSMINER to detect DoS attacks based on amplification attacks, thereby demonstrating its scalability. By automating the detection of traffic size disparities arising from request inconsistencies, we identify 74 new vulnerabilities potentially causing amplification attacks, including three novel techniques: *(i)* HEAD Request-based HTTP Amplification Attack (*HeadAmp*); *(ii)* Conditional Request-based HTTP Amplification Attack (*CondAmp*); and *(iii)* Accept-Encoding-based HTTP Amplification Attack (*AEAmp*). With techniques such as *HeadAmp* or *CondAmp*, an attacker would only need a 1 MB file as the target resource to compel the origin server to generate response traffic *2,000* times larger than that received by the attacker. The amplification factor grows with the target resource size, reaching up to *1,920,000* times when the file size is 1 GB. Furthermore, *AEAmp* enables an attacker to initiate HTTP traffic amplification attacks with the same 1 MB target resource, achieving an amplification factor of up to *650*.

We have responsibly informed the affected CDN vendors about these vulnerabilities. At the time of writing, we have received responses from three vendors, two of which—Azure and Cloudflare—have acknowledged and rectified the vulnerabilities.

Contributions. Overall, we make the following contributions:

- *New tool.* We introduce REQSMINER, a novel detection framework engineered to automatically uncover CDN forwarding request inconsistencies. We release REQSMINER[1] through GitHub for researchers to further study CDN forwarding request inconsistencies in the future.
- *Novel fuzzing techniques.* We incorporate real field values into the grammar-based fuzzer and, for the first time, architect a UCT-based black-box fuzzing algorithm to boost efficiency.
- *Comprehensive results.* We conduct controlled tests on 22 prominent CDN vendors using REQSMINER, uncovering a multitude of unstudied CDN forwarding request inconsistencies. Additionally, we identify three new types of HTTP amplification attacks, involving 74 new DoS vulnerabilities.
- *Disclosure and mitigation.* We responsibly report all security issues to the affected CDN vendors and propose mitigation suggestions.

II. BACKGROUND

In this section, we first provide the basic concepts of content delivery networks (CDNs), briefly present HTTP standards, and finally introduce the techniques we employ, including grammar-based fuzzing and the Monte Carlo tree search algorithm.

[1] https://github.com/Konano/ReqsMiner

A. CDN Overview

A content delivery network (CDN) is a geographically distributed group of servers (nodes) that work together to provide fast delivery of Internet content, such as Cloudflare [13] and Akamai [2]. The CDN improves the performance of hosted websites and provides security protection, including a Web Application Firewall and DDoS (Distributed Denial of Service) defense. CDN is gaining widespread popularity as an essential part of the Internet's infrastructure. For example, in 2022, 64.04% of the top 10,000 popular websites improved their access quality using CDN services [9].

Request-routing Mechanism. Request-routing techniques are the key component of CDN services and determine how web admins host their websites on the CDN. One prevalent way involves redirecting the domain towards the CDN's subdomains, which are mapped to a global network of CDN edge servers. An alternative popular way entails utilizing the CDN's DNS servers as the authoritative name servers for the website's domain [5], [33]. In this paper, we only focus on these two most common techniques.

Architecture. CDN can be divided into two primary parts: *(i)* the central node is responsible for load balancing and content management; *(ii)* the edge nodes, which include ingress and egress nodes, are charged for caching and distributing content. Among the latter, ingress nodes are close to the client and handle access requests, whereas egress nodes are placed near the origin server and forward requests. As shown in Figure 1, CDN acts as a man-in-the-middle between the client and the origin server, dividing the conventional end-to-end connection into two stages: the *client-CDN* connection and the *CDN-origin* connection.

Workflows. After receiving a client request, the CDN first examines the cache for the corresponding data. In the absence of a cache, the CDN forwards the requests for required resources to the origin server and caches responses for later requests. These queries are also called "back-to-origin" requests. Through load balancing, the CDN chooses ingress and egress nodes dynamically and seeks to utilize the cache to reduce overhead on the origin server. This approach can effectively enhance access performance and defend servers against DDoS attacks by decreasing latency and load.

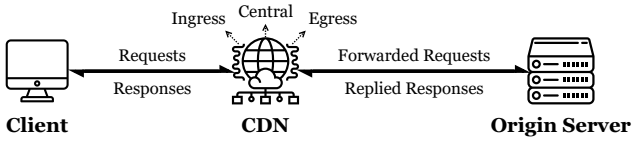


Fig. 1: CDN forwards requests and responses between client and origin.

B. HTTP Standards

The Hypertext Transfer Protocol (HTTP) standards are primarily established in RFC 9110 [19] and have a standardized protocol element format defined by ABNF rules. According to RFC 9110 [19, §2.1], HTTP protocol compliance contains a specific message syntax. An HTTP sender must generate protocol elements that conform to the grammar specified by the ABNF rules [15].

ABNF Rules. Augmented Backus–Naur Form (ABNF) is a context-free grammar (CFG) syntax used to describe the syntax of Internet protocols. Most RFCs employ ABNF to define the formal specifications of protocol messages. As shown in Listing 1, the ABNF specification for HTTP consists of a combination of derived rules formatted as "`Rule = Definition`". The left portion is the rule name, while the right is the rule's definition.

```
1  HTTP-message = start-line CRLF *( field-line CRLF ) CRLF [
       message-body ]
2  HTTP-name = %x48.54.54.50 ; HTTP
3  HTTP-version = HTTP-name "/" DIGIT "." DIGIT
4  ...
5  Accept-Charset = [ ( ( token / "*" ) [ weight ] ) *( OWS
       "," OWS ( ( token / "*" ) [ weight ] ) ) ]
6  Accept-Language = [ ( language-range [ weight ] ) *( OWS
       "," OWS ( language-range [ weight ] ) ) ]
7  ...
8  message-body = *OCTET
9  method = token
10 trailer-section = *( field-line CRLF )
11 token = 1*tchar
12 tchar = "!" / "#" / "$" / "%" / "&" / "'" / "*" / "+" /
       "-" / "." / "^" / "_" / "`" / "|" / "~" / DIGIT /
       ALPHA
```

Listing 1: ABNF Rules Defining HTTP/1.1 Message Syntax Extracted from RFCs [19], [20].

Within the `Definition` field, ABNF syntax enables multiple types of operators, including *concatenation*, *selection*, and *repetition*. Specifically, *concatenation* permits combining a list of rule names to construct a new definition by separating them with a space. In addition, a forward slash ("`/`") denotes *selection* by separating the list of optional subrules. An asterisk ("`*`"), a leading digit, or brackets ("`[]`") are *repetition* operators that limit the number of subrule repetitions.

HTTP Header Fields. HTTP uses *fields* to provide specific header data before the content. These fields are known as *header fields* (or *headers*) [19, §5]. These fields are formatted as pairs of `name:value`. An HTTP header, such as "`Host:example.com`", consists of its case-insensitive name followed by a colon ("`:`") and its values. Header names should be registered within the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [19, §3.5] [35].

C. Grammar-based Fuzzing

Fuzzing analyzes programs for errors like crashes and memory access violations by creating random test cases as input and repeatedly executing the target program [22], [52]. Grammar-based fuzzing is one of the most widely used testing techniques for identifing potential vulnerabilities in software systems. This approach employs a predefined set of rules or "grammar" to generate test inputs. By systematically generating test inputs that conform to this grammar, fuzzers can ensure a comprehensive examination of the system's behavior under various conditions. For instance, previous work utilized grammar-based fuzzing to detect protocol security vulnerabilities in critical infrastructure systems like SCADA systems [12], [26], [45], [48], [51].

D. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that uses simulations to explore potential actions and select the most promising one [1], [3], [8], [16]. It has been utilized in several applications, such as gaming, robotics, optimization, and planning [4], [27], [31], [34], [36], [47], [53]. This algorithm has four stages, *Selection*, *Expansion*, *Simulation*, and *Backpropagation*.

Upper Confidence Bounds Applied to Trees. Upper Confidence Bounds Applied to Trees (UCT) is a variant of MCTS designed to balance exploration and exploitation in game-playing AI [29], [30]. UCT uses the Upper Confidence Bounds (UCB) formula [3], a combination of the current node's value estimates and exploration parameters, to select the most promising node to expand.

$$\pi(s) := \arg\max_{a \in A(s)} \left(V_a + \sqrt{\frac{2 \ln N_s}{N_a}} \right) \tag{1}$$

The formula for UCB is shown in Formula 1, where s denotes the current state and $A(s)$ denotes the nodes that can be reached. The formula has two terms: the estimated mean value of a node and a term reflecting the uncertainty or exploration of that node. The mean value is estimated by averaging the rewards gained from simulations of the node (V_a). Meanwhile, the exploration term is calculated by the number of times the node has been visited (N_a) and state s has been visited (N_s). The exploration term guarantees that the algorithm targets less frequently visited nodes with a higher potential for discoveries. It allows UCT to search the state space effectively and converge on the optimal solution.

III. ReqsMiner: Design and Implementation

In this section, we primarily demonstrate the threat model under investigation, alongside the associated challenges faced. We then proceed to expound on the design of ReqsMiner, an innovative fuzzing framework, conceived to address the identified challenges and to uncover previously unknown inconsistencies in CDN forwarding requests. We conclude with an in-depth discussion of two unique techniques employed by each ReqsMiner module, namely the *Rule Generator* and *Grammar-based Fuzzing*.

A. Threat Model

In line with the end-to-end principle [44], the proxy should uphold the integrity of the request to the maximum extent during the forwarding process. Nonetheless, CDNs may alter the original messages when relaying requests, thereby creating a divergence between the initial and forwarded requests. CDNs might adopt this approach for commercial purposes, such as enhancing cache hit rates, or for safety measures leading to normalized requests. However, these disparities, known as *CDN forwarding request inconsistencies*, may also engender security vulnerabilities, including Denial of Service (*DoS*), Cache Poisoned Denial of Service (*CPDoS*), Forwarding Loop (*FL*), and Web Cache Poisoning (*WCP*) [11], [25], [32], [50].

As illustrated in Figure 2, this paper postulates that an attacker, posing as a legitimate client, has the ability to dispatch delicately crafted requests to the flawed CDNs, which then modify and forward the requests. We further assume that the target website as attacker's goal is hosted on the

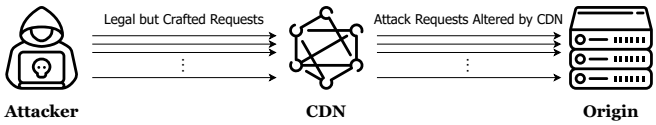


Fig. 2: Threat Model of CDN Forwarding Request Differences.

CDN, or unknowingly hosted by the attacker [23]. With these assumptions in place, the attacker can exploit specific CDN forwarding request discrepancies to manipulate requests within the *CDN-origin* connections. Using these malicious requests, an attacker may execute a variety of attacks on the victimized origin server, encompassing DoS, CPDoS, FL, and WCP.

Historically, the majority of proposed request discrepancies have been discovered manually in prior research [11], [25], [32]. This method may result in some variations in the forwarding request being overlooked. Therefore, our goal is to develop an automated fuzzing framework, termed REQSMINER, aimed at systematically and efficiently mining for forwarding request inconsistencies in CDNs. Furthermore, this framework should possess extensibility to facilitate its application to a wide range of specific attack threat models, and to enable the discovery of novel attack vectors.

B. Challenges in Fuzzing

Previous research typically employs two techniques to evaluate HTTP implementations and CDN behaviors. These include HTTP request test case generation using ABNF rules and automated testing directed towards CDNs [26], [45]. However, these methods inherently possess limitations and present challenges that impact both the efficacy and efficiency of testing. We detail these challenges below.

Initially, the ABNF rules pertaining to the HTTP protocol are unbounded, and test cases generated at random are often ineffective. As an illustration, ABNF rules state that a "request method" can be any "token" consisting of letters, numbers, and special characters (shown in Listing 1, Lines 9 to 12). Yet, in practical scenarios, some CDNs only support request methods that exclude numbers; a select few even only accept standard request methods alongside a handful of additional ones (e.g., `CHECKOUT` and `COPY`). This definition poses a considerable challenge for the fuzzer in terms of generating a valid value randomly. Besides, invalid test cases are generally rejected by the CDN. For instance, the `If-Range` header (`If-Range: Wed, 92 Oct 7180 11:45:14 GMT`) conforms to the ABNF rules; nonetheless, requests including this header will be denied by some CDNs due to its invalidity.

Secondly, the cost of testing CDNs is high. Since CDN is a commercial service, its source code is not publicly accessible. Conventional fuzzing techniques necessitate the dispatch of a large number of test cases to CDN services, which subsequently incurs substantial monetary costs.

Lastly, CDNs offer minimal feedback concerning test requests. Due to their black-box nature, the feedback provided by CDNs is limited. Consequently, it becomes arduous for a fuzzer to effectively detect all request differences with such scant feedback.

C. REQSMINER Overview

To discover more CDN forwarding threats and address the challenges mentioned in Section III-B, we designed REQSMINER to automatically and effectively fuzz the specific CDN forwarding request inconsistencies. Figure 3 illustrates the architecture, including *Rule Generator* and *Grammar-based Fuzzing* modules. Specially, we propose two novel techniques for improving fuzzing efficiency: *(i) Field Values-assisted Rule Fusioner* that merges *Field Values* into the ABNF rules to set the search bound and *(ii) UCT-based Request Generator* that uses the UCT-Rand algorithm to generate more valid test cases for black-box grammar-based fuzzing.

Rule Generator. REQSMINER utilizes *Field Values* to limit the search space of the ABNF rules by combining the ABNF rules and field values to generate an ABNF grammar tree (*Field Values-assisted Rule Fusioner*). Specifically, field values are RFC-compliant predefined data stored as key-value pairs extracted from the RFCs and actual web traffic. The key point is to incorporate expert domain knowledge into the generation rules and improve generation efficiency. In this module, *ABNF Parser* builds the ABNF grammar tree based on the ABNF rules extracted from RFCs to prepare for future generation, and *Rule Fusioner* integrates the field values into the ABNF grammar tree.

Grammar-based Fuzzing. REQSMINER leverages grammar-based fuzzing with the UCT-Rand algorithm to enhance the fuzzing efficiency by generating more valid test cases and detecting differences with less feedback (*UCT-based Request Generator*). UCT-Rand is capable of iterative optimization based on limited feedback and steadily improves the quality of test cases without sacrificing exploratory capability. For each module, the UCT-based *Request Generator* generates the HTTP test requests using the ABNF grammar tree and sends them to the CDN via the client. Both the client here and the server behind the CDN are under REQSMINER's control. After retrieving the CDN's forwarding status from the server, the parameters of the generation algorithm for each branch in the ABNF grammar tree are updated. Meanwhile, *Difference Analyzer* collects client-side and server-side logs, including the original requests sent by the client and the forwarded requests received by the server. It compares these logs to discover any changes to the forwarding requests.

Our framework is semi-automatic, as its initialization only requires the following data: *(i)* ABNF rules extracted from the RFCs; *(ii)* commonly predefined values in the RFCs; and *(iii)* HTTP field values extracted from the history logs of web servers. Besides, the user can optionally contribute extra domain-specific knowledge to the field values.

D. Rule Generator

In this module, *ABNF Parser* and *Rule Fusioner* are used to construct the ABNF grammar tree for effective test case generation from the *ABNF rules* and *Field Values*.

ABNF Parser. For the UCT-Rand algorithm to generate test cases, the ABNF parser must transform the ABNF rules extracted from RFCs into an ABNF grammar tree. As mentioned in Section II-B, ABNF syntax operators can be divided into three types: *concatenation*, *alternative*, and *repetition*. For each

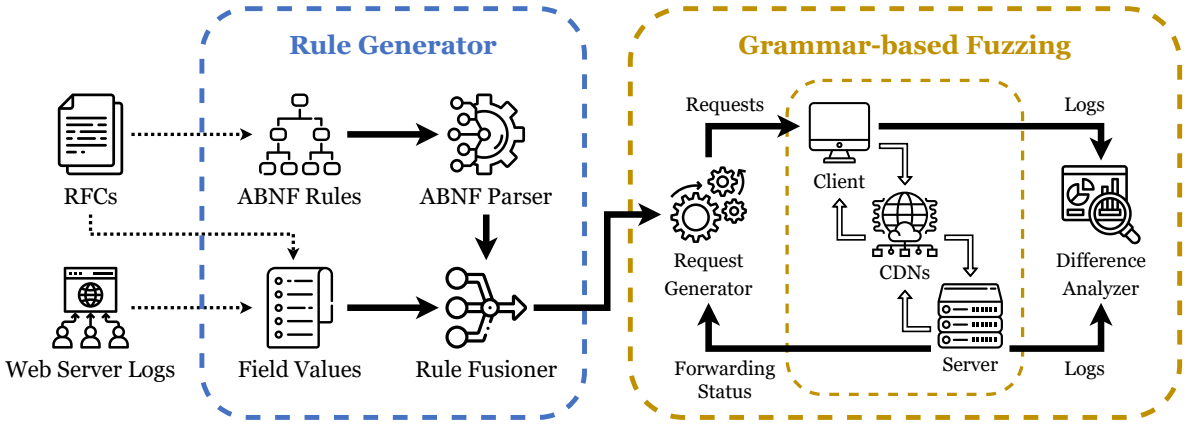


Fig. 3: The Architecture of REQSMINER.

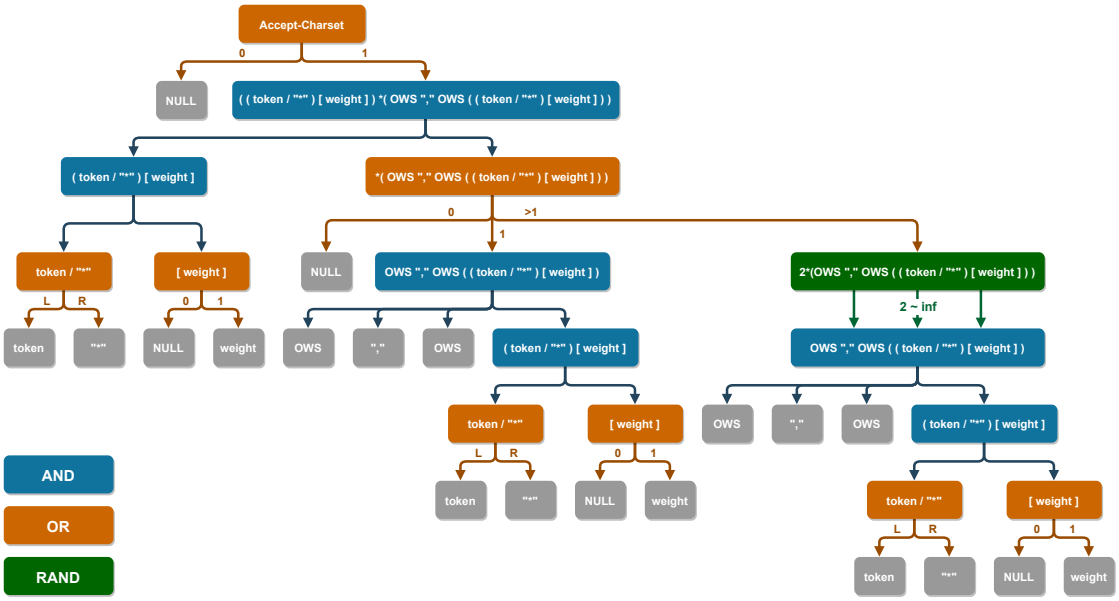


Fig. 4: The ABNF Grammar Tree of the `Accept-Charset` Header.

type of operator, the ABNF parser creates three types of ABNF grammar tree nodes, including `AND`, `OR`, and `RAND`.

As *concatenation* and *unique repetition* operators, `AND` nodes express concatenation in the ABNF grammar tree. All child nodes must be selected recursively to continue generating when visiting these nodes. `OR` nodes indicate selection and represent the *alternative* and *limited repetition* operators. Any child node must be visited recursively when visiting these nodes. Corresponding to the *infinite repetition* operators in the ABNF syntax, the `RAND` nodes mark random, which means that the number of times the child node is visited is random.

Figure 4 shows the ABNF grammar tree of the `Accept-Charset` header (the ABNF rule is shown in Listing 1, Line 5).

Rule Fusioner. To explore the knowledge extracted from *Field Values*, the rule fusioner parses the field values and inserts them into the ABNF grammar tree (*Field Values-assisted Rule Fusioner*). In detail, the fusioner first parses the field values following the ABNF rules to extract a set of non-terminal symbol values at the ABNF semantic level. For example, for the field value "`Accept-Language: en-US,en;q=0.9,en-GB;q=0.8,zh;q=0.7,ja`", this value contains a variety of `Language-Ranges`. After parsing it using the ABNF rule (shown in Listing 1, Line 6), the fusioner can extract multiple `Language-Ranges` values (e.g., `en-US`, `en`, `en-GB`, `zh`, and `ja`). This is to increase the likelihood that fuzzer will generate valid values, such as `en-US`, `en`, and `zh`.

After parsing, the fusioner expands the ABNF grammar tree with the set of non-terminal symbol values. The values of the non-terminal symbol are inserted as leaf nodes to the `OR` node and become the children of that node. For example, Figure 5 shows the subtree of `Language-Range` after inserting the values obtained from the previous step into the ABNF grammar tree. This is designed to increase the number of subtrees of the `OR` nodes in the ABNF grammar tree so that the UCT-Rand algorithm used by the request generator will have more options when traversing to that node.

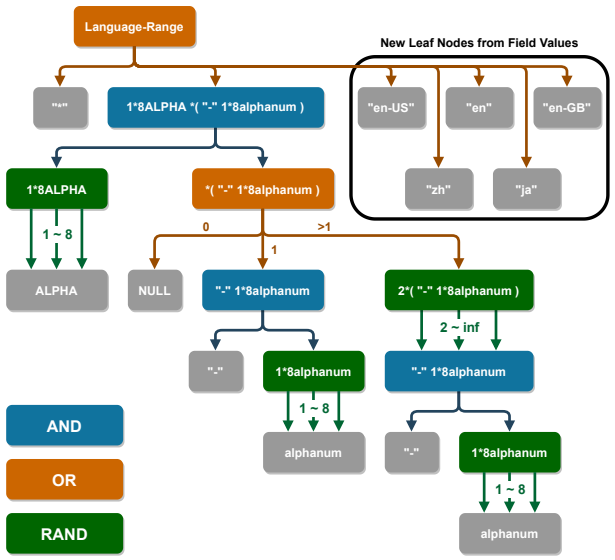


Fig. 5: The ABNF Grammar Tree of `Language-Range` with Field Values.

E. Grammar-based Fuzzing

REQSMINER uses *Request Generator* to generate more valid test cases and perform *Grammar-based Fuzzing* on CDNs to identify request differences and potential vulnerabilities effectively.

Request Generator. *Request Generator* is designed to generate numerous valid test cases from the ABNF grammar tree. The grammar tree includes three types of nodes, each of which represents an operation that can direct a depth-first traversal of the tree. REQSMINER will locate the target node (e.g., `HTTP-message`) as the root node and traverse the ABNF grammar tree recursively downwards. Among them, the leaf node (e.g., string literals, and num literals) is the termination node.

To make REQSMINER have superior effectiveness and exploration, inspired by the MCTS (UCT) algorithm, we propose a UCT-based weighted random generation algorithm (*UCT-Rand*) as illustrated in Algorithm 1. In contrast to UCT, UCT-Rand uses weighted random selection rather than the *argmax* function to choose the next child node during the selection phase. Specifically, the generation algorithm consists of 4 phases: *Expansion*, *Selection*, *Simulation*, and *Backpropagation*.

During the expansion phase, the request generator traverses the ABNF grammar tree starting from the root node. The traversed nodes produce a derivation tree that corresponds to an HTTP request. When traversing, the request generator employs specific strategies at different nodes. If the node is an `AND` node, it directly traverses all sub-nodes; otherwise, it enters the selection phase.

Throughout the selection phase, depending on the node type, the request generator uses several selection algorithms. If the node is a `RAND` node, the traversing times of the child nodes are decided at random. If the current node is an `OR` node with unvisited sub-nodes, then a random sub-node is

Algorithm 1: UCT-Rand Generation Algorithm.

Input: the ABNF grammar tree T
repeat
  $V \leftarrow \{T.root\}$ /* stack of nodes to visit */
  $S \leftarrow \{\}$ /* visited leaf nodes */
  $D \leftarrow \{\}$ /* decisions */
  /* Expansion */
  while V is not empty **do**
    $v \leftarrow V$.pop()
    if v is a leaf node **then**
      insert v into S, and goto Line 5
    switch the type of v **do**
      case `AND` node **do** V.push($v.children$)
      /* Selection */
      case `OR` node **do**
        if $v.unvisited\_children$ is not empty **then**
          $c \leftarrow$ choice($v.unvisited\_children$)
        else
          $c \leftarrow$ weighted rand$_{v' \in v.\text{children}} \left(Q(v,v') + \sqrt{\frac{2 \ln N(v)}{N(v,v')}} \right)$
        end
        insert (v, c) into D, and V.push(c)
      case `RAND` node **do**
        $t = v.lower\_bound$
        while $t < v.upper\_bound$ **do**
          t plus 1 or **break**, fifty-fifty
        end
        V.push($v.children$), repeat t times
    end
  end
  /* Simulation */
  $result \leftarrow$ RunTest(S) /* forwarding status */
  /* Backpropagation */
  for $(v, c) \in D$ **do**
    update $Q(v, c)$ and $N(v, c)$ based on $result$
  end
until EndConditions()

selected for expansion traversal. Alternatively, if all sub-nodes of the current node have been explored in earlier iterations, signaling that the node has completed its search, the UCB formula is used to determine the weights of all sub-nodes. For the subsequent traversal, a sub-node is chosen using a random selection with weights.

With UCB, the sub-node selected at node v is defined as follows:

$$\pi(v) := \text{weighted rand}_{v' \in v.\text{children}} \left(Q(v,v') + \sqrt{\frac{2 \ln N(v)}{N(v,v')}} \right) \quad (2)$$

Where:

- $Q(v, v')$ is the probability of generating successfully forwarded requests by the CDN after selecting sub-node v' under node v.
- $N(v)$ is the number of times that node v has been visited.
- $N(v, v')$ is the number of times that sub-node v' has been selected under node v.

During the simulation phase, the request generator transforms visited leaf nodes into HTTP requests and sends them to the CDN via the client. The generator then retrieves the forwarding status of the CDN from the server and logs if the request was successfully forwarded by the CDN and forwarded to the server.

In the backpropagation phase, the request generator updates the parameters of the generation algorithm (including $Q(v, v')$ and $N(v, v')$) of each node in the ABNF grammar tree based on the success of CDN forwarding.

Courier Platform. The courier platform comprises the client, the server, and the targeted CDNs that undergo testing. The client sends test requests to the targeted CDNs, which are subsequently forwarded to the server by the CDNs. These requests are recorded and sent to the difference analyzer. Additionally, the server receives the request from the CDN and responds with an HTTP response that is unrelated to the request to the CDN. The server logs the forwarded request from the CDN and channels it to the difference analyzer. The server also provides a log to the request generator indicating that the CDN has successfully forwarded the request.

Difference Analyzer. The difference analyzer is responsible for detecting alterations made by the CDN when forwarding the request by performing a comparison analysis. Specifically, it extracts the request structures from both the original and forwarded requests using simplified ABNF rules. Through this comparison of structures, the difference analyzer reveals the various kinds of modifications implemented on the request by the CDN. These include:

- *Alteration*: The CDN modifies the original request's header value.
- *Insertion*: The CDN introduces a header that is absent in the original request.
- *Deletion*: The CDN removes a header from the original request.
- *Duplicate Header Insertion*: The CDN introduces a header already present in the original request.
- *Duplicate Header Deletion*: The CDN eliminates the duplicate header from the original request.

Ultimately, the various types of modifications are archived in a database, facilitating their use in subsequent research.

IV. Experiments and Findings

In this section, we deploy ReqsMiner on the server and execute it to evaluate the CDNs in the controlled experiment to reveal any discrepancies in request forwarding behavior. Subsequently, we classify these multiple discrepancies observed in the request forwarding of CDNs and investigate whether these identified inconsistencies potentially contribute to security risks.

A. Experiment Setup

To conduct an evaluation of the ReqsMiner, we systematically analyzed 22 widely recognized CDN services known for their substantial market shares and deployment rates. The list includes Akamai, Aliyun, Azure, Baidu Cloud, BunnyCDN, CDN77, CDNetworks, CDNSun, ChinaCache, ChinaNetCenter, Cloudflare, CloudFront, Fastly, Gcore, Google Cloud, Huawei Cloud, KeyCDN, Qiniu Cloud, StackPath, Tencent Cloud, Udomain, and Verizon [9]. Their security issues could potentially pose a threat to a broad spectrum of users.

Experiment Platform Setup. We conducted our experiments on two Linux servers with Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-125-generic x86_64) as client and server. Both were equipped with a 2.40GHz 32-core CPU (Intel Xeon E5-2640 v4), 16GB RAM, and 1000Mbps bandwidth capacity. As for the CDNs, we opted for the default configurations and set one of the experimental servers as the origin server by CNAME or domain hosting [33]. The second experimental server was designated as the client.

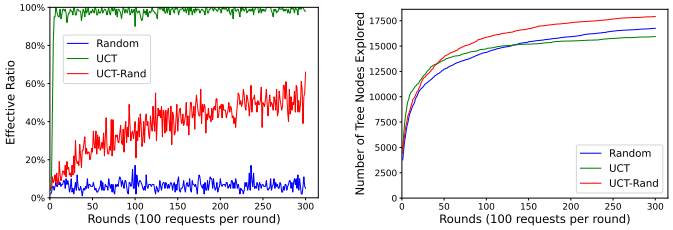
Test WorkFlow. During our experiment, we analyzed the core specification of HTTP/1.1 protocols (RFC 9110-9112), and other supplementary specifications (RFC 3986, 4647, 5234, 5646) [6], [15], [18]–[20], [42], [43]. In addition, we collected web access logs for the lab homepage that was deployed on our server. We also collected the HTTP logs of the network security scanner Xray during vulnerability scans [10]. Targeted adversarial traffic can improve the complexity of the generated requests. Based on these protocols and the HTTP request logs we collected, we extracted 442 ABNF rules and 63 sets of field values that were used as input for ReqsMiner. Then, using the request generator, ReqsMiner automatically generated numerous HTTP test requests, each associated with a unique Universal Unique Identifier (UUID). To prevent cache hits on the CDN and to facilitate the server's acquisition of the forwarded request's UUID, ReqsMiner inserted the UUID into the request-target URL. After storing these test cases in the database, the client sent them to the target CDN using multithreading. To minimize confusion during packet parsing, we utilized low-level network programming (e.g., raw socket) to send and receive packets. Then, normally, the CDN received the request and forwarded it to the origin server under our control. Requests with high distortion were likely to be rejected by the CDN.

On the origin server, ReqsMiner listened on port 80 directly and received requests forwarded by the CDN. After parsing, ReqsMiner extracted the UUID associated with the request and stored this request in the database, using the UUID as the index.

For each tested CDN, we ran 100 rounds of testing with 1,000 test cases per round, for a total of 100,000. With normal network quality, a CDN can complete a full test within 300 minutes at a low rate of 6 requests per second. We intentionally kept this rate low to minimize any potential adverse impact on the tested CDN. During the testing, we mainly collected two types of logs for difference analysis: *(i)* client logs including the original requests; *(ii)* server logs including the requests forwarded by the CDN.

Experiment Cost. For most CDN vendors, we subscribe to their free service or trial service. Akamai and Verizon only provide commercial CDN services for enterprise customers. However, we can configure Akamai and Verizon CDN services on the Microsoft Azure platform, and they can work normally. The same goes for ChinaCache and ChinaNetCenter. We obtained test accounts with a trial service by contacting customer service. Aliyun, Baidu Cloud, Huawei Cloud, Qiniu Cloud, and Tencent Cloud do not offer free or trial services, so we subscribed to their paid service. In the end, the whole experiment cost less than $10.

Ethical Considerations. We take utmost care to prevent ethical problems in our experiments. *(i)* All the CDN services



(a) Effectiveness (b) Exploration

Fig. 6: Evaluation of Different Generation Algorithms.

involved in this experiment were purchased at our own expense for the sole purpose of this experiment. Besides, we sent requests to the CDN at a relatively low frequency, which will not affect normal CDN services. *(ii)* The origin servers involved in this experiment were under our control, and no attack was launched against any real users or external servers. *(iii)* We actively informed all CDN vendors being tested and put contact information up on the self-built testing website. *(iv)* We implemented measures to monitor CDN status in real-time to mitigate potential impacts on CDN deployment. If a crash is detected, we will stop testing immediately and report any payloads that may have caused the crash to expedite resolution and minimize disruption. In our experiments, we did not actually detect any crashes. *(v)* We followed the established coordinated disclosure best practices. Vulnerabilities identified in this work have been reported to all relevant CDN providers.

B. *Evaluation of* REQSMINER

Unlike conventional methodologies for assessing fuzzer performance, the evaluation of REQSMINER's efficiency using metrics such as false positive and true negative rates presents unique challenges due to the intricacy involved in vulnerability determination. The distinctive innovation of REQSMINER resides in its capacity to identify a broad spectrum of unique inconsistencies, outperforming random generation approaches. To accomplish this, the fuzzer must adequately explore the input space and maintain a high success rate in generating requests. Consequently, our evaluation concentrates on gauging the efficiency of various algorithms in terms of both effectiveness and exploration.

To evaluate the effectiveness of REQSMINER, we conducted additional experiments. Specifically, we employed a local Nginx instance as the CDN for testing and configured Nginx to operate in reverse proxy mode, thereby enabling the detection of differing requests during forwarding. In order to generate test cases, we employed three distinct algorithms: *(i)* Random: In this algorithm, child nodes are randomly selected; *(ii)* UCT: This algorithm uses the *argmax* function to determine child nodes; *(iii)* UCT-Rand: Similar to UCT, but utilizes weighted random selection of child nodes. For each algorithm, we executed 300 testing rounds, generating 100 test cases per round. Throughout this evaluation, we noted the number of traversed ABNF grammar tree nodes by the generation algorithm and the proportion of test cases successfully forwarded by the CDN.

Figure 6 displays the evaluation results, where the effective ratio indicates how many requests that were forwarded normally by Nginx rather than refused. Concerning effectiveness, the Random consistently falls below 20% in each round; the UCT is at least 90% effective starting from the second round, and UCT-Rand commences under 10% effective, gradually rising to over 60% effective by the final round. In terms of exploration, Random and UCT-Rand share similar growth curves, yet UCT-Rand always boasts more explored nodes than Random. After 50 rounds, the count of explored nodes with UCT grows markedly slower than those in Random and UCT-Rand.

In summary, while requests generated by UCT exhibit effectiveness in terms of forwarding, they fall short in exploring the grammar tree. This is attributed to UCT prioritizing the most promising nodes in the ABNF grammar tree, which are the most probable nodes to facilitate CDN request forwarding, thus becoming ensnared in the local optimization of CDN forwarded requests. Since UCT presumes these similar requests will be forwarded, it persistently generates them with high probability, thereby manifesting an overfitting issue.

Contrastingly, UCT-Rand, through weighted randomness, maintains a balance between successful forwarding and ABNF exploration, effectively breaking free from local optimization. This algorithm refines the quality of generated test cases based on feedback while exploring more grammar tree nodes. Consequently, UCT-Rand has demonstrated superior effectiveness and exploration, both of which are crucial requirements for our fuzzing test. Based on this, we have selected UCT-Rand as the generation algorithm for our experiment.

C. *Difference Findings*

In our experiment, REQSMINER spotted numerous CDN forwarding request inconsistencies. We categorized them into three major groups based on where they occur in the HTTP message: *(i)* Differences in Request Line, *(ii)* Differences in Header Fields, and *(iii)* Differences in Message Body. As we elaborated on these findings, we concurrently conducted a brief analysis to discern whether these inconsistencies might be a product of the CDN's security mechanisms, as described in the CDN provider's documentation [14], [17]. Additionally, through some simple manual verification processes, we provided a preliminary exploration of the potential for these inconsistencies to be exploited by adversaries in the initiation of attacks.

1) Differences in Request Line: The format of the request line for the HTTP/1.1 protocol is specified by RFC 9112 [20, §3]. There are three parts in the request line: request method, request URL target, and HTTP version.

Differences in Request Method. In HTTP protocols, a request method is the method used to request data from the server. The most common request methods are `GET`, which requests a resource from the server, and `POST`, which submits data to be processed by the server. RFC 9110 defines 8 common standardization methods used in HTTP, and specifies that additional methods ought to be registered within the "Hypertext Transfer Protocol (HTTP) Method Registry" [19].

In our experiments, differences in request method are observed in nearly half of the CDNs. 11 CDN services change the

HEAD request to a GET request. After receiving the complete request resource from the origin server, the CDN removes the message body of the response and forwards it to the requesting client, in order to comply with the RFC for HEAD requests. However, this can lead to unequal response traffic sizes. An attacker can exploit this inequality to launch a DoS attack against the origin server by consuming the server's outbound bandwidth with little bandwidth of request traffic through the CDN. We will further discuss how to exploit it and propose a novel class of DoS attack in Section V-B.

In addition, Aliyun and Qiniu Cloud change some special request methods (e.g., LOCK, MERGE, and MKACTIVITY) to the GET method. Unlike the case above, the CDN does not alter the content when forwarding the response. However, this modification causes the origin server to receive an incorrect request, even if the server supports the particular request method listed above.

Differences in Request URL Target. The request URL target is part of an HTTP request that specifies the location of the requested resource on the server. It is typically part of the URL following the domain name and can include parameters and query strings.

In our experiments, we noticed that most CDNs forward the URLs to the origin server unchanged, but there are some exceptions. 6 CDNs discard the part of the URL after the hash ("#") when forwarding requests: Akamai, Aliyun, BunnyCDN, CloudFront, Qiniu Cloud, and Verizon. This is correct because the hash is used to identify a specific location in the document that the CDNs do not think needs to be forwarded to the origin server. Akamai will combine two adjacent slashes ("/") in the URL into one. Azure and BunnyCDN will parse the URL and removes things like "/./" and "/<dir>/../" before forwarding.

Differences in HTTP Version. The HTTP version in the request line indicates the version of the HTTP protocol that clients are using. Web servers need to identify the HTTP version and respond to HTTP requests accordingly. The format of the HTTP version is HTTP/X.X where X.X is the version number.

We observe that different CDNs allow different HTTP versions. Aliyun, Qiniu Cloud, and Tencent Cloud permit requests with HTTP versions 1.0 and 1.1. All other CDNs allow versions 1.0–1.9, with Verizon also allowing version 0.9, and Akamai and StackPath permitting versions 0.0–9.9. CDNs also modify the HTTP version when forwarding requests, with most changing it to 1.1. However, Verizon modifies the version to 1.0 when forwarding requests with versions 0.9/1.0, and StackPath alters it from 0.0–0.9 to 0.9.

We think that it would be better for CDNs to force the HTTP version to be 1.1 when forwarding requests, as HTTP/0.9 and HTTP/1.0 have both been deprecated [7]. However, it is still possible for Verizon and StackPath to modify the version to 0.9 or 1.0 when forwarding. In cases where the origin server does not support the HTTP version modified by the CDN, a response with status code 400/505 will be received by the CDN If the CDN classifies these status codes as cacheable, it could result in a CPDoS attack.

2) Differences in Header Fields: HTTP headers allow clients and servers to pass additional information via HTTP requests or responses. HTTP headers are included in the message header, after the request line (or status line in the case of a response) and separated from the message body by a blank line.

Differences due to Duplicate Headers. RFC 9110 specifies how to handle duplicate headers. When a header is duplicated, its combined field values consist of a list of the corresponding field row values within that section, joined in order, with each field row value separated by a comma. And field names are case-insensitive [19], [46].

The experiments found that although all CDNs ignored the case of field names by default, they handled duplicate headers differently. Most CDNs forwarded all duplicate headers, while some modified the case of header names or merged the values in specific ways. Notably, BunnyCDN merges duplicate header values with semicolons instead of commas, which violates the RFC. Suppose the origin server cannot resolve header values separated by semicolons. In that case, the CDN may receive a response with a status code of 4xx/5xx, putting it at risk of CPDoS attacks if that code is tagged as cacheable by the CDN.

Because the ABNF rules cannot restrict headers uniqueness, the requests generated by REQSMINER might contain duplicate headers. The experiments have further revealed that CDNs use distinct strategies when dealing with specific duplicate headers, such as Host and Content-Length.

In cases where a request is received with multiple Host headers, most CDNs discard all but the first one, except 6 CDNs that reject it with status code 400/500. In particular, Google Cloud forwards the request by combining the values of multiple Host headers, separated by commas, violating RFC rules.

In cases where a request with multiple Content-Length (CL) headers is received, most CDNs will reject it with "400 Bad Request". However, Aliyun and Fastly accept the request and forward it without modification, keeping only the first CL. Tencent Cloud does the same, but only keeps the last CL.

Differences caused by Adding Headers. When forwarding requests, CDNs append certain headers containing information related to clients. For instance, Cloudflare adds four headers: CF-Connecting-IP, CF-IPCountry, CF-RAY, and CF-Visitor.

To protect against forwarding loop attacks, CDNs add CDN-Loop and Via headers to indicate that the CDN has forwarded the request [11]. However, we observed that only five CDNs (CDNetworks, ChinaNetCenter, Cloudflare, Fastly, and Gcore) added CDN-Loop headers. Conversely, among the remaining CDNs, five CDNs do not add CDN-Loop or Via. This means that most CDNs are still vulnerable to forwarding loop attacks. Attackers can cascade multiple CDNs through malicious configurations and perform attacks.

Differences caused by Removing Headers. CDNs also remove specific headers when forwarding requests. Some headers, such as TE and Upgrade, are removed as the RFC requires. However, removing some headers, such as Range

and conditional headers, can cause changes to the expected response and lead to DoS attacks. In our experimental results, 12 CDNs removed the `Range` headers, while 17 CDNs removed some or all conditional headers.

A previous study has revealed the RangeAmp attack that results from removing `Range` headers [32]. Moreover, in Section V-C, we will further discuss the DoS attacks that can occur due to removing conditional headers.

Differences caused by Altering Headers. In addition to adding and removing headers, CDNs alter certain headers, such as `Accept-Language` and `Accept-Encoding`. In HTTP, `Accept-Language` specifies the preferred languages for the response from the server. When CDNs modify `Accept-Language` in forwarded requests, it may result in response content that does not match the user's actual preferred language, thus affecting user experience. We found that the `Accept-Language` header is modified by three CDNs (Azure, BunnyCDN, and Verizon). Meanwhile, 15 CDNs modify the `Accept-Encoding` header, which may result in DoS attacks. Section V-D will further discuss the DoS attacks that can result from changes to `Accept-Encoding`.

3) Differences in Message Body: The message body in an HTTP request is the data contained in the request after the headers. Before sending a request to the server, clients can either compress or chunk the message body.

Differences by Removing Message Body. According to RFC 7231, a payload within a GET/HEAD request message has no defined semantics, leading some implementations may reject such requests [21]. These are also referred to as "fat requests". Different CDN services handle such requests inconsistently, with only Akamai and Azure removing the message body from GET/HEAD requests. Other CDNs do not show this behavior, which can lead to a response with status code 4xx/5xx from the origin server that does not support the fat requests. This risks CPDoS attacks if the status code can be cached by the CDNs.

Differences in Transfer Encoding. Transfer encoding is the process of chunking the message body before sending it to web servers, which is useful for sending large files or streaming data. The most common `Transfer-Encoding` value is "chunked". However, some CDNs like Aliyun, CDN77, Cloudflare, and Gcore send all the chunked transfer data to the origin server in one piece, altering the `Transfer-Encoding` of original requests in the process. This means that the CDN needs to receive all the data before forwarding the request to the origin server, which can defend against Slowloris-like attacks but also potentially cause pulse-based DoS attacks. An attacker can cause pulse-based DoS attacks by exploiting CDNs. This can be done by opening multiple connections to the CDN, sending a large amount of data using chunked encoding during the connection lifetime (in low bandwidth), and then ending all the request-sending processes at the same time, causing the CDN to send a large amount of data (in high bandwidth) to the origin server instantaneously.

V. HTTP Amplification Attacks

Based on the exposition provided in Section IV, it is understood that discrepancies do not directly signal the existence of potential security implications. Nevertheless, it is possible to uncover new attack vectors by extending ReqsMiner and integrating it into the threat model of a specified attack. For instance, we chose the amplification attack as a threat model for an in-depth study. We augmented the analysis module of ReqsMiner, enabling it to detect discrepancies in traffic size due to inconsistencies in the requests. As a result of this extension, we identified three novel HTTP amplification attacks, yielding 74 vulnerabilities across 19 CDN providers. These vulnerabilities allow powerful DoS attacks against origin servers, utilizing minimal bandwidth, yet yielding substantial effects. The amplification factor reach a *2,000* times in general, and under certain conditions, can even surpass *1,920,000* times, which is significantly higher than previous attacks [25], [32], [40].

A. Overview of HTTP Amplification Attacks

HTTP amplification attack is a type of DoS attack where attackers exploit vulnerabilities in HTTP servers to amplify their network traffic, causing the victim's network to become overloaded and unable to provide normal services.

Threat Model. As shown in Figure 7, an attacker can impersonate a normal client and exploit vulnerabilities by sending a legal but delicately crafted request to the CDN towards a website hosted on it (step 1). This crafted request would "manipulate" the CDN to fetch large content from the website's origin server, which will become the victim (step 2). Then the origin server returns the requested large content to the CDN (step 3). Once numerous large responses consume the outbound bandwidth, this can cause a DoS attack. The basic aim of this threat model is to leverage the CDN's inconsistent request-handling behaviors to increase the traffic scale between CDN servers and origin servers (step 3) while reducing the traffic expense between attackers and CDN servers (step 4).

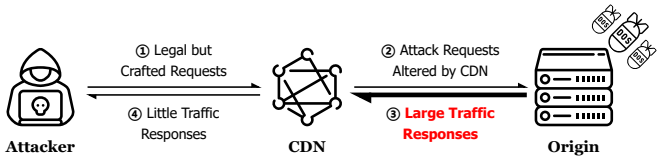


Fig. 7: Threat Model of CDN-based Amplification Attacks.

In order to effectively leverage this threat model, the attacker must successfully avoid the CDN's cache. If they fail to do so, the CDN will inhibit them from requesting content directly from the origin server. CDNs typically include a response header signifying the cache lookup result, which allows attackers to determine if the resource is cached by issuing requests. If it is cached, attackers can employ cache-busting strategies to compel the CDN to re-cache it.

It is important to note that CDNs have implemented defenses against such attacks. In response to researchers proposing a strategy to amplify DoS attacks via connection termination [49], nearly all CDNs have implemented mitigation measures. If the attacker disconnects from the CDN before receiving all the data, the CDN will sever the connection to the origin server immediately, effectively thwarting the attack [23]. However, as our research progressed, we discovered three new

attacks based on this threat model that bypass the aforementioned mitigations. These attacks will be described in detail in the following sections.

Experiments Setup. Firstly, we enhanced REQSMINER with the capability to unearth potential vectors for amplification attacks. In our improved version, REQSMINER's analysis module also records the traffic size of both the client and the server. If it is found that the server's traffic size is more than ten times that of the client, we will identify the originating request that led to this discrepancy as a potential vector for an amplification attack.

Subsequently, similar to previous experiments (Section IV), we configured the CDN services and added a controlled origin server. We deployed Nginx as the web application and sent requests with different payloads to affected CDNs according to different types of amplification attacks. Then, we captured all response traffic in the *client-CDN* connection and *CDN-origin* connection and calculated the ratio of *CDN-origin* traffic and *client-CDN* traffic as the amplification factor. This metric effectively illustrates the discrepancy in bandwidth resource consumption between the attacker and the victim [32]. Besides, we evaluated our attacks with vendors' consent and limited impact.

B. HEAD Request-based HTTP Amplification Attack

We explore the significant traffic size differences between `GET` requests and `HEAD` requests and introduce a novel class of traffic amplification attacks, dubbed *HEAD Request-based HTTP Amplification (HeadAmp) Attack*. If a CDN converts the request into a GET request when it forwards a HEAD request, an attacker can craft a HEAD request to launch a HeadAmp attack.

Attack Details. As shown in Figure 8, the attacker crafts a HEAD request and sends it to a vulnerable CDN (step 1). To cache the resource, the CDN transforms the request into a GET request and forwards it to the origin server (step 2). This causes the origin server to respond with the entire target resource (step 3), while the CDN returns only the response line and headers to the attacker (step 4). The response traffic in the *client-CDN* connection is only a few hundred bytes, while the response traffic in the *CDN-origin* connection is equal to the entire target resource, resulting in an amplification factor that increases with the size of the target resource. However, the amplification ratio will be limited if the CDN disconnects from the origin server after receiving responses.

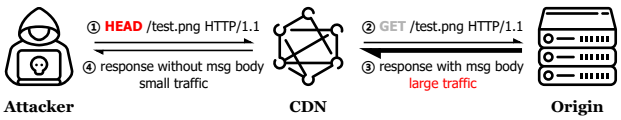


Fig. 8: Example of HeadAmp Attacks.

Additionally, besides meeting the prerequisites mentioned in Section V-A, the attacker needs to ensure that the target resource used by the attack is cacheable by the CDN, which can also be determined by making normal requests and checking the cache lookup status header in the response. If it is not cacheable, the CDN will only make a HEAD request to the origin server instead of a GET request.

TABLE I: Amplification Factors with Different Target Resource Size of HeadAmp Attacks.

CDN	Amplification Factor			
	1MB	10MB	25MB	Max (≤1GB)
Aliyun[1]	137.52	144.05	140.49	154.20
Azure[1]	56.70	56.56	56.48	56.70
BunnyCDN	1119.00	11198.95	27575.01	1095296.82*
CDN77[1]	23.79	35.54	59.12	59.28
CDNetworks	1595.73	15599.15	39056.21	1330849.57*
ChinaNetCenter	1566.94	15667.43	38567.16	1315155.58*
Cloudflare[2]	967.15	9717.67	23827.26	483332.05
Fastly[3]	1465.48	14540.97	30.79	29243.69
Gcore	1725.39	16963.68	43094.88	1680775.18*
KeyCDN[1]	27.20	27.13	57.94	58.25
StackPath	1607.70	15853.18	40150.99	1573951.48*
Udomain[4]	1489.30	1488.06	1485.17	1491.31

* Amplification factor can be greater if the file size is larger than 1GB.
[1] Terminate the request as soon as all the headers received.
[2] Terminate the request if the file size is larger than 512MB.
[3] Refuse with "503 Service Unavailable" if the file size is larger than 20MB.
[4] First request for the first 1MB of file, then response to the client with headers.

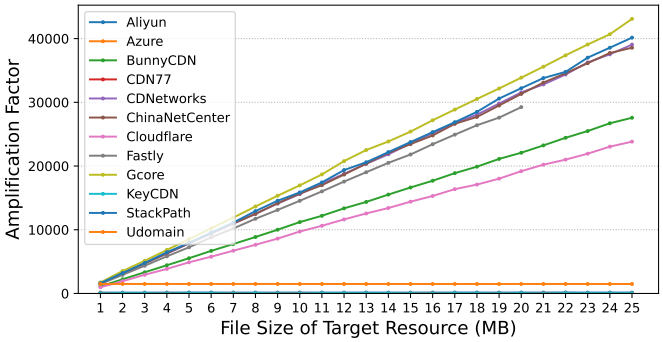
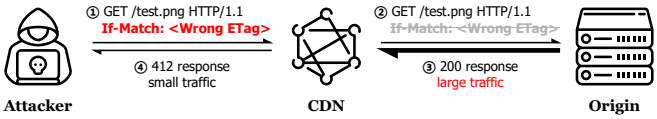


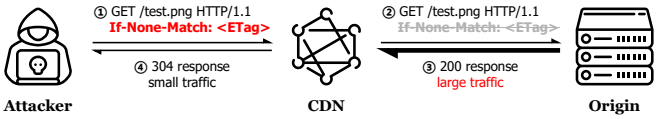
Fig. 9: Distribution of Amplification Factors for HeadAmp Attacks with Different Target Resource Size and CDNs.

Real-World Attack Analysis. As shown in Section IV-C1, 12 CDNs transform HEAD requests to GET requests while forwarding, making them vulnerable to HeadAmp attacks. We conducted a controlled experiment in the wild to determine their amplification factors by requesting target resource sizes ranging from 1MB to 25MB in 1MB increments. The specific amplification factors for the target resource sizes of 1MB, 10MB, 25MB, and "Max" are listed in Table I, with the maximum possible amplification factor limited to a requested resource size of no more than 1GB. As shown in Figure 9, when the target resource size is fixed, the amplification factor is nearly identical across all CDNs except for Cloudflare and BunnyCDN, which insert more headers in the response, resulting in slightly lower amplification factors. Certain CDNs also do not limit the size of the requested resources, resulting in amplification factors up to 1,680,775 when limiting the requested resource size to a maximum of 1GB.

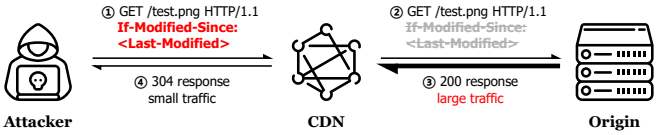
Several CDNs take various measures to limit the amplification effect: *(i)* Aliyun, Azure, CDN77, and KeyCDN stop requests immediately after receiving all headers, but there is a possibility that some data may have already been sent from



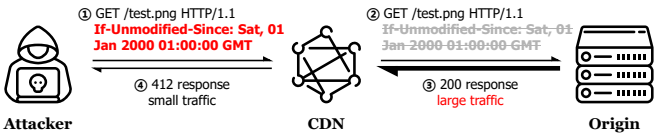
(a) Attack with `If-Match` Header.



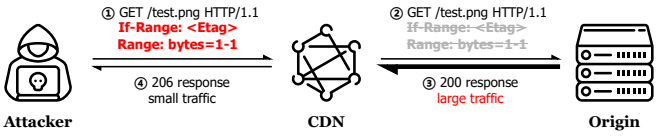
(b) Attack with `If-None-Match` Header.



(c) Attack with `If-Modified-Since` Header.



(d) Attack with `If-Unmodified-Since` Header.



(e) Attack with `If-Range` Header.

Fig. 10: Examples of CondAmp Attacks.

the origin server before the connection was terminated; *(ii)* Cloudflare terminates a request upon receiving all headers if the requested resource is larger than 512MB; *(iii)* By default, Fastly responds with a "503 Service Unavailable" error if the requested resource is greater than 20MB; *(iv)* Udomain adds header "`Range:1-1048576`" when requesting and does not request again after getting all headers, hence its amplification factor has an upper bound.

C. Conditional Request-based HTTP Amplification Attack

If a CDN removes the conditional headers when forwarding conditional requests, it can make it possible for an attacker to craft a "malicious" but legal conditional request and launch a *Conditional Request-based HTTP Amplification (CondAmp) Attack*.

Attack Details. There are five conditional headers that can be exploited in CondAmp attacks. As shown in Figure 10, the attacker constructs a conditional request and sends it to a vulnerable CDN (step 1). The CDN forwards the request to the origin server while dropping the conditional headers (step 2), which causes the origin server to return a complete copy of the requested resource (step 3). The attacker then manipulates the conditional header to prevent the CDN from returning the resource (step 4), resulting in the CDN responding with just the response line and headers. Similar to HeadAmp, the amplification factor in a CondAmp attack increases with the size of the target resource. Additionally, for reasons akin to HeadAmp, the targeted resource, for a CondAmp attack, must be uncached but cacheable.

Real-World Attack Analysis. As shown in Section IV-C2, we found that 16 CDNs discard conditional headers when forwarding conditional requests, making them vulnerable to CondAmp attacks. Similar to the prior experiment, we set different target resource sizes, including 1MB, 10MB, and 20MB, resulting in amplification factors listed in Columns 2-4 of Table II. The maximum achievable amplification with a target resource size of no more than 1GB is listed in Column 5 of Table II.

D. Accept-Encoding-based HTTP Amplification Attack

The variances in traffic caused by Accept-Encoding-specific policies result in a new form of traffic amplification attack, labeled as *Accept-Encoding-based HTTP Amplification (AEAmp) Attack*. Specifically, if a CDN adopts the deletion policy for handling the `Accept-Encoding` header, an attacker can devise an `Accept-Encoding` header that comprises a compression algorithm to execute an AEAmp attack.

Attack Details. Figure 11 illustrates the attacker's modus operandi for executing an AEAmp attack. Firstly, the attacker generates a request with the header "`Accept-Encoding: gzip`" and sends it to a vulnerable CDN (step 1). Upon receipt, the CDN removes the `Accept-Encoding` headers and forwards the request to the origin server (step 2), resulting in the origin server returning an uncompressed target resource (step 3) while the CDN furnishes a compressed target resource (step 4).

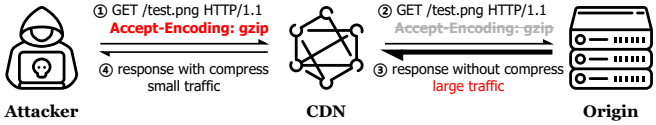


Fig. 11: Example of AEAmp Attacks.

In an AEAmp attack, the response traffic in the *client-CDN* connection pertains to the compressed target resource (small), while the response traffic in the *CDN-origin* connection corresponds to the complete uncompressed target resource (large). As a result, the amplification factor is higher for target resources with greater compression rates. To boost the compression rate, an attacker can utilize the upload feature of a website to upload a file that consists of identical content (e.g., all content is *\00*) and exploit this file as the target resource to launch a potent AEAmp attack.

Real-World Attack Analysis. Section IV-C3 of our study demonstrates that four CDNs are vulnerable to AEAmp attacks as they discard `Accept-Encoding` headers upon forwarding requests. In our experiments, we set the target resources to solely comprise zero-byte files to obtain the maximum amplification factor. We utilized distinct attack loads for different CDNs (listed in Column 2 of Table III) and tested various target resource sizes, ranging from 1MB to 25MB. The amplification factors for the respective target resource sizes are presented in Columns 3-5 of Table III.

TABLE II: Amplification Factors with Different Target Resource Size of CondAmp Attacks.

(a) Attack with `If-Match`.

CDN	Amplification Factor			
	1MB	10MB	20MB	Max (≤1GB)
Azure[1]	1279.62	13640.29	23688.26	23688.26
Baidu Cloud[1]	1380.63	4765.35	4756.42	6826.85
BunnyCDN[1]	46.79	46.72	40.51	46.79
CDNetworks[1]	24.59	28.08	31.56	31.56
CDNSun[1]	63.95	63.95	57.51	63.95
ChinaNetCenter	1231.49	12731.49	24831.49	1178739.20*
Cloudflare[1]	34.58	34.52	32.28	34.58
Gcore[1]	48.33	33.67	63.14	63.14
Huawei Cloud	1302.69	12965.64	25804.73	1286965.74*
Udomain[1]	63.67	63.58	63.74	63.74

(b) Attack with `If-None-Match`.

CDN	Amplification Factor			
	1MB	10MB	20MB	Max (≤1GB)
Aliyun	1376.95	13746.82	29480.45	1143179.80*
Azure[1]	1494.58	14582.42	27178.19	27178.19
Baidu Cloud[1]	1493.35	5132.33	5147.56	7395.95
BunnyCDN	1197.92	11764.44	23553.99	1172958.12*
CDNetworks	1555.06	17321.00	30671.49	1721487.32*
CDNSun	1955.17	19475.55	39074.55	1927288.09*
ChinaNetCenter	1526.73	16045.67	30289.85	1511756.22*
Cloudflare[2]	1015.18	10154.17	20302.83	575322.41
Fastly[3]	1831.95	18274.44	32919.45	32919.45
Gcore	1917.59	18870.12	37761.45	1884424.91*
Huawei Cloud	1255.05	12579.15	24936.88	1235931.80*
Qiniu Cloud	1503.22	14855.64	29300.20	1355751.89*
Udomain[4]	1631.73	1631.83	1810.82	1810.82

(c) Attack with `If-Modified-Since`.

CDN	Amplification Factor			
	1MB	10MB	20MB	Max (≤1GB)
Aliyun	1473.16	14946.99	29574.55	1354919.63*
Azure[1]	1376.15	14298.48	30676.27	30676.27
Baidu Cloud[1]	1492.15	5139.53	5140.73	5193.15
CDNetworks	1555.73	15505.32	31018.09	1442440.80*
CDNSun	1955.65	19182.45	38724.78	1925005.47*
ChinaNetCenter	1527.57	15182.11	30269.21	1429888.15*
Cloudflare[2]	1021.95	10045.26	20106.55	519026.19
CloudFront[1]	140.36	140.36	140.26	140.36
Fastly[3]	1814.89	16190.66	36467.10	36467.10
Gcore	1917.71	19163.23	38311.90	1877317.00*
Huawei Cloud	1252.63	12370.51	24861.33	1242765.42*
Qiniu Cloud	1505.18	13189.91	26181.69	1337846.65*
Udomain[4]	1632.43	1631.73	1648.74	1648.74

(d) Attack with `If-Unmodified-Since`.

CDN	Amplification Factor			
	1MB	10MB	20MB	Max (≤1GB)
Akamai[1]	48.99	41.28	56.91	56.91
Azure[1]	1280.01	13613.49	22754.04	22754.04
Baidu Cloud[1]	1379.85	4758.04	4752.64	4796.06
BunnyCDN[1]	1121.41	11086.40	18373.83	18373.83
CDNetworks[1]	1326.17	13018.93	26066.93	26066.93
ChinaNetCenter[1]	1183.76	12927.97	25911.45	25911.45
Gcore[1]	48.36	37.85	42.03	48.36
Huawei Cloud	1301.47	12914.44	25778.09	1288875.96*
Udomain[1]	63.58	63.58	50.90	63.58

(e) Attack with `If-Range`.

CDN	Amplification Factor			
	1MB	10MB	20MB	Max (≤1GB)
Aliyun[1]	177.59	181.68	145.18	181.68
Azure[1]	1247.07	15821.07	25013.66	25013.66
Baidu Cloud[1]	1377.37	4772.99	4754.65	4772.99
BunnyCDN	1008.24	10090.34	20136.81	1070350.65*
CDNetworks	1388.60	13651.43	27445.86	1357456.36*
CDNSun	1698.61	16729.29	33194.18	1654906.12*
ChinaCache[1]	217.55	226.84	216.76	265.65
ChinaNetCenter	1364.86	12188.11	25755.39	1348074.75*
Cloudflare[2]	940.34	9333.86	18654.03	477756.97
Fastly[3]	1383.88	12735.87	27453.46	27453.46
Gcore	1669.96	16595.73	33264.03	1636764.26*
Huawei Cloud	1144.63	11311.80	22565.36	1126143.05*
Udomain[4]	1588.88	1447.45	1445.06	1588.88

* Amplification factor can be greater if the file size is larger than 1GB.
[1] Terminate the request as soon as all the headers received.
[2] Terminate the request if the file size is larger than 512MB.
[3] Refuse with "503 Service Unavailable" if the file size is larger than 20MB.
[4] First request for the first 1MB of file, then response to the client with headers.

TABLE III: Amplification Factors with Different Target Resource Size of AEAmp Attacks.

CDN	Exploited Case	Amplification Factor		
		1MB	10MB	25MB
Baidu Cloud	`gzip;q=1`	580.44	946.17	—
CDN77	`gzip`	571.68	929.30	963.03
CDNSun	`gzip`	650.43	972.92	984.34
Udomain	`gzip`	202.03	227.95	230.10

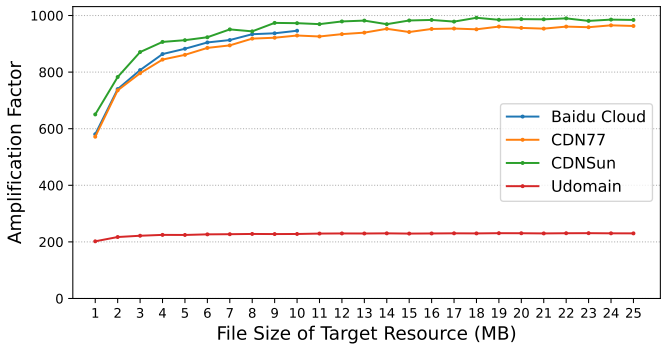


Fig. 12: Distribution of Amplification Factors for AEAmp Attacks with Different Target Resource Size and CDNs.

Based on the result in Figure 12, Udomain exhibits a relatively low compression level compared to other CDNs, thereby resulting in a lower amplification factor of 230. Furthermore, Baidu Cloud furnishes the uncompressed version of the file to the client when the target resource size exceeds 10MB, causing the AEAmp attack to fail. On the other hand, CDN77 and CDNSun manifest an amplification factor exceeding 960.

E. Impact Analysis

Widespread and Serious Impacts. According to our experimental results, the amplification factor of the HeadAmp attack and CondAmp attack can reach up to 1 million, while that of AEAmp can reach close to 1,000. As shown in Table IV, we found 74 vulnerabilities of HTTP amplification attacks across 19 CDN vendors, which can potentially affect multiple websites that use these CDNs worldwide.

Efficient and Low-cost DDoS Attacks. The attacker in our threat model does not require a huge botnet like traditional DDoS attacks. Instead, the attacker can use a standard laptop and leverage the egress nodes of CDNs worldwide to conduct the attack. Even if there's a shared cache among nodes, the attacker can employ cache-busting strategies to negate the effect [24]. This makes the overall cost of the attack very low for the attacker.

TABLE IV: CDN Vendors Vulnerable to Three HTTP Amplification Attacks.

CDN	Head-Amp	CondAmp					AE-Amp
		M.[1]	N.-M.[2]	M.-S.[3]	Un.-S.[4]	R.[5]	
Akamai					✓		
Aliyun	✓		✓	✓		✓	
Azure	✓	✓	✓	✓	✓	✓	
Baidu Cloud		✓	✓	✓	✓	✓	✓
BunnyCDN	✓	✓	✓		✓	✓	
CDN77	✓						✓
CDNetworks	✓	✓	✓	✓	✓	✓	
CDNSun		✓	✓	✓		✓	✓
ChinaCache						✓	
ChinaNetCenter	✓	✓	✓	✓	✓	✓	
Cloudflare	✓	✓	✓	✓		✓	
CloudFront				✓			
Fastly	✓		✓	✓		✓	
Gcore	✓	✓	✓	✓	✓	✓	
Huawei Cloud		✓	✓	✓	✓	✓	
KeyCDN	✓						
Qiniu Cloud			✓	✓			
StackPath	✓						
Udomain	✓	✓	✓	✓	✓	✓	✓

✓: The target CDN is vulnerable. [1] `If-Match`. [2] `If-None-Match`. [3] `If-Modified-Since`. [4] `If-Unmodified-Since`. [5] `If-Range`.

Huge Financial Losses for Victims. These attacks can cause website or application outages, financial losses, and harm the victim's business reputation. The resulting losses for victims may be significant, as they might have to purchase additional bandwidth or DDoS security services, as well as bear the costs of analyzing and mitigating the attack.

VI. Discussion

A. Limitation

Our experiments only tested the default configuration of the CDN, and we may not have uncovered all potential forwarding request inconsistencies since some CDN configurations may differ in real-world deployments. Additionally, we did not manually enable DoS protection for all CDN vendors, as some features are turned off by default or only available to commercial customers. However, our findings are still valuable because attackers can abuse CDNs to attack websites [23], and as a malicious CDN user, the attacker can modify the caching policy and disable security measures to make the attack more effective.

Extending the use of ReqsMiner to detect other threat models might introduce new challenges. For instance, if one intends to utilize ReqsMiner to identify attack vectors potentially leading to Web Cache Poisoning (WCP), it will necessitate the additional consideration of CDN caching policies. Hence, bridging the gap from detected inconsistencies to pinpointing potential security impact remains a complex task.

In our experiments, we set a 300-minute duration for the fuzzing test. This decision was informed by our observations that the exploration of ABNF grammar tree nodes reached a saturation point at this juncture, implying that ReqsMiner had generated nearly all legal HTTP grammar structures. This highlights not only efficiency but also limitation. Although it is conceivable that prolonging the testing period might unearth previously undiscovered cases, we believe that the likelihood of such an occurrence is minuscule.

Moreover, due to restrictions imposed by the ABNF rules, ReqsMiner does not generate malformed malicious requests. As a result, part of request forwarding inconsistencies may evade detection. Nevertheless, as in manual attempts we found, this is not a significant concern since most CDN security mechanisms will automatically reject requests that contravene ABNF rules.

Regarding the manual efforts, the set of ABNF rules and values using in our experiments were extracted manually by reading the RFC documentation. This process took us approximately two days of dedicated effort. However, we acknowledge the possibility of unintentional omissions or extraction errors in our manual extraction process.

B. Root Cause and Mitigation

CDN forwarding request inconsistencies predominantly underpin the vulnerabilities identified by ReqsMiner. To efficiently support a range of protocols and operational mechanisms, optimize cache hits, and circumvent forwarding duplicate requests to the backend server, CDNs modify requests. By normalizing requests through such modifications, CDNs can proficiently manage content delivery and diminish potential inconsistencies between themselves and the backend webserver. It is important to note that these inconsistencies are often introduced by CDNs for commercial or security reasons.

Implementation of preventive measures is challenging due to the fact that inconsistencies do not invariably lead to security risks. Consequently, we advocate for the following strategies to mitigate the risks associated with inconsistent forwarding requests: *(i)* Security assessment. CDNs must carefully evaluate their modifications and ensure they do not negatively impact the security of the origin server; *(ii)* Request inspection. CDNs should inspect forwarding request inconsistencies in real-time, and monitor the resulting abnormal traffic; *(iii)* Risks informed. The CDNs should inform and educate users about the risks associated with certain features that may lead to forwarding request inconsistencies (e.g., whether to ignore the query string) and recommend implementing a more secure caching policy to mitigate those risks. Besides, for CDN users, turning on DDoS protection can also mitigate the risk of attacks brought by CDN forwarding request inconsistencies.

C. Disclosure

We have reported all vulnerabilities to affected CDN vendors, and our experiments are authorized for vendors that replied. We provide them with mitigation solutions to eliminate potential risks. We have received feedback from 3 vendors, some of which, e.g., Azure and Cloudflare, have confirmed the vulnerabilities and planned to fix them. We are still waiting for responses from other vendors.

Azure. They confirmed the vulnerability. For HeadAmp attacks, they stated that the request method was changed to satisfy customer origins that could not accept HEAD requests. Their patch strategy involves just forwarding HEAD requests from the user to the customer origin. They indicate that this fix is underway.

Cloudflare. They acknowledged the vulnerability. They said they already have a defense rule against these attacks; however,

the thresholds are undergoing revision. They also thanked us for the report and awarded us a bounty.

Fastly. They thanked us for our report but believed it was a configuration issue with the client. Fastly provides customers with configurable options to control the cached file size and disable cache bursting techniques. They said that they would gladly reconsider if we could demonstrate that this attack would still be successful even with these mitigation strategies in place. We are still in discussion with them about providing new evaluation results.

VII. Related Work

CDN Forwarding Request Inconsistencies. In recent years, several works studied CDN forwarding request inconsistencies, such as forwarding loop attacks [11], RangeAmp attacks using Range headers [32], amplification attacks using the HTTP/2 header compression mechanism [25], and the inconsistent interpretations of requests between the CDN and the origin server also lead to CPDoS attacks [41], web cache deception attacks [37], [38], and HTTP desync attacks [28]. However, these works rely on manual analysis, a process that is heavily dependent on human knowledge and expertise and lacks scalability This approach is inherently limited, as it is time-consuming and prone to human errors, and it cannot adapt to the increasing complexity of security issues.

Unlike the existing study, which is fragmented and issue-specific, our work provides a more holistic view of the security landscape, enabling us to identify and address a wider range of vulnerabilities. It is the first to conduct a systematic and comprehensive examination of the security issues associated with CDN forwarding request inconsistencies. By using semi-automating the detection framework, we can significantly enhance the scalability and efficiency of vulnerability discovery, thereby providing a more robust and effective protection to the CDN.

Grammar-based Fuzzing. Grammar-based Fuzzing has also been used in many works to automate vulnerability detection [12], [26], [45], [48], [51]. One such effort is T-Reqs [26], which utilizes grammar-based fuzzing to uncover HRS attacks. However, T-Reqs primarily focuses on generating malformed HTTP requests and produces requests that are less than one-thousandth as effective, which results in it taking nearly 100 hours to generate enough valid requests. Instead of employing the original ABNF rules found in the RFCs, T-Reqs employs a distinct set of context-free grammars (CFGs) that they construct themselves for generating test cases, which means it requires more manual effort. In another study, HDiff [45] applies NLP-based techniques to extract ABNF rules from RFCs and construct test cases. However, due to the inherent uncertainty of NLP, it still needs to manually verify all ABNF rules for critical errors prior to experimentation. This verification process introduces a significant amount of manual effort.

In contrast, ReqsMiner constructs test cases using unchanged ABNF rules in RFCs and real data from server traffic logs to reduce manual efforts. Users only need to manually copy the ABNF rules that can be typically found in the appendix from the RFCs, and provide an adequate amount of traffic logs. ReqsMiner also allows users to further incorporate expert knowledge by specifying specific ABNF variable values, thus enhancing the extensibility of the framework. Additionally, ReqsMiner employs a UCT-based fuzzer to improve the efficiency of detecting vulnerabilities in black box systems such as CDNs, so that the entire testing process can be completed within a few hours.

VIII. Conclusion

This study is the first systematic exploration of CDN forwarding request inconsistencies. We proposed a framework, ReqsMiner, for the efficient discovery of CDN forwarding request inconsistencies, and developed a novel MCTS-based grammar-based fuzzer. We tested 22 popular CDNs and found numerous request differences, and further discovered three novel high-impact HTTP traffic amplification attacks, involving a total of 74 vulnerabilities on 19 CDNs. Beyond amplification DoS attacks, CDN forwarding request inconsistencies harbor the potential to morph into more sophisticated forms of attack, such as CPDoS and WCP. We believe that ReqsMiner possesses significant potential for deployment in the detection of these complicated attacks.

We hope this work can act as a catalyst, galvanizing the security community to accord serious attention to the issue of CDN forwarding request inconsistency. We advocate for the proactive detection and defense against the security risks that may stem from these inconsistencies, thereby enhancing the overall security landscape of the CDNs.

Acknowledgement

We express our deep gratitude to all the anonymous reviewers and our shepherd. Their insightful reviews and comments have improved this paper, with a special note of appreciation to our shepherd for their thoughtful and patient guidance. We also acknowledge the prompt response from CDN vendors, notably Azure and Cloudflare, in fixing the vulnerabilities we found. Lastly, we appreciate the invaluable assistance of Qianhui Wang in editing this paper. This work was partly supported by the National Natural Science Foundation of China (grant #62272265). Any opinions, findings, conclusions, or recommendations expressed in this paper do not necessarily reflect the views of sponsors.

REFERENCES

[1] B. Abramson and R. E. Korf, "A model of two-player evaluation functions," in *Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, USA, July 1987*, K. D. Forbus and H. E. Shrobe, Eds. Morgan Kaufmann, 1987, pp. 90–94. [Online]. Available: http://www.aaai.org/Library/AAAI/1987/aaai87-016.php

[2] Akamai, "What is a CDN (Content Delivery Network)?" 2023. [Online]. Available: https://www.akamai.com/our-thinking/cdn/what-is-a-cdn

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, 2002. [Online]. Available: https://doi.org/10.1023/A:1013689704352

[4] H. Baier and M. Kaisers, "Guiding multiplayer MCTS by focusing on yourself," in *IEEE Conference on Games, CoG 2020, Osaka, Japan, August 24-27, 2020*. IEEE, 2020, pp. 550–557. [Online]. Available: https://doi.org/10.1109/CoG47356.2020.9231603

[5] A. Barbir, B. Cain, R. Nair, and O. Spatscheck, "Known content network (CN) request-routing mechanisms," *RFC*, vol. 3568, pp. 1–19, 2003. [Online]. Available: https://doi.org/10.17487/RFC3568

[6] T. Berners-Lee, R. T. Fielding, and L. Masinter, "Uniform resource identifier (URI): generic syntax," *RFC*, vol. 3986, pp. 1–61, 2005. [Online]. Available: https://doi.org/10.17487/RFC3986

[7] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen, "Hypertext transfer protocol - HTTP/1.0," *RFC*, vol. 1945, pp. 1–60, 1996. [Online]. Available: https://doi.org/10.17487/RFC1945

[8] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. P. Liebana, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012. [Online]. Available: https://doi.org/10.1109/TCIAIG.2012.2186810

[9] BuiltWith, "Content Delivery Network Usage Statistics," 2023. [Online]. Available: https://trends.builtwith.com/CDN/Content-Delivery-Network

[10] Chaitin Tech, "Xray: A powerful security assessment tool," 2023. [Online]. Available: https://github.com/chaitin/xray

[11] J. Chen, X. Zheng, H. Duan, J. Liang, J. Jiang, K. Li, T. Wan, and V. Paxson, "Forwarding-loop attacks in content delivery networks," in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/forwarding-loop-attacks-content-delivery-networks.pdf

[12] S. R. Choudhary, M. R. Prasad, and A. Orso, "Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications," in *Fifth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Montreal, QC, Canada, April 17-21, 2012*, G. Antoniol, A. Bertolino, and Y. Labiche, Eds. IEEE Computer Society, 2012, pp. 171–180. [Online]. Available: https://doi.org/10.1109/ICST.2012.97

[13] Cloudflare, "What is a CDN? — How do CDNs work?" 2022. [Online]. Available: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

[14] ——, "HTTP request headers — Cloudflare Fundamentals docs," 2023. [Online]. Available: https://developers.cloudflare.com/fundamentals/get-started/reference/http-request-headers/

[15] D. Crocker and P. Overell, "Augmented BNF for syntax specifications: ABNF," *RFC*, vol. 5234, pp. 1–16, 2008. [Online]. Available: https://doi.org/10.17487/RFC5234

[16] A. Czechowski and F. A. Oliehoek, "Decentralized MCTS via learned teammate models," *CoRR*, vol. abs/2003.08727, 2020. [Online]. Available: https://arxiv.org/abs/2003.08727

[17] Fastly, "Header reference — Fastly Developer Hub," 2023. [Online]. Available: https://developer.fastly.com/reference/http/http-headers/

[18] R. T. Fielding, M. Nottingham, and J. F. Reschke, "HTTP caching," *RFC*, vol. 9111, pp. 1–35, 2022. [Online]. Available: https://doi.org/10.17487/RFC9111

[19] ——, "HTTP semantics," *RFC*, vol. 9110, pp. 1–194, 2022. [Online]. Available: https://doi.org/10.17487/RFC9110

[20] ——, "HTTP/1.1," *RFC*, vol. 9112, pp. 1–46, 2022. [Online]. Available: https://doi.org/10.17487/RFC9112

[21] R. T. Fielding and J. F. Reschke, "Hypertext transfer protocol (HTTP/1.1): semantics and content," *RFC*, vol. 7231, pp. 1–101, 2014. [Online]. Available: https://doi.org/10.17487/RFC7231

[22] P. Godefroid, "Fuzzing: hack, art, and science," *Commun. ACM*, vol. 63, no. 2, pp. 70–76, 2020. [Online]. Available: https://doi.org/10.1145/3363824

[23] R. Guo, J. Chen, B. Liu, J. Zhang, C. Zhang, H. Duan, T. Wan, J. Jiang, S. Hao, and Y. Jia, "Abusing cdns for fun and profit: Security issues in cdns' origin validation," in *37th IEEE Symposium on Reliable Distributed Systems, SRDS 2018, Salvador, Brazil, October 2-5, 2018*. IEEE Computer Society, 2018, pp. 1–10. [Online]. Available: https://doi.org/10.1109/SRDS.2018.00011

[24] R. Guo, J. Chen, Y. Wang, K. Mu, B. Liu, X. Li, C. Zhang, H. Duan, and J. Wu, "Temporal cdn-convex lens: A cdn-assisted practical pulsing ddos attack," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 6185–6202. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/guo-run

[25] R. Guo, W. Li, B. Liu, S. Hao, J. Zhang, H. Duan, K. Sheng, J. Chen, and Y. Liu, "CDN judo: Breaking the CDN dos protection with itself," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/cdn-judo-breaking-the-cdn-dos-protection-with-itself/

[26] B. Jabiyev, S. Sprecher, K. Onarlioglu, and E. Kirda, "T-reqs: HTTP request smuggling with differential fuzzing," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 1805–1820. [Online]. Available: https://doi.org/10.1145/3460120.3485384

[27] B. Kartal, E. Nunes, J. Godoy, and M. L. Gini, "Monte carlo tree search for multi-robot task allocation," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, D. Schuurmans and M. P. Wellman, Eds. AAAI Press, 2016, pp. 4222–4223. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12154

[28] J. Kettle, "HTTP Desync Attacks : Smashing into the Cell Next Door Core concepts," PortSwigger Web Security, Tech. Rep., 2019. [Online]. Available: https://i.blackhat.com/USA-19/Wednesday/us-19-Kettle-HTTP-Desync-Attacks-Smashing-Into-The-Cell-Next-Door-wp.pdf

[29] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Springer, 2006, pp. 282–293. [Online]. Available: https://doi.org/10.1007/11871842_29

[30] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved monte-carlo search," *Univ. Tartu, Estonia, Tech. Rep*, vol. 1, 2006.

[31] Y. Labbé, S. Zagoruyko, I. Kalevatykh, I. Laptev, J. Carpentier, M. Aubry, and J. Sivic, "Monte-carlo tree search for efficient visually guided rearrangement planning," *IEEE Robotics Autom. Lett.*, vol. 5, no. 2, pp. 3715–3722, 2020. [Online]. Available: https://doi.org/10.1109/LRA.2020.2980984

[32] W. Li, K. Shen, R. Guo, B. Liu, J. Zhang, H. Duan, S. Hao, X. Chen, and Y. Wang, "CDN backfired: Amplification attacks based on HTTP range requests," in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*. IEEE, 2020, pp. 14–25. [Online]. Available: https://doi.org/10.1109/DSN48063.2020.00022

[33] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 67–82. [Online]. Available: https://doi.org/10.1109/SP.2014.12

[34] A. Liu, T. Wu, I. Wu, H. Guei, and T. Wei, "Strength adjustment and assessment for mcts-based programs [research frontier]," *IEEE Comput. Intell. Mag.*, vol. 15, no. 3, pp. 60–73, 2020. [Online]. Available: https://doi.org/10.1109/MCI.2020.2998315

[35] R. F. Mark Nottingham, "Hypertext Transfer Protocol

(HTTP) Field Name Registry," 2022. [Online]. Available: HypertextTransferProtocol(HTTP)FieldNameRegistry

[36] J. Mern, A. Yildiz, Z. Sunberg, T. Mukerji, and M. J. Kochenderfer, "Bayesian optimized monte carlo planning," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 11 880–11 887. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17411

[37] S. A. Mirheidari, S. Arshad, K. Onarlioglu, B. Crispo, E. Kirda, and W. Robertson, "Cached and confused: Web cache deception in the wild," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 665–682. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/mirheidari

[38] S. A. Mirheidari, M. Golinelli, K. Onarlioglu, E. Kirda, and B. Crispo, "Web cache deception escalates!" in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 179–196. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/mirheidari

[39] MordorIntelligenc, "Content Delivery Network (CDN) Market - Growth, Trends, COVID-19 Impact, and Forecasts (2023 - 2028)," 2022. [Online]. Available: https://www.mordorintelligence.com/industry-reports/content-delivery-market

[40] M. M. Nesary and A. Aydeger, "vdns: Securing DNS from amplification attacks," in *10th IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2022, Sofia, Bulgaria, June 6-9, 2022*. IEEE, 2022, pp. 102–106. [Online]. Available: https://doi.org/10.1109/BlackSeaCom54372.2022.9858278

[41] H. V. Nguyen, L. L. Iacono, and H. Federrath, "Your cache has fallen: Cache-poisoned denial-of-service attack," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 1915–1936. [Online]. Available: https://doi.org/10.1145/3319535.3354215

[42] A. Phillips and M. Davis, "Matching of language tags," *RFC*, vol. 4647, pp. 1–20, 2006. [Online]. Available: https://doi.org/10.17487/RFC4647

[43] ——, "Tags for identifying languages," *RFC*, vol. 5646, pp. 1–84, 2009. [Online]. Available: https://doi.org/10.17487/RFC5646

[44] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," in *Proceedings of the 2nd International Conference on Distributed Computing Systems, Paris, France, 1981*. IEEE Computer Society, 1981, pp. 509–512.

[45] K. Shen, J. Lu, Y. Yang, J. Chen, M. Zhang, H. Duan, J. Zhang, and X. Zheng, "Hdiff: A semi-automatic framework for discovering semantic gap attack in HTTP implementations," in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022, Baltimore, MD, USA, June 27-30, 2022*. IEEE, 2022, pp. 1–13. [Online]. Available: https://doi.org/10.1109/DSN53405.2022.00014

[46] H. Siewert, M. Kretschmer, M. Niemietz, and J. Somorovsky, "On the security of parsing security-relevant HTTP headers in modern browsers," in *43rd IEEE Security and Privacy, SP Workshops 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 342–352. [Online]. Available: https://doi.org/10.1109/SPW54247.2022.9833880

[47] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nat.*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: https://doi.org/10.1038/nature16961

[48] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana, "Hvlearn: Automated black-box analysis of hostname verification in SSL/TLS implementations," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 521–538. [Online]. Available: https://doi.org/10.1109/SP.2017.46

[49] S. Triukose, Z. Al-Qudah, and M. Rabinovich, "Content delivery networks: Protection or threat?" in *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*, ser. Lecture Notes in Computer Science, M. Backes and P. Ning, Eds., vol. 5789. Springer, 2009, pp. 371–389. [Online]. Available: https://doi.org/10.1007/978-3-642-04444-1_23

[50] C. Xu, J. Li, and J. Liu, "Yet another traffic black hole: Amplifying CDN fetching traffic with rangefragamp attacks," in *Collaborative Computing: Networking, Applications and Worksharing - 17th EAI International Conference, CollaborateCom 2021, Virtual Event, October 16-18, 2021, Proceedings, Part I*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, H. Gao and X. Wang, Eds., vol. 406. Springer, 2021, pp. 439–459. [Online]. Available: https://doi.org/10.1007/978-3-030-92635-9_26

[51] H. Yoo and T. Shon, "Grammar-based adaptive fuzzing: Evaluation on SCADA modbus protocol," in *2016 IEEE International Conference on Smart Grid Communications, SmartGridComm 2016, Sydney, Australia, November 6-9, 2016*. IEEE, 2016, pp. 557–563. [Online]. Available: https://doi.org/10.1109/SmartGridComm.2016.7778820

[52] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler, "The Fuzzing Book," 2019. [Online]. Available: https://www.fuzzingbook.org/

[53] Y. Zhao, X. Wang, L. Zhao, Y. Cheng, and H. Yin, "Alphuzz: Monte carlo search on seed-mutation tree for coverage-guided fuzzing," in *Annual Computer Security Applications Conference, ACSAC 2022, Austin, TX, USA, December 5-9, 2022*. ACM, 2022, pp. 534–547. [Online]. Available: https://doi.org/10.1145/3564625.3564660