USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

H3Act: Automated Measuring Semantic Conversion Anomalies of HTTP/3-to-HTTP/1.1 Translation in CDNs

Qihang Peng and Siyuan Tian, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University;* Yongxin Qiu, *Beihang University;* Jinyang Huang, *Central South University;* Yaru Yang, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University;* Xiang Li, *Nankai University; Zhongguancun Laboratory;* Jia Zhang, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University; Zhongguancun Laboratory;* Yiming Zhang, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University;* Haixin Duan, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University; Quancheng Laboratory;* Yunsenxiao Lin and Shugen Chen, *Tencent Technology Co., Ltd.;* Liqun Yang, *Beihang University*

https://www.usenix.org/conference/usenixsecurity26/presentation/peng-qihang

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

H3Act: Automated Measuring Semantic Conversion Anomalies of HTTP/3-to-HTTP/1.1 Translation in CDNs

Qihang Peng[1], Siyuan Tian[1], Yongxin Qiu[2], Jinyang Huang[3], Yaru Yang[1], Xiang Li[4,5,✉], Jia Zhang[1,5,✉], Yiming Zhang[1], Haixin Duan[1,6], Yunsenxiao Lin[7], Shugen Chen[7], Liqun Yang[2]
[1]Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, China
[2]Beihang University, [3]Central South University, [4]Nankai University
[5]Zhongguancun Laboratory, [6]Quancheng Laboratory, [7]Tencent Technology Co., Ltd.

Abstract

Content Delivery Networks (CDNs) are adopting HTTP/3 to enhance performance; however, they often need to convert it to HTTP/1.1 for compatibility. This conversion creates significant attack surfaces and may reintroduce confirmed or even patched vulnerabilities in HTTP/2 or HTTP/1. Unfortunately, existing tools struggle to adapt to the HTTP/3 environment and efficiently leverage accumulated attack knowledge. To overcome these challenges and systematically measure HTTP/3-to-HTTP/1.1 conversion anomalies within the black-box CDN environment, we present H3Act, a dual-agent, knowledge-driven fuzzing framework targeting HTTP/3-to-HTTP/1.1 semantic conversion anomalies. Our approach combines Large Language Models (LLMs) with Hybrid Retrieval-Augmented Generation (RAG) to automatically transform protocol specifications and historical threat intelligence into high-precision HTTP/3 test payloads, enabling regression testing of semantic translation risks. In a large-scale study of 9 commercial CDNs, including Cloudflare, Cloudfront, and Tencent CDN, we found a significant regression in protocol security, which means vulnerabilities in old protocols are reintroduced in HTTP/3. Our research identified 7 common attack vectors across various categories, including request smuggling, cache poisoning, and Denial-of-Service (DoS) amplification. Every CDN is vulnerable to at least one attack vector. We have responsibly disclosed these vulnerabilities and have received confirmations from some vendors. These findings highlight that the CDN ecosystem currently lacks the capacity to maintain security consistency checks while pursuing improvements in protocol performance.

1 Introduction

Content Delivery Networks (CDNs), an essential part of Internet infrastructure, need to continuously translate between modern protocols (e.g., HTTP/3) and traditional protocols (e.g., HTTP/1.1) to balance performance and compatibility. This translation introduces complex intermediate states that significantly expand the attack surface. Specifically, in the context of HTTP/2-to-HTTP/1.1 translation, both academia and industry have identified critical vulnerabilities, such as request smuggling and bandwidth amplification [24, 28, 32].

While HTTP/3 adoption is accelerating due to the performance benefits of QUIC, its security posture within CDN protocol translation scenarios remains underexplored. Although HTTP/3 maintains high semantic consistency with HTTP/2 [6, 16, 57], this similarity often fosters a false sense of security. Developers may assume that security flaws already patched in the HTTP/2-to-HTTP/1 translation are inherently absent in the HTTP/3-to-HTTP/1 context. However, the migration from TCP to QUIC requires a fundamental reimplementation of the protocol stack. This contradiction, arising from semantic inheritance through implementation reconstruction, may cause new code to fail to inherit historical security patches, thereby regressing known protocol semantic conversion anomalies. The regression of old attacks is not an exaggeration, as demonstrated by previous studies [7, 48]. Therefore, a systematic measurement of HTTP/3 security in real-world CDN deployments is urgently needed.

However, assessing this risk presents unique challenges. First, new mechanisms introduced in HTTP/3 hinder the direct migration of existing attack payloads from HTTP/2 or HTTP/1 [7], and transforming the textual knowledge associated with these payloads into usable HTTP/3 test cases incurs substantial manual effort [56, 62]. Second, detecting anomalies in protocol semantic translation requires a deep understanding of semantics [56, 60, 62, 63]. During detection, tools should "understand" request intent to capture subtle logical discrepancies between messages before and after translation. Third, as black-box environments, CDNs provide almost no additional effective feedback. Traditional rule-based fuzzing methods cannot directly leverage knowledge from documentation, nor can they effectively capture semantic understanding during testing. Consequently, they require significant manual effort to translate knowledge into generation and detection rules [28, 29, 56, 62, 63]. Meanwhile, white-box or grey-box testing lacks feedback signals in CDN environments.

Large Language Models (LLMs) offer a unique solution to these challenges. They leverage a deep understanding of textual knowledge, reasoning capabilities for cross-protocol semantic mapping, and the creativity to generalize new attack vectors from historical data [40, 41, 62, 64]. LLMs convert abstract semantic constraints from RFCs and attack primitives from papers and reports into test strategies. They infer internal translation logic by analyzing differences in messages across black-box environments, thereby systematically exploring semantic vulnerabilities within the protocol translation layer.

To overcome these challenges and systematically measure HTTP/3-to-HTTP/1.1 conversion anomalies within the black-box CDN environment, we propose H3Act, a knowledge-driven fuzzing framework that targets HTTP/3-to-HTTP/1.1 semantic conversion anomalies. We employ a dual-agent fuzzing architecture to generate HTTP/3 test requests and analyze logs to identify conversion anomalies and guide the generation process. To equip agents with the necessary expert knowledge, we implement a hybrid RAG architecture. This system statically injects existing attack primitives into the system prompt and integrates an external dynamic-retrieval tool, enabling agents to consult sliced and vectorized RFC documents. We evaluated the runtime performance and false positive rate of H3Act. Our results show that H3Act maintains an average false positive rate between 10.3% and 15.8%, which falls within an acceptable range.

Using this framework, we evaluate 9 commercial CDNs across 7 major CDN providers and identify 7 attack vectors, including DoS, cache poisoning, and request smuggling. Every CDN is vulnerable to at least one vector. We have conducted responsible disclosure; Baidu and Tencent have confirmed our findings, while other CDN providers are currently assessing the associated risks and impacts. This study confirms the resurgence of protocol translation risks in the HTTP/3 era.

Contributions. Specifically, our contributions are as follows:

Additionally, we validate the effectiveness of our architecture through ablation studies.

1. We conduct empirical measurements across major commercial CDNs, confirming the prevalence of protocol translation vulnerabilities and highlighting the challenge of security regression faced by CDN vendors during protocol upgrades.

2. To conduct these measurements, we propose H3Act, a knowledge-driven fuzzing framework that efficiently migrates existing protocol-semantic threat intelligence from HTTP/3 to HTTP/1.1.

2 Background

2.1 HTTP/3 & QUIC

Defined in RFC 9114 [6], HTTP/3 is the third major version of the Hypertext Transfer Protocol. Its most significant feature is the transition from TCP to QUIC at the transport layer. Initially termed "HTTP/2-over-QUIC," this name implies a high degree of semantic similarity between HTTP/3 and HTTP/2 from the outset. However, to prevent confusion, the IETF decoupled the application and transport layers in the final specification: QUIC was established as a general-purpose transport protocol, while HTTP/3 was defined as the application-layer protocol that uses it.

The IETF released the QUIC standard [26] first, followed by the HTTP/3 standard one year later. Concurrent with RFC 9114, the IETF redefined HTTP semantics in RFC 9110 [16] and caching in RFC 9111 [14]. Based on these new specifications, the IETF redefined HTTP/1.1 [15] and HTTP/2 [57] and formulated HTTP/3. This consolidation resolved the previous fragmentation caused by numerous scattered documents [3,5,12,13,17–20,51–54]. According to Cloudflare [4], approximately 21% of global requests directed to their network in 2025 utilize HTTP/3. Furthermore, W3Techs [59] reports that roughly 36.8% of websites support HTTP/3 as of late December 2025.

Defined in RFC 9000, QUIC is a next-generation transport layer protocol. Built upon UDP, QUIC provides reliability comparable to TCP while effectively resolving the Head-of-Line blocking issue inherent in TCP, thereby significantly reducing latency. Additionally, QUIC integrates and mandates the use of TLS 1.3 to encrypt application-layer messages and protect the content of communications.

2.2 CDN Overview

CDNs are a critical component of modern internet infrastructure. Due to their massive scale and scheduling mechanisms, CDNs are favored for three benefits: accelerating global content delivery, concealing the identities of origin servers, and defending against DoS attacks. According to 6sense's data [1], more than 6 million companies worldwide use CDNs.

The CDN architecture consists of edge nodes and central nodes, both of which are widely deployed globally [10]. Edge nodes serve as the ingress and egress points of the CDN, handling content caching and distribution tasks. Conversely, central nodes are responsible for functions such as load balancing. Furthermore, CDNs employ scheduling strategies, such as DNS-based or anycast-based scheduling, to ensure users access the nearest edge node. When a client accesses a server proxied by a CDN, the request is routed to the nearest ingress node. If the requested data hits the CDN cache and the replica is valid, the edge node returns the cached copy directly to the client. Otherwise, the CDN performs internal scheduling to initiate a request to the origin from an egress node (a process known as "back-to-origin"). The data retrieved is then returned to the client via the ingress node, which caches the resource for future use.

As demonstrated by this workflow, the globally distributed delivery and caching architecture effectively accelerates global content interaction. Since the client interacts exclusively with the CDN, the origin server's information re-

mains unexposed. Additionally, DoS attack traffic is distributed across multiple CDN nodes for filtering and scrubbing, thereby significantly mitigating its impact.

2.3 HTTP Attack in CDN

Recent studies identify various attacks targeting HTTP environments. Based on their techniques and impacts, these attacks fall into three broad categories: Denial of Service (DoS), cache poisoning, and request smuggling.

The Denial of Service (DoS) attack is focused on making a resource (site, application, server) unavailable for the purpose it was designed [47]. In CDN environments, attackers typically exhaust the bandwidth between edge nodes and origin servers (e.g., H2Amp [24], RangeAmp [34], CondAmp [63]) or consume all available connections (e.g., Slow Attack [24]). This prevents the CDN from retrieving necessary files from the origin, which disrupts normal service delivery. Other variants also exist. Attackers may disable routing nodes between the CDN edge and the origin server to block communication [24], or redirect users to suboptimal edge nodes to increase latency or render the service unavailable [25].

Cache poisoning targets the native caching mechanism of CDNs [46]. The goal is to force the CDN to store malicious responses. If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue to receive the malicious content until the cache entry is purged. The "Host of Troubles" attacks by Chen et al. [8] demonstrate this technique. Attackers craft requests that keep the cache key as victim.com, while the cache actually fetches and stores a malicious response from attacker.com. As a result, all subsequent users requesting victim.com through this cache receive the malicious content from attacker.com.

Request smuggling exploits parsing inconsistencies across different servers, particularly regarding request boundaries. This technique hides additional requests, which security policies might otherwise block, within a seemingly benign request [15, 61]. Kettle et al. [31–33] demonstrate several effective request smuggling techniques with real-world impact. A common approach exploits conflicts between the Content-Length and Transfer-Encoding headers to trigger parsing discrepancies. Notably, HTTP/2 does not use the Transfer-Encoding header. Therefore, when an HTTP/2 request containing this header is converted to HTTP/1.1, it introduces additional smuggling risks [32].

These three categories are not strictly mutually exclusive. For instance, attackers can leverage cache poisoning to perform request smuggling (CPDoS) [44], or use request smuggling to achieve cache poisoning [61].

3 Methodology of H3Act

In this section, we primarily present the threat model and the associated challenges. Subsequently, we expound on the architecture and workflow of H3Act. We further provide an in-depth discussion of H3Act's two modules: Hybrid RAG and Dual-Agent Fuzzing. Finally, we detail the implementation of H3Act.

3.1 Framework Overview

3.1.1 Threat Model and Challenge

We consider an attacker that can impersonate a legitimate client and issue HTTP/3 requests to the CDN ingress node without obstruction. The CDN processes these requests normally, modifying and forwarding them in accordance with its standard workflow when forwarding is required. The threat model is illustrated in Figure 1.

The attacker's capabilities are subject to the following constraints: 1. The attacker possesses no specific knowledge of the target origin and cannot access it directly. 2. The attacker cannot access the CDN's internal network or its source code. 3. The client can only craft HTTP/3 messages at the semantic layer; it cannot modify the transport layer (QUIC) or lower layers. 4. The attacker controls a server proxied by the CDN and uses it to poison the CDN cache.

Attackers aim to compromise the origin through improper semantic translation by the CDN, posing three primary threats: DoS, Cache Poisoning and Request Smuggling.

Detecting semantic vulnerabilities in this threat model requires extensive expert knowledge, a deep understanding of protocol semantics, and effective handling of black-box environments. Traditional rule-based or machine-learning-based fuzzers struggle to meet these requirements, motivating the use of LLMs in this study. We detail these challenges below.

Challenge 1: Difficulty in utilizing expert knowledge. Substantial attack knowledge applicable to this threat model already exists, though it is often embedded in texts such as academic papers, blogs, and vulnerability reports. Traditional fuzzers generally cannot extract knowledge directly from these sources and rely heavily on manual definitions [56, 62]. In contrast, LLMs possess strong natural language understanding capabilities. They can directly get knowledge from text, translate it into HTTP/3 requests, and generalize from existing concepts to discover novel vulnerabilities [40, 41, 62, 64].

Challenge 2: Complexity of semantic definitions and implementations. HTTP semantics involve complex definitions and implementations [56, 60, 63]. Some standards are strictly enforced with zero tolerance for errors. Simple random mutation or generation often produces numerous invalid test cases that are immediately discarded by parsers, hindering deep testing. Conversely, LLMs equipped with expert knowledge can comprehend protocol specifications and abstract rules. This allows them to construct targeted, malicious samples that pass

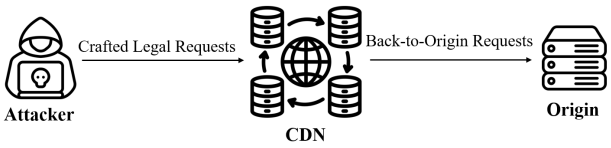


Figure 1: The Threat Model of CDN Attacks.

parser validity checks, thereby enabling more rigorous testing. Similarly, in the analysis task, LLMs can 'understand' request intent. This allows them to precisely capture subtle logical discrepancies between pre-translation and post-translation messages, thereby identifying anomalies.

Challenge 3: Adaptability to black-box environments. CDNs are typically black-box environments, lacking open-source code or accessible code coverage data [28, 29, 56, 60, 63]. This may prevent many fuzzers that rely on such information from functioning correctly, often resulting in blind searching. However, LLMs with strong logical reasoning capabilities can better grasp the context of protocol translation. They can identify internal translation logic from complex logs and infer potential "translation ambiguities" or mishandling, thereby enabling effective deep testing.

3.1.2 Framework and Workflow

To address the challenges, we propose H3Act, a fuzzing framework based on Hybrid RAG and Dual-agent Fuzzing. It aims to efficiently detect anomalies that arise during the semantic protocol conversion from HTTP/3 to HTTP/1.1 within CDNs. The overall framework adopts a closed-loop feedback approach. The architecture is illustrated in Figure 2.

The architecture comprises two components: Hybrid RAG and Dual-Agent Fuzzing. The Hybrid RAG module aggregates data from attack-related papers, blogs, and RFC documents to provide the two agents with attack knowledge and the ability to query specific RFC content. The Dual-Agent Fuzzing module is responsible for case generation and anomaly analysis during the fuzzing and possesses feedback guidance capabilities. The framework operates in batch mode to conduct testing against target CDNs. During the preliminary phase, expert knowledge from RAG is injected into both agents. For each batch, the Generator receives feedback from the Analyzer and uses it to generate a batch of valid HTTP/3 requests. These requests are transmitted to the CDN by the sender. Meanwhile, the framework captures essential data from the current batch's transmission, including client-side and origin-server logs, and submits them to the Analyzer. The Analyzer then analyzes the data to identify potential conversion anomalies, evaluates the quality of the current batch of cases, and proposes directions for further testing, thereby guiding the execution of the subsequent batch.

3.2 Hybrid RAG

To address the expert-knowledge requirements for fuzzing tasks, we propose a Hybrid RAG. We categorize expert knowledge into two types based on differences in timeliness, information length, and the base model's prior mastery.

Category 1: Attack Primitives. This knowledge is primarily scattered across academic papers, technical blogs, and vulnerability advisories. It is characterized by relatively short text length but high value. Furthermore, due to the rapid iteration of attack methods, base models struggle to cover the latest attack intelligence during pre-training. Consequently, their prior mastery of this knowledge is relatively low.

Category 2: Protocol Specifications. This primarily refers to RFCs. These documents are lengthy and detailed, including numerous state machines and error code definitions, yet they exhibit greater structure. We consider RFC documents to be high-quality corpora that are likely to be included in the LLM's pre-training dataset. While models typically possess a macroscopic understanding of protocols, they are prone to "Factuality Hallucination" when dealing with specific parameter values, obscure error codes, or edge-case state transitions.

Based on this analysis, we propose a hybrid strategy: static injection for attack primitives, and dynamic retrieval for protocol specifications.

3.2.1 Static Injection via System Prompt

We implement static injection for attack primitives by embedding them into the system prompt. Static injection is well-suited for these primitives not only because of their brevity but also because they should remain continuously active during the process to shape an attack mindset for each generation. These primitives represent discrete, high-value knowledge units; their value lies in providing transferable attack paradigms rather than descriptive details. Unlike dynamic retrieval, which exposes the model to knowledge only during specific queries and hinders stable mastery, static injection leverages In-Context Learning. This mechanism allows the model to use these primitives as a reasoning baseline for every generation. Furthermore, attack primitives demonstrate strong generalization, a CL.TE conflict in HTTP/2 may inspire the construction of inconsistencies between `Content-Length` and DATA frame lengths in HTTP/3. This capacity for cross-scenario transfer requires the knowledge to be resident in the model context rather than retrieved on demand.

We initially collect papers and blogs containing knowledge regarding HTTP attacks and vulnerabilities. This collection comprises approximately 18 documents [8, 9, 24, 27–34, 36, 37, 42, 43, 48, 56, 63]. During the preprocessing phase, we perform knowledge distillation on all documents. We focus exclusively on one key aspect of the described attacks and vulnerabilities: Principle, which is the underlying mechanism. Other information, such as impact scope and background context, is omitted from this distillation process. Ultimately, each

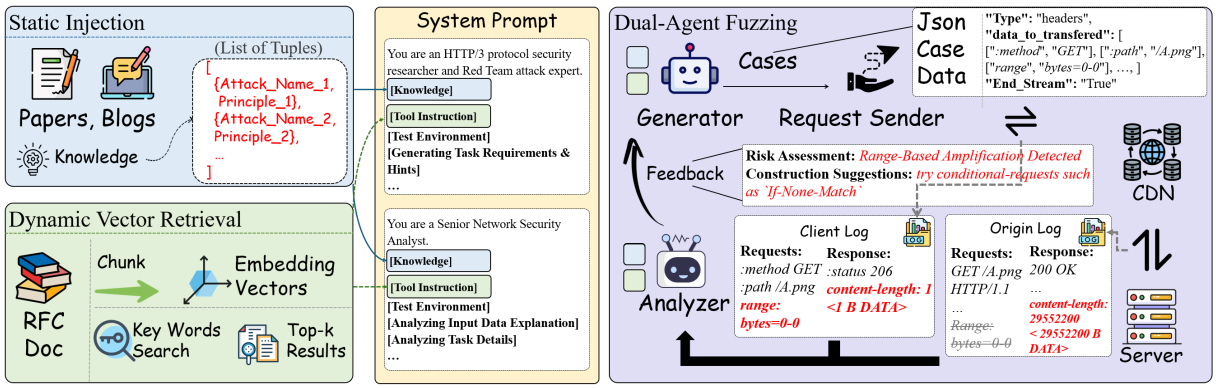


Figure 2: The Architecture of H3Act.

document is distilled into a list of tuples: `[{Attack Name, Principle}]`. The total knowledge content amounts to approximately 32,000 tokens, which falls well within the context window limits of mainstream LLMs. We integrate this static knowledge into the system prompt. It is injected into the model during the initial interaction batch, ensuring it learns domain-specific knowledge about attacks and vulnerabilities from the outset.

3.2.2 Dynamic Vector Retrieval

For long-text RFC protocol specifications and complex attack details, direct static injection results in significant waste of the context window and increases the risk of the model overlooking information. Therefore, we design a dynamic retrieval mechanism based on tool invocation, enabling the agent to "consult the manual."

The rationale for this design is that, despite having a general understanding of the HTTP/3 protocol, agents may still produce factuality hallucinations regarding protocol details. Dynamic retrieval enables the model to access raw RFC fragments on demand when precise specifications are required. This approach avoids context dilution from static injection and forces the model to 'consult the manual' via tool invocation, significantly reducing the risk of hallucination. Furthermore, the hierarchical structure of RFCs (chapters, subsections, clauses) naturally aligns with the semantic chunking of vector retrieval, enabling dynamic retrieval to balance precision and efficiency.

We construct a local vector database using ChromaDB. To address the hierarchical structure of RFC documents, we adopt a logic-based slicing strategy. Specifically, we perform an initial segmentation of the RFC text by section, formatting each slice as structured text: `{Source, Section ID, Description, Content}`. Additionally, we slice longer text paragraphs into logical segments. We embed these slices and distilled attack primitives in the database as well.

We encapsulate retrieval as a tool function callable by the agent Within the system prompt, we explicitly define the criteria for tool invocation: when the agent needs to verify specific RFC clauses, state machine transition rules, or error code definitions, it pauses the current generation process and issues a tool-invocation instruction. Upon receiving the search request, the system performs a Top-N similarity search in the vector database and returns the retrieved raw fragments to the agent as tool output. The agent then combines the context with this newly acquired, precise knowledge to complete packet construction.

3.3 Dual-Agent Fuzzing

We decompose the complex fuzzing task into two functionally orthogonal yet closely coordinating agents.

3.3.1 Necessity of Dual-Agent Architecture

We first emphasize the necessity of the dual-agent architecture in H3Act. This design addresses two primary issues: attention dilution from long contexts and conflicting temperature requirements across two tasks.

1. **Attention Dilution**: The dilution effect of long contexts motivates our decision to split the system into double agents. The attention of LLMs degrades as context length increases [39, 50, 58].

2. **Temperature Discrepancy across Two Tasks**: In fuzzing, the generation task requires a higher temperature to encourage divergent thinking and creative payload construction. Conversely, the analysis task requires a low-temperature, deterministic setting to interpret logs objectively and rigorously. The black box of CDNs reinforces this requirement. As the Analyzer only accesses logs from the client and origin, it should perform meticulous comparisons to detect even subtle anomalies without missing any details.

Assigning generation and analysis tasks to separate agents significantly mitigates context accumulation. We find that the Generator operates effectively as a stateless entity. In each batch, its input consists solely of the system prompt and the feedback from the previous batch, with all state maintenance

handled by the Analyzer. Compared to a stateful approach, this allows the Generator to focus more intently on the expert knowledge in the system prompt and on immediate feedback. This concentrated attention facilitates smoother execution of the generation task. Furthermore, the dual-agent setup enables distinct temperature settings optimized for each task.

In contrast, a single-agent architecture forces all conversation history into one model, accelerating context accumulation and diluting attention. Consequently, during the same fuzzing batch, the single agent performs worse in specific case generation compared to the dual-agent architecture. It also consumes more input tokens, thereby increasing costs. Finally, merging two orthogonal tasks into a single agent inevitably leads to 'role confusion' and complicates the implementation of differentiated temperature settings.

3.3.2 Generator

The Generator's duty is to apply divergent thinking to generate concrete test cases. It accepts feedback from the Analyzer as input and, by combining it with learned knowledge from RAG, produces executable cases that strictly adhere to the HTTP/3 protocol specifications.

Given the strict input format requirements of the transmission tool, the Generator should ensure absolute syntactic precision in its output. Consequently, we constrain the Generator to adhere to a predefined template. Furthermore, to address issues such as truncation or calculation errors common in LLMs when generating ultra-long strings, we introduce an Abstract Syntax Tree (AST)-based Dynamic Payload Generation mechanism. The Generator is permitted to use Python-style expressions (e.g., `{{"A"*10000+"B"*50}}`) within JSON fields to describe repetitive or patterned data. The system intercepts these expressions before transmission to execute safe local evaluation and expansion. This mechanism circumvents the LLM's Context Window Limit, facilitating the construction of substantial payloads with minimal token consumption.

3.3.3 Analyzer

The core responsibility of the Analyzer is to infer attack details from a diverse array of interaction logs. It aggregates the raw HTTP/3 cases produced by the Generator, client-side logs, and the converted HTTP/1.1 request and response logs from the origin. Leveraging its knowledge base, it analyzes these inputs to identify conversion anomalies that could potentially lead to attacks or other risks. By leveraging a moderate temperature setting, the Analyzer can infer potential attack surfaces from subtle discrepancies in protocol behavior.

Ultimately, the Analyzer generates not only a detailed risk assessment report identifying which potential attack surfaces were triggered by the current batch, but also, more importantly, guidance for the subsequent testing batch. This guidance serves as a feedback signal, transmitted in a closed loop to the Generator, directing it to refine the attack vector, thereby enabling feedback guidance during testing.

3.3.4 Externalized Memory Stream

Although task decomposition mitigates some resource constraints, the accumulated interaction history in continuous, multi-batch fuzzing rapidly exhausts the LLM's context window. To maintain the system's long-term memory capabilities without degrading performance, we design a State Distillation mechanism. This mechanism divides the memory stream into two distinct components:

1. **Short-term Memory.** This component uses a sliding-window mechanism. It retains full interaction details only for the most recent batches (e.g., the last 3), ensuring the Agent remains aware of the immediate testing context.

2. **Long-term Memory.** At fixed intervals (e.g., every 3 batches), the system triggers a "memory consolidation" operation. The Analyzer is invoked to perform an in-depth summary of past generations and testing histories. They extract a high-value "Technical Summary". The old raw conversation history for Generator and Analyzer will be flushed (retaining only the initial system prompt) and replaced with this highly condensed Technical Summary. This mechanism simulates the process of experience accumulation in human experts. It ensures that the agent maintains clarity regarding its objectives and direction even after long-duration, multi-batch testing, thereby avoiding redundant testing and resource waste.

3.4 Implementation

We implement our framework using `Python 3.12.7` [49].In the implementation, we configure the two agents with distinct temperatures and prompts: an initial system prompt and a batch prompt used per batch. Both prompts include sections for identity definition, task description, tool usage, environment configuration, input data explanation, output requirements, and task hints. However, the system prompt additionally incorporates statically injected knowledge. For each agent, shared sections between the system and batch prompts remain consistent. This leverages Ollama's KV Cache mechanism, allowing the agent to reuse precomputed cache data and maximize efficiency.

For the Generator's predefined templates, we use `Pydantic` to generate a JSON Schema. This schema defines the structure of HTTP/3 HEADERS and DATA frames, as well as the commands for issuing a sending pause. We inject this schema during each generation step, guiding the model to produce a JSON array that describes a complete HTTP/3 request. This array is subsequently converted into an executable HTTP/3 request for the sender. If the output violates the JSON Schema, the Generator automatically regenerates.

Regarding implementation details, we develop the sender using the `aioquic` library to ensure transport-layer correct-

ness during case transmission. Similarly, we use the python `ast` module to generate dynamic payloads.

Given the limitations of our local experimental environment and the model's performance, we select `gpt-oss:120b` [45] as the base model and deploy it locally via Ollama. The program interacts with the model through the Ollama interface.

4 Experiments and Evaluation

In this section, we first establish the complete experimental environment for H3Act, including the CDN configuration and the experimental platform settings. Subsequently, we conduct a comprehensive evaluation of H3Act in this environment, covering runtime performance, false-positive rate assessment, and ablation studies. Finally, we present a qualitative comparison of H3Act with other approaches.

4.1 Experiment Setup

4.1.1 Experimental Subject Overview

We select CDNs based on three criteria: market popularity, direct registration for individual users, and free or low-cost pricing. For market popularity, we aim to select widely studied CDNs [9, 22–24, 34, 35, 37, 38, 63]. For pricing, since our work spans a long period and involves multiple rounds of iteration and testing, for cost control purposes, we expect that the services should support monthly or usage-based billing, with an average monthly cost under $5. Based on these criteria, we exclude less popular CDNs, as well as providers such as Akamai, Google, and Microsoft, which either do not allow direct individual registration or charge higher fees. Finally, we selected 9 CDNs across 7 providers that support HTTP/3 client connections: Alibaba CDN, Alibaba ESA, Baidu CDN, Cloudflare, CloudFront, Fastly, Huawei CDN, Tencent CDN, and Tencent EdgeOne. Given their significant market share and influence, we believe security vulnerabilities in these services could expose a vast number of general users to threats.

We adopted a principle of minimal modification for CDN configurations. We primarily only enabled HTTP/3 functionality (which typically requires manual activation). We essentially left caching rules, durations, and other optimization settings unchanged, except for specific configurations required by our test cases, such as enabling `Range` requests.

4.1.2 Experiment Platform Setup

We establish the experimental platform using two separate devices. The first device functions as the client, running the H3Act and sending requests to the target CDNs. This device operates on Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-87-generic x86_64). Its hardware configuration includes dual Intel Xeon Gold 6326 CPUs (2.90 GHz, 16 cores, 32 threads), 192 GB of RAM, 10 Gbps of network bandwidth, and four NVIDIA GeForce RTX 4090 GPUs (450W TDP, 24,564MB VRAM). The second device serves as the origin, responsible for receiving forwarded requests from the CDNs and returning corresponding responses. It utilizes Nginx 1.24.0-2ubuntu7.5 as the web server software and runs on Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-88-generic x86_64). The hardware specifications include a single Intel Xeon Silver 4314 CPU (2.40 GHz, 8 cores, 8 threads), 8 GB of RAM, and 10 Gbps of network bandwidth, with no dedicated GPU configuration. We configured the Nginx logging module to capture received HTTP/1.1 requests and sent HTTP/1.1 responses. Note: Due to the large size of the target resource (which includes a 29MB PNG file), we solely record the byte count of the transmitted response body, omitting the actual content. The request and response headers are recorded fully.

4.2 Evaluation of Framework

4.2.1 Runtime performance

We conducted our evaluation by first assessing basic runtime performance. We identified a primary variable in our architecture: the number of execution batches. We selected three levels based on various factors: 10, 20, and 30 batches. We executed each configuration 10 times, yielding 30 experimental runs. For all evaluations, we consistently used Cloudflare, keeping other conditions, such as the knowledge base and base prompts, constant. We statistically analyzed the system's various runtime performance metrics, and the averaged results are presented in Table 1. The metrics are defined as follows: "Cases Generated" indicates the total number of valid HTTP/3 requests generated per run. "Agent Interactions" refers to the number of interactions with the agent per request, including calls to dynamic query tools. "Agent Time" represents the total time spent on the agent per run, covering both reasoning and response generation. "Sum Time" denotes the total execution time per run. "Tokens (Input)" and "Tokens (Output)" indicate the total number of input and output tokens, respectively, across all agents and interactions during a single run. By combining these token counts with official API pricing, users who plan to replace H3Act's base model with alternative APIs can estimate the expected cost per run.

4.2.2 Evaluation of False Positive Rate

We evaluated the accuracy of our approach. We conducted experiments on all 9 CDNs mentioned in Section 4.1.1. We perform five independent runs for each CDN. Each run comprises 10 batches, yielding a total of 45 experiments. We collected all evaluation reports from each batch, totaling 450.

To ensure rigorous evaluation, we defined "a successful discovery" of H3Act as the generation of a valid HTTP/3 request by the Generator, followed by the Analyzer confirming a specific anomaly derived from dual-ended logs. Human experts manually verify whether these high-confidence reports

Table 1: Runtime Performance of H3Act.

Batch	10	20	30
Cases Generated	74.4	136.5	215.6
Agent Interactions	36.7	72.5	106.1
Agent Time (s)	1405.2	2922.4	4477.2
Sum Time (s)	1903.6	3815.0	5846.2
Tokens (Input)	2068380.0	4291985.0	6633276.0
Tokens (Output)	74567.8	142442.4	222392.7

correspond to genuine anomalies. Specifically, three human experts grade the outputs. When an alert fires, they first check whether an anomaly has already been reported; if not, they examine the corresponding HTTP/3 request, the translated HTTP/1.1 request observed at the origin, and the logs. For skilled experts, reviewing a 10-batch run (approx. 70 requests as in Table 1) takes only about 5 minutes.

We used the False Positive Rate (FPR) as our metric. We defined FPR as the ratio of manually confirmed false alarms to the total number of anomalies reported by the model in a single experiment. The final result is the average of the five runs. We did not investigate whether false positives stem from agent hallucination or other causes. We treated all instances equally, relying solely on the reported results.

Table 2: FPR and Discovered Anomalies of H3Act running on CDNs.

CDN	Avg. Generated Cases	Avg. Unique Anomalies	Avg. FPR	Total Unique Anomalies	Avg. TTFF (Batch)
Alibaba CDN	72.8	4.2	11.4%	12	1.4
Alibaba ESA	74.0	3.6	10.3%	10	1.6
Baidu	76.2	4.0	12.5%	10	2.8
Cloudflare	79.8	3.0	11.0%	8	2.2
Cloudfront	75.8	3.4	15.8%	7	4.0
Fastly	74.8	4.0	12.4%	11	1.2
Huawei	72.0	4.2	12.6%	13	2.0
Tencent EdgeOne	71.6	3.8	14.0%	10	1.6
Tencent CDN	74.6	3.8	10.8%	13	1.2

We present the results in Table 2. The method maintains a high discovery capability, identifying an average of 3-4 unique anomalies per batch. Simultaneously, it has a false-positive rate ranging from 10.3% to 15.8%. Additionally, Time-to-First-Failure (TTFF) data indicates that the method typically detects the first anomaly within the first three batches, demonstrating its detection efficiency.

4.2.3 Ablation Study

We investigated the impact of single-agent versus dual-agent architectures and the inclusion of RAG on our method. Focusing on runtime performance, we selected Cloudflare as the experimental subject. We conducted 10 runs of 10 batches each on Cloudflare using two configurations: Single-Agent with RAG and Dual-Agent without RAG. We present the average results in Table 3. For all architectures, the generation component is stateless.

Table 3: Runtime Performance of Single-Agent with RAG and Dual-Agent without RAG.

Architecture	Dual-Agents with RAG	Single-Agent with RAG	Dual-Agents without RAG
Cases Generated	74.4	65.6	86.6
Agent Interactions	36.7	33.6	34.0
Agent Time (s)	1405.2	1353.2	1520.8
Sum Time (s)	1903.6	1879.7	2141.1
Tokens (Input)	2068380.0	2200551.0	843118.0
Tokens (Output)	74567.8	58027.2	70596.6

Regarding the number of agents, we observed that the single-agent architecture offers no significant advantage over the dual-agent architecture in terms of average generation time per sample or output token cost. However, its input cost is significantly higher.

We also evaluated the FPR of these architectures. To reduce workload, we selected the Huawei CDN and the Tencent CDN as targets. The experimental methodology and evaluation metrics remain consistent with Section 4.2.3. The results, as shown in Table 4, indicate that both the dual-agent structure and RAG reduce the false-positive rate and, to some extent, increase the number of detected anomalies.

Table 4: FPR and Discovered Anomalies of Two Architectures running on CDNs.

CDN	Architecture	Avg. Generated Cases	Avg. Unique Anomalies	Avg. FPR	Total Unique Anomalies	Avg. TTFF (Batch)
Huawei	Dual-Agent with RAG	72.0	4.2	12.6%	13	2.0
	Single-Agent with RAG	63.2	3.4	14.2%	9	2.0
	Dual-Agent without RAG	78.6	2.4	30.6%	7	2.4
Tencent CDN	Dual-Agent with RAG	74.6	3.8	10.8%	13	1.2
	Single-Agent with RAG	68.6	3.6	13.2%	8	1.6
	Dual-Agent without RAG	88.6	2.2	20.7%	5	2.2

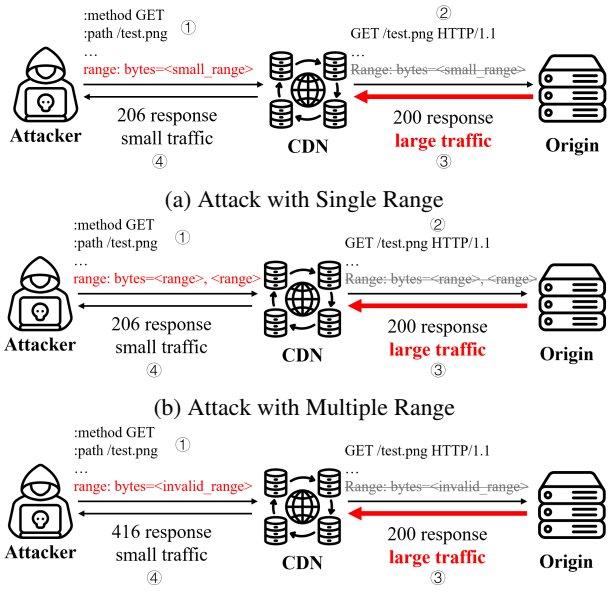
4.2.4 Comparison of Other Frameworks

This study does not directly compare with existing tools. In the black-box CDN environment, traditional coverage metrics are unavailable. Moreover, the number of discovered vulnerabilities depends heavily on the target implementation, making objective quantitative comparisons difficult. Here, we provide a brief qualitative comparison of H3Act against existing tools, including Reqsminer [63], HDiff [56], and Frameshifter [28].

We conduct an in-depth analysis of the theoretical design and implementation of these tools. We consider H3Act to possess the following advantages over existing approaches:

1. Input Flexibility: H3Act can fuzz directly from textual knowledge. In contrast, Reqsminer and HDiff require mutations to begin with HTTP semantics defined by ABNF rules, whereas Frameshifter requires mutations based on complete HTTP/2 request frame sequences.

2. Feedback Guidance: H3Act possesses feedback guidance during testing. Conversely, HDiff and Frameshifter lack feedback guidance for mutation, relying essentially on random generation. Although Reqsminer incorporates CDN response states for feedback, its guidance remains limited.



(a) Attack with Single Range

(b) Attack with Multiple Range

(c) Attack with Wrong Range

Figure 3: RangeAmp when CDNs remove `Range`.

5 Measurement and Findings

We summarize the HTTP/3-to-HTTP/1.1 conversion anomalies detected during the evaluation and identify 7 distinct attack vectors across 9 CDNs, as shown in Table 5. We also measure the occurrence of these conversion anomalies in HTTP/2-to-HTTP/1.1.

5.1 Denial-of-Service Bugs

5.1.1 RangeAmp (Range-Header-based Attack)

Our research reveals that most CDNs exhibit flaws in processing `Range` headers. Attackers can exploit these flaws by constructing syntactically valid yet malicious `Range` requests to launch HTTP amplification attacks against the origin.

To evaluate the behavior of CDNs, our testing covers three typical `Range` header types: 1. **Single Range**: Requests a single, small byte range. 2. **Multiple Range**: Requests multiple overlapping byte ranges. 3. **Invalid Range**: Requests a byte range with logical errors (e.g., `100-90`).

According to RFC specifications [12,16], the ideal handling logic for these requests should be: direct forwarding, forwarding after merging overlapping ranges, and rejecting at the edge node, respectively. However, practical testing indicates that CDNs in the wild do not fully adhere to these specifications. We categorize observed CDN behaviors as follows: 1. **Forwarding**: During translation, the CDN remains the `Range` header unchanged, regardless of its validity. 2. **Expansion**: The CDN extends the requested range during translation. 3. **Removal**: The CDN removes the `Range` header during translation. 4. **Ignoring**: The CDN completely ignores the `Range` header. 5. **Refusing**: The CDN refuses the request.

Among these behaviors, Removal poses a severe risk of HTTP amplification attacks. An attacker can request minimal data via HTTP/3 while forcing CDN to retrieve the entire resource from origins, potentially exhausting origin bandwidth and disrupting normal service for legitimate users, as shown in Figure 3. Expansion similarly introduces risks of amplification; however, its amplification factor is lower than that of Removal, as the CDN requests only finite resource fragments from the origin. Forwarding carries the risk of CPDoS when handling Invalid Range requests. This occurs because the origin will return a 416 error, which the CDN may cache, thereby denying access to the resource for normal clients.

We summarize the behaviors of each CDN in Table 6. The results show that nearly all tested CDNs exhibit at least one anomalous behavior. According to Li et al [34], Alibaba, Huawei, and CloudFront have fixed this vulnerability in HTTP/1.1, but it has regressed in HTTP/3. We will show the amplification factors of the attack in Appendix A.

5.1.2 CondAmp (Conditional-Request-based Attack)

In our research, we also observe flaws in the processing logic of CDNs regarding Conditional Requests. Attackers can exploit compliant conditional request headers to construct malicious traffic, thereby inducing HTTP amplification attacks against the origin.

Our testing covers five types of conditional request headers: `If-Match`, `If-None-Match`, `If-Modified-Since`,

Table 5: Attack Overview

Attack Vector	Alibaba CDN	Alibaba ESA	Baidu	Cloudflare	Cloudfront	Fastly	Huawei	Tencent CDN	Tencent EdgeOne
RangeAmp [34]	✓	✓	✓	✓	✓	✓	✓		
CondAmp [63]	✓	✓	✓	✓	✓	✓	✓		
HTTP Slow Attack [24]	✓	✓	✓		✓		✓	✓	✓
URI-Based CP [8]							✓	✓	
Multi-Host-Based CP [8]								✓	✓
Foo-CRLF Attack							✓		
Meta Characters [36]	✓	✓					✓	✓	✓

Table 6: Behaviors of CDN when dealing `Range` Headers.

	Single Range	Multiple Range	Wrong Range
Alibaba CDN	Expansion (512k)	Expansion[1] (512k)	Removal
Alibaba ESA	Expansion (512k)	Expansion[1] (512k)	Removal
Baidu	Other Behaviors[2]		
Cloudflare	Removal	Removal	Removal
Cloudfront	Expansion (1M)	Expansion[3] (1M)	Ignoring
Fastly	Removal	Ignoring	Ignoring
Huawei	Expansion (512k)	Ignoring	Expansion (512k)
Tencent EdgeOne	Expansion[4] (4k)	Expansion[4] (4k)	Refusing
Tencent CDN	Forwarding	Expansion[4] (4k)	Refusing

[1] Aliyun CDN and Aliyun ESA process only the first byte range specified, ignoring all subsequent ranges.
[2] Baidu CDN initially issues a request for the first 1MB of the resource. It subsequently forwards the request containing the original headers to the origin server. The response received by the client appears normal.
[3] Cloudfront merges multiple byte ranges into a single contiguous range then expands it.
[4] Tencent CDN decomposes a `Range` request containing multiple byte ranges into multiple separate requests, each specifying a single byte range. Furthermore, each requested range is expanded to 4KB before forwarding. Tencent EdgeOne behaves similarly but merges ranges in 4KB units. Consequently, a single back-to-origin request may satisfy multiple client `Range` requests.

`If-Unmodified-Since`, and `If-Range`. For the first four headers, we constructed test cases where the preset conditions are not met (i.e., edge nodes should theoretically return lightweight responses, such as 304 or 412 status codes). CDNs should either intercept these requests at the edge or transparently forward the original conditions to the origin for verification. For the `If-Range` header, we constructed standard requests with the expectation that the CDN would forward them without modifying the header values.

We categorize the CDN processing behaviors for conditional request headers as follows: 1. **Forwarding**: The CDN remains the conditional request header unchanged during translation. 2. **Removal**: The CDN removes the conditional request header during translation. 3. **Ignoring**: The CDN completely ignores the conditional request header. 4. **Expansion**: Specifically for `If-Range` requests, the CDN correctly remains the `If-Range` header but expands the requested `Range` within the associated `Range` header during translation. The implications of these behaviors are similar to those described in Section 5.1.1. When a CDN adopts the Removal behavior, the workflow by which an attacker exploits this mechanism to launch an amplification attack against the origin server is illustrated in Figure 4.

Table 7 presents our observations regarding CDN processing behaviors. All tested CDNs exhibit flaws in the processing logic for at least one type of conditional request. We will show the amplification factors of the attack in Appendix A.

5.1.3 HTTP Slow Attack

Because our architecture supports constructing HTTP/3 requests with active transmission pauses, we can observe the impact of HTTP/3 slow-rate attacks on CDNs. This study primarily identifies two categories of slow attack vectors:

1. **Slow Headers (Pre-Headers)**: Transmitting complete or partial HTTP/3 headers to CDN edge nodes and maintaining connection liveness without executing other operations . This aims to determine whether the CDN establishes and sustains a persistent connection with the origin.

2. **Slow Body**: Transmitting headers that either lack a `Content-Length` or contain an excessively big `Content-Length` to CDN edge nodes, followed by the transmission of the message body at an extremely low rate (e.g., one byte every 3 to 10 seconds). This serves to test whether the CDN continuously occupies connection resources with the origin.

Regarding request method coverage, our testing framework supports both `POST` and `GET`. Table 8 summarizes the defensive effectiveness of CDNs against these attacks. The data indicate that the vast majority of CDNs are susceptible to at least one of the aforementioned slow attack vectors and fail to prevent the exhaustion of connection resources effectively.

5.2 Cache Poisoning Bugs

We extend our research to multi-domain environments and observe that certain CDNs exhibit consistency issues during the parsing and conversion of HTTP/3 requests, thereby creating cache-poisoning vulnerabilities.

Table 7: Behaviors of CDN when dealing Conditional Request Headers.

	If-Match	If-None-Match	If-Modified-Since	If-Unmodified-Since	If-Range
Alibaba CDN	Forwarding	Removal	Removal	Forwarding	Forwarding
Alibaba ESA	Forwarding	Removal	Removal	Forwarding	Expansion (512k)
Baidu	Other Behavior[1]				
Cloudflare	Ignoring	Removal	Removal	Ignoring	Removal
Cloudfront	Forwarding	Removal	Removal	Forwarding	Expansion (1M)
Fastly	Forwarding	Removal	Removal	Forwarding	Removal
Huawei	Removal[2]	Removal[2]	Removal[2]	Removal[2]	Expansion (512k)
Tencent EdgeOne	Forwarding	Ignoring	Ignoring	Forwarding	Expansion (4k)
Tencent CDN	Forwarding	Ignoring	Ignoring	Forwarding	Forwarding

[1] Baidu CDN initially issues a request for the first 1MB of the resource. It subsequently forwards the request containing the original headers to the origin. The response received by the client appears normal.
[2] Huawei CDN requests only the first 512 KB of the resource in the origin request.

Table 8: Whether CDN is Susceptible to the Specific Attack.

	Pre Headers (POST)	Slow Body (POST)	Pre Headers (GET)	Slow Body (GET)
Alibaba CDN	YES	YES	NO	NO
Alibaba ESA	YES	YES	YES	YES
Baidu	YES	YES	YES	YES
Cloudflare	NO	NO	NO	NO
Cloudfront	YES	YES	NO	NO
Fastly	NO	NO	NO	NO
Huawei	NO	YES	NO	YES
Tencent EdgeOne	YES	YES	YES	YES
Tencent CDN	YES	YES	YES	YES

5.2.1 URI-Based

We observe flaws in the processing logic of absolute URIs within the `:path` of HTTP/3 messages across certain CDNs. Our tests demonstrate that when Huawei CDN and Tencent CDN generate back-to-origin HTTP/1.1 requests, if the domain information in the `:path` conflicts with that in the `:authority`, the system prioritizes the information within the `:path` field. Consequently, this value is used to generate the `Host` header of the back-to-origin HTTP/1.1 message. The attack is illustrated in Figure 5.

Specifically, an attacker can construct a malicious message specifying the victim domain (`victim.com`) in the `:authority` header, while using an absolute URI pointing to the attacker's server (e.g., `https://attacker.com/`) in the `:path` header. The CDN's protocol translation layer forwards the request to `attacker.com`. Since the cache key is typically generated based on `:authority`, the malicious content returned by the attacker's server is erroneously cached under the victim domain, thereby achieving cache poisoning. According to Chen et al [8], Tencent has fixed this vulnerability in HTTP/1.1, but it has regressed in HTTP/3.

5.2.2 Multi-Host-Based

Beyond URI parsing issues, we identified vulnerabilities concerning the handling of duplicate or conflicting `:authority` headers. This issue was exclusively detected in Tencent CDN and EdgeOne, with the attack scenario illustrated in Figure 6.

Our experiments reveal that these CDNs accept requests containing multiple `Host` headers, multiple `:authority` headers, or a combination of both. Crucially, these CDNs adopt a "Last-one-wins" strategy for routing decisions. Empirical data indicates that when a request contains conflicting headers (e.g., sending `:authority: test-tencentcdn` followed by `:authority: test-tencentcdn-2`), the translated HTTP/1.1 request utilizes the latter header (in this example, the `Host` field of the translated request is set to `test-tencentcdn-2`). If the CDN's caching mechanism uses the first header to generate the cache key while the routing logic relies on the last, this parsing discrepancy will directly facilitate cache poisoning attacks. According to Chen et al [8], Tencent has fixed this vulnerability in HTTP/1.1, but it has regressed in HTTP/3.

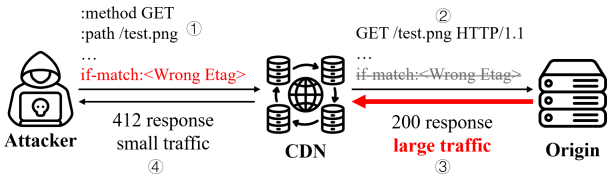
5.3 Request Smuggling Bugs

5.3.1 Foo-CRLF Attack

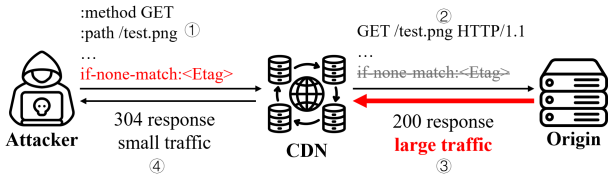
We identify a critical CRLF injection vulnerability within Huawei CDN, stemming from insufficient input validation during the HTTP/3-to-HTTP/1.1 conversion process.

Experiments reveal that Huawei CDN fails to filter CRLF sequences embedded within custom header values. As illustrated in Figure 7, an attacker can inject a payload containing `bar\r\n\r\n ...` into the value of a `Foo` header within an HTTP/3 request.

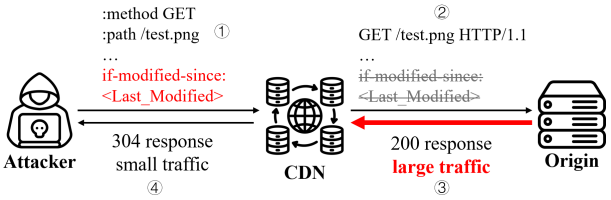
When the CDN converts the request to HTTP/1.1, it fails to strictly validate CRLF sequences, allowing them to persist in the translated request. According to the HTTP/1.1 specification, a double CRLF sequence signifies the end of the request [15]; thus, the injected CRLF sequence prematurely terminates the current request. Consequently, the origin in-



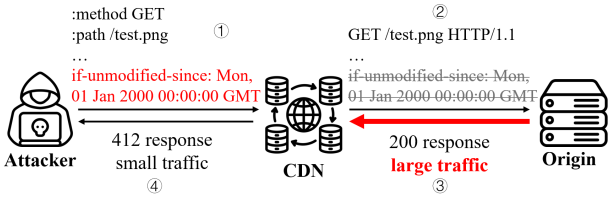
(a) Attack with `If-Match` header



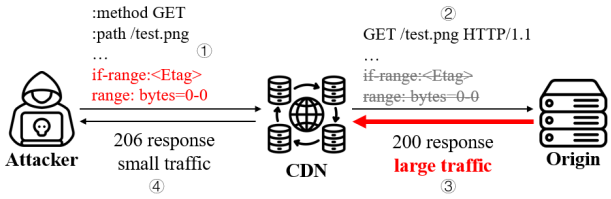
(b) Attack with `If-None-Match` header



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



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



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

Figure 4: CondAmp when CDNs remove conditional headers.

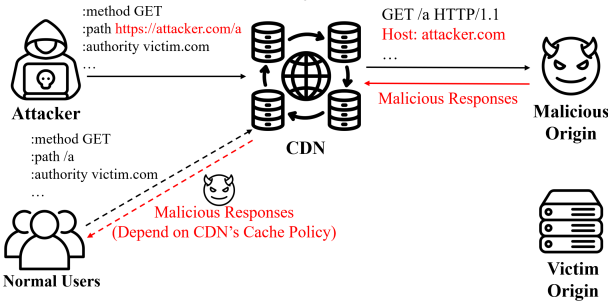


Figure 5: Workflow of URI-Based Cache Poisoning.

terprets the residual data as the beginning of a subsequent request. By exploiting this behavior, an attacker can inject

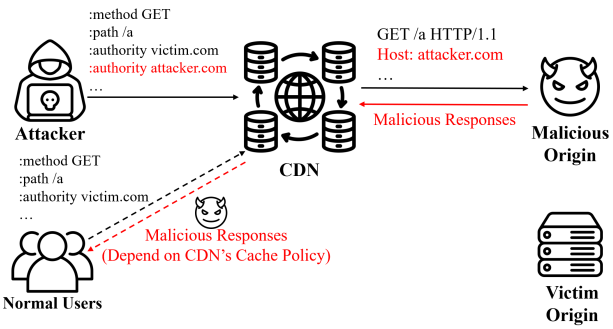


Figure 6: Workflow of Multiple-Host-Based Cache Poisoning.

a malicious request following the double CRLF sequence. Upon conversion, this payload is parsed by the origin as an independent request, thereby enabling HTTP request smuggling. Furthermore, by specifying `Host: attacker.com` in the smuggled request, the attacker can induce the CDN to cache a malicious response from the attacker's server under the victim's domain, thereby achieving cache poisoning against legitimate users.

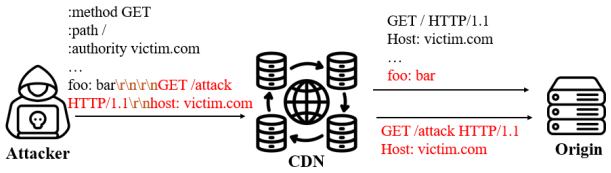
5.3.2 Meta Characters

We analyzed the meta character filtering mechanisms across various CDNs and discovered that certain providers exhibit overly lenient handling of non-ASCII and control characters (meta characters). This issue affects Alibaba CDN, Alibaba ESA, Huawei CDN, Tencent CDN, and Tencent EdgeOne. Specifically, these CDNs forward directly dangerous meta characters, such as `\f`, `\a` and `\b`, from HTTP/3 requests to the HTTP/1.1 origin without performing any encoding, sanitization, or rejection. Since HTTP parsers differ in their interpretation of control characters (e.g., some parsers may treat `\f` as line terminators or delimiters), this behavior may create a high risk of triggering request smuggling variants tailored to specific backend implementations (such as specific versions of HAProxy [11]).

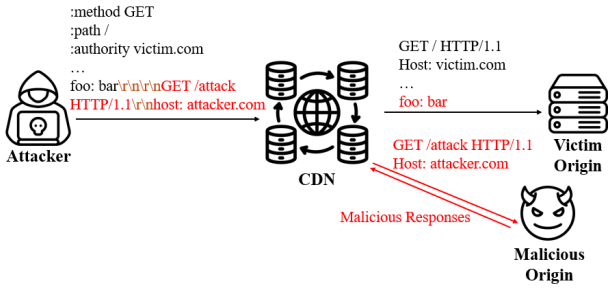
5.4 HTTP/2 vs HTTP/3

We manually test the HTTP/2 implementations of CDNs. The result is shown in Appendix B. Our results show that certain issues observed in HTTP/3 are mitigated or absent in HTTP/2, such as Meta Characters. Conversely, some issues appear more frequently in HTTP/2 than in HTTP/3, such as Multi-Host-Based CP. These findings suggest that the HTTP/2 and HTTP/3 protocol stacks in CDNs are implemented independently and orthogonally.

We also observe broader behavioral differences across CDNs when handling the same requests sent via HTTP/2 and HTTP/3. For instance, HTTP/3 stacks tend to reject requests containing extra spaces or uppercase letters in header fields, while HTTP/2 stacks are more tolerant, either filtering



(a) Foo-CRLF Request Smuggling Attack.



(b) Foo-CRLF Cache Poisoning Attack.

Figure 7: Workflow of Foo-CRLF Attack.

out extra spaces or converting uppercase letters to lowercase. HTTP/3 stacks often reject requests that include a host field (with correct syntax, simply replacing **:authority** with host), whereas HTTP/2 stacks typically accept such requests. Additionally, when rejecting malformed requests, some CDN HTTP/3 stacks terminate the QUIC connection directly without returning an HTTP response with an error status code. Even when error codes are returned, they might differ from those returned by the HTTP/2 stack.

Overall, these observations support the conclusion that HTTP/2 and HTTP/3 protocol stacks are implemented as independent and distinct components. This conclusion is further supported by open-source proxy software such as Nginx, which also implements its HTTP/2 and HTTP/3 stacks separately.

6 Discussion

In this section, we primarily discuss the mitigation solution, the limitations of H3Act and future directions.

6.1 Mitigation Solution

To address the identified issues, we propose the following mitigation recommendations:

1. **Validate Range Headers**: We recommend that CDNs first validate the effectiveness of `Range` header values. Invalid `Range` fields should immediately result in a 416 error. Furthermore, single-range requests should be forwarded directly. For multi-range requests, CDNs should merge overlapping segments to minimize the volume of data retrieved from the origin before forwarding [12, 16, 34].

2. **Validate Conditional Headers**: We recommend that CDNs prioritize validating conditional request headers. If the condition allows for a local response, it should be returned immediately without contacting the origin. If interaction with the origin is necessary, the conditional request fields should be preserved and forwarded to the origin.

3. **Disable Streaming Mode for Upstream Connections**: We advise against using "streaming mode" [24, 28], where the CDN initiates a connection to the origin before receiving the complete client request. We recommend that CDNs wait until the full request is received and buffered before establishing communication with the origin.

4. **Enforce Strict Absolute URI Checks**: We recommend that CDNs strengthen absolute URI validation. Specifically, if there is a discrepancy between the address specified in the absolute URI and the domain information in the header fields, the request should be immediately rejected [8, 15, 16, 19].

5. **Restrict Duplicate Headers**: We recommend enhancing checks for duplicate header fields. Requests containing duplicates of headers that should hold a unique value (such as `Host`) should be directly rejected [8, 15, 16, 19].

6. **Filter CRLF and Meta-characters**: We recommend that CDNs enforce stricter filtering of CRLF sequences and other meta-characters within header fields to prevent these anomalies from reaching the origin [15, 19, 44].

Although conversion between old and new protocols introduces new risks, we do not recommend that service providers withdraw from or avoid adopting new protocols such as HTTP/3. Nor do we advise them to aggressively phase out legacy protocols like HTTP/1.1. The former approach may stifle community innovation and hinder the development of the new protocol ecosystem, while the latter may harm the interests of certain users. We first urge vendors to address the security risks introduced by the protocol translation layer. We also believe that vendors can collaborate and build consensus with customers and the broader community. This collaboration can drive iterative protocol updates at the appropriate time and under suitable conditions, ensuring the orderly replacement of legacy protocols.

6.2 Limitations of Methodology

We acknowledge the following limitations in H3Act:

1. **High Operational Cost and Low Speed**: Although we have optimized token consumption to the greatest extent possible, the inference latency of the LLM-based generation process remains significantly higher than that of rule-based binary fuzzers. This inherent latency restricts the throughput required for large-scale scanning. Furthermore, the volume of packets generated per batch is notably lower than that of traditional fuzzers.

2. **Interference from Safety Alignment**: Commercial and open-source LLMs typically undergo safety alignment training to prevent the generation of malicious content. In some

cases, the model misinterprets the research context as a violation of safety policies. Consequently, it refuses to construct complex attack payloads (such as request smuggling vectors) or analyze the provided logs. Although rare, we have observed this phenomenon.

3. **LLM Hallucination**: LLMs inevitably produce hallucinations. In our false-positive rate evaluation, we observed that some false alarms originate from hallucinations within the analysis agent. Although the overall scale is manageable, this remains a significant limitation of the approach.

6.3 Future Directions

Future work can proceed in the following directions:

1. **Protocol Coverage Expansion**: Our current testing primarily focuses on HTTP/3-to-HTTP/1.1 conversion. Future work can extend protocol coverage in two dimensions: first, by investigating anomalies in HTTP/3-to-HTTP/2 conversion; and second, by performing a comparative security analysis of HTTP/2 and HTTP/3 implementations under identical cases.

2. **Scope Expansion**: We aim to extend our testing to include various ubiquitous open-source reverse proxies (e.g., Nginx, Apache). This will allow us to measure whether "traditional" security issues are resurfacing on a broader scale following the upgrade from HTTP/2 to HTTP/3.

3. **Model Capability Enhancement**: Constrained by cost and other factors, our current testing utilized only gpt-oss:120b as base models. However, our architecture is designed to be highly modular and open. We anticipate that integrating more powerful next-generation models, such as Gemini 3 Pro or ChatGPT-5.2, will enable the architecture to uncover a greater number of novel vulnerabilities.

7 Related Work

7.1 CDN Security

As CDNs become critical internet infrastructure, their security threats are receiving increasing attention. In their respective surveys, Ghaznavi et al. [21] and Al-Hamdani et al. [2] systematically reviewed the threats facing CDNs, emphasizing the defense challenges ranging from edge to origin.

In terms of forwarding and optimization mechanisms, researchers have uncovered significant weaknesses. Chen et al. [9] demonstrated the construction of "forwarding loop" attacks by exploiting forwarding configuration flaws. As research deepens, utilizing protocol inconsistencies for traffic amplification has become a new focus. RangeAmp proposed by Li et al. [34], CDN-Convert and BtOAmp discovered by Lin et al. [37, 38], exploiting back-to-origin policy flaws) all demonstrate that attackers can trigger high bandwidth consumption. To counter CDN defenses, Guo et al. [24] proposed several methods, including HTTP/2 bandwidth amplification, to use the CDN itself to break its own DoS protection. Meanwhile, REQSMINER, developed by Zheng et al. [63], systematically mines various forwarding inconsistency vulnerabilities through automated fuzzing.

Regarding the caching mechanism of CDNs, attackers have also developed methods to compromise availability and confidentiality. In terms of availability, Nguyen et al. [44] proposed Cache Poisoned Denial of Service (CPDoS), which exploits semantic discrepancies to force edge nodes to cache error pages, thereby blocking legitimate access. Regarding data privacy, Mirheidari et al. [42] systematically quantified the risk of sensitive information leakage caused by Web Cache Deception (WCD) attacks for the first time. Subsequently, by improving detection methods, they further revealed that WCD threats are widespread among the Alexa Top 10,000 websites and are more severe than previously estimated [43]. These studies indicate that the security boundary of CDNs faces comprehensive challenges, spanning from the underlying network infrastructure to upper-layer application logic.

7.2 HTTP Security

In the field of HTTP security, inconsistencies between protocol implementations and specifications are a significant cause of critical vulnerabilities. Empirical research by Chen et al. [8] revealed that ambiguities in Host header parsing within HTTP implementations trigger "Host of Troubles" attacks, leading to cache poisoning and policy bypass. Subsequently, James Kettle [31] revived the dormant "request smuggling" attack through groundbreaking research. By systematically exploiting parsing discrepancies between Content-Length and Transfer-Encoding headers, he proposed a reliable methodology for detection and exploitation. This approach not only bypassed security controls but also enabled large-scale cache poisoning and session hijacking.

To discover these complex parsing discrepancies more efficiently, researchers have proposed various automated detection frameworks. Jabiyev et al. [29] introduced T-Reqs, a grammar-based differential fuzzing tool. By systematically testing combinations of web servers and proxies, this tool uncovered multiple novel request smuggling vectors. To address the limitations of relying solely on direct testing, Shen et al. [56] proposed the HDiff framework. This approach creatively combined NLP analysis of RFC documents with differential testing, systematically identifying semantic gap attacks across mainstream HTTP implementations. To overcome black-box testing bottlenecks, Jabiyev et al. [27] proposed GUDIFU. Utilizing grey-box testing and comprehensive search strategies, it detects deep parsing discrepancies more effectively than existing methods.

Regarding attack impact and protocol evolution, threats stemming from parsing discrepancies persist despite technical advancements. Liang et al. [36] developed the HCache tool targeting Web Cache Poisoning (WCP), discovering seven new

attack vectors and over 1,000 vulnerable websites on the internet. With the adoption of HTTP/2, Kettle [32] revealed new risks introduced when front-end servers downgrade HTTP/2 requests to HTTP/1.1 for forwarding (e.g., H2.CL, H2.TE). Targeting this protocol translation layer, Jabiyev et al. [28] proposed the Frameshifter tool to identify various attacks caused by HTTP/2-to-HTTP/1 conversion anomalies. Even with the latest HTTP/3 protocol, Pisu et al. [48] demonstrated that header validation issues in proxies still lead to request smuggling risks if RFC specifications are not strictly followed.

7.3 LLM for Vulnerability Discovery

Recent research increasingly explores the application of Large Language Models (LLMs) in vulnerability discovery and network protocol security. Meng et al. [41] proposed CHATAFL, which integrates LLMs into mutation-based protocol fuzzing. By extracting message grammars, enriching seed corpora, and guiding state transitions, it significantly improved code coverage and vulnerability discovery for stateful protocols. Similarly, Wei et al. [62] introduced NeTestLLM, which automated network protocol testing by using LLMs to extract protocol specifications for test case generation and converting natural language descriptions into executable test scripts. In the specific domain of Web Application Firewalls (WAFs), Wang et al. [60] presented WAF Manis. This grammar-based testing approach automatically discovers protocol-level evasion vulnerabilities by generating payload-aware malformed HTTP requests that exploit parsing discrepancies. Liu et al. [40] proposed PROMEFUZZ. It guided LLMs by constructing a knowledge base containing code metadata, API documentation, and user usage patterns to generate syntactically and semantically correct fuzzing harnesses. Furthermore, LLMs are used to extract formal specifications for verification. Sharma [55] introduced PROSPER, which utilizes LLMs to extract protocol Finite State Machines (FSMs) from RFC documents. It achieved this by combining the analysis of natural language text with visual artifacts such as state diagrams. Finally, Zheng et al. [64] proposed PARVAL, which employed a multi-agent LLM approach to extract and compare format specifications from both source code and RFCs, thereby detecting semantic inconsistencies in protocol parser implementations.

8 Conclusion

In this paper, we present H3Act, through which we evaluated security during HTTP/3-to-HTTP/1.1 protocol conversion in CDN environments. Our research reveals a concerning reality: despite the significant advancements inherent in the HTTP/3 protocol, the translation layer introduced to ensure backward compatibility has emerged as a novel security weakness. Our framework successfully identified 7 attack vectors across 9 CDNs, including DoS, cache poisoning, and request smuggling. The ubiquity of these vulnerabilities indicates that during the implementation of next-generation network protocol stacks, the industry may overlook the complexity of semantic translation between different protocols. Consequently, security risks that were previously mitigated in the HTTP/1.1 and HTTP/2 eras have regressed in the HTTP/3. Our work serves not only as an efficient automated testing tool but also underscores a critical imperative: as internet infrastructure evolves, the crucial process of "protocol translation" demands more rigorous security auditing and compliance verification. We hope this study will galvanize the community to prioritize the security of CDN protocol translation and drive the adoption of more secure HTTP/3 deployment practices.

Acknowledgments

This work is supported by the Deng Feng Fund of the Beijing National Research Center for Information Science and Technology. Authors from Nankai University (Prof. Xiang Li) were supported by the National Natural Science Foundation of China (No. 62502236, No. U25B2025), the Natural Science Foundation of Tianjin (No. 24JCQNJC02070), and the Open Project of National Engineering Laboratory for Technology of Internet Domain Name (No. KF202516). Prof. Yiming Zhang is in part supported by the NSFC #62302258.

Ethical Considerations

As this study involves security testing against real-world commercial CDN infrastructure, we strictly adhered to responsible research guidelines throughout every stage of the experimental design. Our objective was to ensure no disruption to the normal operation of public services and no infringement upon third-party rights.

1. **Controlled Experimental Environment**: All test traffic was directed exclusively towards origin and domains registered and controlled by the authors. We never attempted to access, modify, or interfere with any third-party origin data or cached content. Our threat model explicitly restricts the attacker's capabilities to possessing a controlled origin, thereby ensuring clear and safe testing boundaries.

2. **Minimized Impact**: During the verification of DoS-related vulnerabilities (e.g., amplification and slow-rate attacks), we strictly limited the traffic rate and the number of concurrent connections (e.g., pauses for 0.5 seconds after each send). We transmitted only the minimum sample size necessary to verify the vulnerability's existence and ceased transmission immediately upon confirming the anomalous behavior. This ensured that no denial-of-service or resource-exhaustion attacks occurred at the CDN edge nodes.

3. **Data Privacy Protection**: This study involves interactions strictly at the protocol level and does not involve any

personal user data. All logs generated during the experiments contain only constructed test data and are stored within a controlled local environment.

For the vulnerability disclosure process. Overall, we reported our findings to 7 major CDN vendors and have received active responses and engaged in in-depth interactions with 6 of them:

Tencent: The Tencent security team confirmed the vulnerabilities regarding the handling of anomalous paths and multiple Host/Authority headers (which could lead to cache poisoning). As of now, these issues have been successfully patched. Other reported vectors will be addressed as optimizations in the future.

Baidu: Upon internal review, Baidu explicitly confirmed the potential real-world harm of the CondAmp and has patched this vulnerability.

Alibaba: The Alibaba security team conducted a detailed internal evaluation of our four reported attack vectors. They pointed out that the traffic amplification effects of Range and Conditional-Request behaviors primarily manifest at the origin server, and will not cause the CDN itself to become unavailable. Therefore, they consider this a Shared Responsibility model, suggesting that appropriate defense and rate-limiting mechanisms should be deployed and configured by the customer at the origin side, rather than being directly blocked by the CDN edge nodes. We reviewed Alibaba's relevant documentation, which describes a back-to-origin chunk size between 512KB and 4MB, but notably lacks clear warnings to users regarding the potential security risks of bandwidth amplification.

Cloudflare: After weeks of technical interaction, Cloudflare fully understood our constructed attack patterns. They categorized this as expected "cache-fill behavior" and stated that this architectural design is not unique to Cloudflare.

CloudFront: The CloudFront team cited their official Developer Guide, pointing out that proactively requesting more bytes than specified by the client is an expected system optimization behavior designed to improve performance and cache efficiency.

Fastly: Our vulnerability report has been successfully submitted and is currently pending initial review by their security team.

Huawei: We have communicated with Huawei for several weeks. Despite we have reported multiple times, the vendor still fails to understand the reported issues.

The primary stakeholders in this work include CDN providers, CDN customers (origin operators), end users, and the broader research and security community. Different stakeholders may experience both benefits and burdens from disclosure.

CDN providers may incur operational, remediation, and reputational costs associated with investigating and addressing translation-layer inconsistencies. At the same time, the findings may help providers identify regression risks during HTTP/3 deployment, improve protocol-translation validation practices, and better define security boundaries between CDN infrastructure and customer-controlled origins. Furthermore, during the disclosure process, we make every effort to report vulnerabilities to all tested providers and minimize negative impact.

CDN customers may face risks from amplification behaviors, cache inconsistencies, or request-confusion effects under certain deployment conditions. However, disclosure also enables customers to make more informed deployment decisions, evaluate whether and how to enable HTTP/3, and strengthen origin-side mitigations and cache-control policies.

End users could indirectly experience degraded service availability if such behaviors were abused in practice. Conversely, improved awareness and remediation of protocol-translation weaknesses may contribute to a more secure and stable ecosystem over time.

The research and security community benefits from a better understanding of HTTP/3-to-HTTP/1.1 translation risks and regression patterns in independently implemented protocol stacks. At the same time, public discussion of these behaviors may increase awareness of attack techniques among adversaries. We therefore intentionally omit exploit automation details, operational attack tooling, and deployment-specific offensive guidance from the paper and released artifacts.

We carefully considered the potential risks of public disclosure before publication. Some affected vendors acknowledged parts of the reported behaviors and initiated remediation efforts, while others considered the observed behaviors to be expected operational trade-offs or shared-responsibility issues. Consequently, not all identified risks are fully mitigated at the time of writing. However, we nevertheless believe publication is justified for several reasons. First, the reported attack vectors are derived from publicly documented HTTP semantics and previously studied classes of parser-confusion and translation-layer attacks, rather than from undisclosed implementation secrets. Second, exploiting these behaviors in practice still requires nontrivial environmental conditions, including compatible CDN configurations, origin behaviors, and controlled request construction. Third, public discussion of these findings enables operators and customers to better evaluate the risks introduced by heterogeneous protocol translation, particularly during HTTP/3 adoption.

Open Science

We have made the entire codebase used in this research open source. The link is https://github.com/AsadaShino666/H3Act or https://zenodo.org/records/20398181.

The repository contains two parts. The first part provides the full source code for the H3Act tool, including all prompts and all modules. Researchers wishing to reproduce the tool can copy these directly to a local environment for execution. However, please note that our code currently supports only a

local Ollama environment and does not include API adaptation; details are provided in the repository's README. The second part comprises simple HTTP/3 request transmission logic and test cases corresponding to the vulnerabilities described in Section 5. Users who merely wish to scan CDNs for these vulnerabilities can use this simple logic to send packets and manually verify if the messages received by the origin match our descriptions. This is also detailed in the README.

References

[1] 6sense, 2026. https://6sense.com/tech/content-delivery-network-cdn.

[2] Saja Al-Hamdani and Dujan B. Taha. Security in content delivery networks (cdns): A literature review. In *2025 International Conference on Computer Science and Software Engineering (CSASE)*, pages 132–139, 2025.

[3] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, May 2015.

[4] David Belson. The 2025 Cloudflare Radar Year in Review: The rise of AI, post-quantum, and record-breaking DDoS attacks, 2025. https://blog.cloudflare.com/radar-2025-year-in-review/.

[5] David Benjamin. Using TLS 1.3 with HTTP/2. RFC 8740, February 2020.

[6] Mike Bishop. HTTP/3. RFC 9114, June 2022.

[7] Efstratios Chatzoglou, Vasileios Kouliaridis, Georgios Kambourakis, Georgios Karopoulos, and Stefanos Gritzalis. A hands-on gaze on HTTP/3 security through the lens of HTTP/2 and a public dataset. *Comput. Secur.*, 125:103051, 2023.

[8] Jianjun Chen, Jian Jiang, Haixin Duan, Nicholas Weaver, Tao Wan, and Vern Paxson. Host of troubles: Multiple host ambiguities in http implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1516–1527, New York, NY, USA, 2016. Association for Computing Machinery.

[9] Jianjun Chen, Jian Jiang, Xiaofeng Zheng, Haixin Duan, Jinjin Liang, Kang Li, Tao Wan, and Vern Paxson. Forwarding Loop Attacks in Content Delivery Networks. In *Proceedings 2016 Network and Distributed System Security Symposium*, 2016.

[10] Cloudflare. What is cdn?, 2026. https://www.cloudflare.com/learning/cdn/what-is-a-cdn/.

[11] CVE. CVE-2019-18277, 2019. https://www.cve.org/CVERecord?id=CVE-2019-16782.

[12] Roy T. Fielding, Yves Lafon, and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Range Requests. RFC 7233, June 2014.

[13] Roy T. Fielding, Mark Nottingham, and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Caching. RFC 7234, June 2014.

[14] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Caching. RFC 9111, June 2022.

[15] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP/1.1. RFC 9112, June 2022.

[16] Roy T. Fielding, Mark Nottingham, and Julian F. Reschke. HTTP semantics. *RFC*, 9110:1–194, 2022.

[17] Roy T. Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Authentication. RFC 7235, June 2014.

[18] Roy T. Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. RFC 7232, June 2014.

[19] Roy T. Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, June 2014.

[20] Roy T. Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, June 2014.

[21] Milad Ghaznavi, Elaheh Jalalpour, Mohammad A. Salahuddin, Raouf Boutaba, Daniel Migault, and Stere Preda. Content delivery network security: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2166–2190, 2021.

[22] Run Guo, Jianjun Chen, Baojun Liu, Jia Zhang, Chao Zhang, Haixin Duan, Tao Wan, Jian Jiang, Shuang Hao, and Yaoqi Jia. Abusing cdns for fun and profit: Security issues in cdns' origin validation. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 1–10, 2018.

[23] Run Guo, Jianjun Chen, Yihang Wang, Keran Mu, Baojun Liu, Xiang Li, Chao Zhang, Haixin Duan, and Jianping Wu. Temporal CDN-Convex Lens: A CDN-Assisted Practical Pulsing DDoS Attack. In *32th USENIX Conference on Security Symposium*, 2023.

[24] Run Guo, Weizhong Li, Baojun Liu, Shuang Hao, Jia Zhang, Haixin Duan, Kaiwen Sheng, Jianjun Chen, and Ying 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.

[25] Shuai Hao, Yubao Zhang, Haining Wang, and Angelos Stavrou. End-Users get maneuvered: Empirical analysis of redirection hijacking in content delivery networks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1129–1145, Baltimore, MD, August 2018. USENIX Association.

[26] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[27] Bahruz Jabiyev, Anthony Gavazzi, Kaan Onarlioglu, and Engin Kirda. Gudifu: Guided differential fuzzing for http request parsing discrepancies. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '24, page 235–247, New York, NY, USA, 2024. Association for Computing Machinery.

[28] Bahruz Jabiyev, Steven Sprecher, Anthony Gavazzi, Tommaso Innocenti, Kaan Onarlioglu, and Engin Kirda. FRAMESHIFTER: Security Implications of HTTP/2-to-HTTP/1 Conversion Anomalies. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 1061–1075. USENIX Association, 2022.

[29] Bahruz Jabiyev, Steven Sprecher, Kaan Onarlioglu, and Engin Kirda. T-reqs: Http request smuggling with differential fuzzing. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1805–1820, New York, NY, USA, 2021. Association for Computing Machinery.

[30] Ben Kallus, Prashant Anantharaman, Michael E. Locasto, and Sean W. Smith. The HTTP garden: Discovering parsing vulnerabilities in HTTP/1.1 implementations by differential fuzzing of request streams. *CoRR*, abs/2405.17737, 2024.

[31] James Kettle. HTTP Desync Attacks: Request Smuggling Reborn, 2019. https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn.

[32] James Kettle. HTTP/2: The Sequel is Always Worse, 2021. https://portswigger.net/research/http2.

[33] James Kettle. HTTP/1.1 must die: the desync endgame, 2025. https://portswigger.net/research/http1-must-die.

[34] Weizhong Li, Kaiwen Shen, Run Guo, Baojun Liu, Jia Zhang, Haixin Duan, Shuang Hao, Xiarun Chen, and Yao Wang. Cdn backfired: Amplification attacks based on http range requests. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 14–25, 2020.

[35] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, Tao Wan, and Jianping Wu. When https meets cdn: A case of authentication in delegated service. In *2014 IEEE Symposium on Security and Privacy*, pages 67–82, 2014.

[36] Yuejia Liang, Jianjun Chen, Run Guo, Kaiwen Shen, Hui Jiang, Man Hou, Yue Yu, and Haixin Duan. Internet's invisible enemy: Detecting and measuring web cache poisoning in the wild. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 452–466, New York, NY, USA, 2024. Association for Computing Machinery.

[37] Ziyu Lin, Zhiwei Lin, Ximeng Liu, Jianjun Chen, Run Guo, Cheng Chen, and Shaodong Xiao. CDN cannon: Exploiting CDN Back-to-Origin strategies for amplification attacks. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5717–5734, Philadelphia, PA, August 2024. USENIX Association.

[38] Ziyu Lin, Zhiwei Lin, Ximeng Liu, Zuobing Ying, and Cheng Chen. Unveiling the bandwidth nightmare: Cdn compression format conversion attacks. In *2024 2nd International Conference on Big Data and Privacy Computing (BDPC)*, pages 97–106, 2024.

[39] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

[40] Yuwei Liu, Junquan Deng, Xiangkun Jia, Yanhao Wang, Minghua Wang, Lin Huang, Tao Wei, and Purui Su. Promefuzz: A knowledge-driven approach to fuzzing harness generation with large language models. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, page 1559–1573, New York, NY, USA, 2025. Association for Computing Machinery.

[41] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. Large language model guided protocol fuzzing. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, 2024.

[42] Seyed Ali Mirheidari, Sajjad Arshad, Kaan Onarlioglu, Bruno Crispo, Engin Kirda, and William Robertson. Cached and confused: Web cache deception in the wild.

In *29th USENIX Security Symposium (USENIX Security 20)*, pages 665–682. USENIX Association, August 2020.

[43] Seyed Ali Mirheidari, Matteo Golinelli, Kaan Onarlioglu, Engin Kirda, and Bruno Crispo. Web cache deception escalates! In *31st USENIX Security Symposium (USENIX Security 22)*, pages 179–196, Boston, MA, August 2022. USENIX Association.

[44] Hoai Viet Nguyen, Luigi Lo Iacono, and Hannes 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 '19, page 1915–1936, New York, NY, USA, 2019. Association for Computing Machinery.

[45] OpenAI. gpt-oss, 2025. https://ollama.com/library/gpt-oss.

[46] OWASP. Cache Poisoning, 2019. https://owasp.org/www-community/attacks/Cache_Poisoning.

[47] OWASP. Denial of Service, 2019. https://owasp.org/www-community/attacks/Denial_of_Service.

[48] Lorenzo Pisu, Federico Loi, Davide Maiorca, and Giorgio Giacinto. HTTP/3 will not Save you from Request Smuggling: A Methodology to Detect HTTP/3 Header (mis)Validations . In *2024 22nd International Symposium on Network Computing and Applications (NCA)*, pages 97–104, Los Alamitos, CA, USA, October 2024. IEEE Computer Society.

[49] PSF. Python 3.12.7, 2024. https://www.python.org/downloads/release/python-3127/.

[50] Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear transformer. *arXiv preprint arXiv:2210.10340*, 2022.

[51] Julian Reschke. HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields. RFC 7615, September 2015.

[52] Julian Reschke. Hypertext Transfer Protocol (HTTP) Client-Initiated Content-Encoding. RFC 7694, November 2015.

[53] Julian Reschke. The Hypertext Transfer Protocol Status Code 308 (Permanent Redirect). RFC 7538, April 2015.

[54] Eric Rescorla. HTTP Over TLS. RFC 2818, May 2000.

[55] Prakhar Sharma and Vinod Yegneswaran. Prosper: Extracting protocol specifications using large language models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, page 41–47, New York, NY, USA, 2023. Association for Computing Machinery.

[56] Kaiwen Shen, Jianyu Lu, Yaru Yang, Jianjun Chen, Mingming Zhang, Haixin Duan, Jia Zhang, and Xiaofeng Zheng. Hdiff: A semi-automatic framework for discovering semantic gap attack in http implementations. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–13, 2022.

[57] Martin Thomson and Cory Benfield. HTTP/2. RFC 9113, June 2022.

[58] Yuan Tian and Tianyi Zhang. Selective prompt anchoring for code generation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[59] w3tech. Usage statistics of HTTP/3 for websites, 2025. https://w3techs.com/technologies/details/ce-http3.

[60] Qi Wang, Jianjun Chen, Zheyu Jiang, Run Guo, Ximeng Liu, Chao Zhang, and Haixin Duan. Break the wall from bottom: Automated discovery of protocol-level evasion vulnerabilities in web application firewalls. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 185–202, 2024.

[61] Watchfire. HTTP Request Smuggling, 2005. https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf.

[62] Yunze Wei, Kaiwen Chi, Shibo Du, Xiaohui Xie, Ziyu Geng, Yuwei Han, Zhen Li, Zhanyou Li, and Yong Cui. Large language model driven automated network protocol testing. In *Proceedings of the 2025 Applied Networking Research Workshop*, ANRW '25, page 32–38, New York, NY, USA, 2025. Association for Computing Machinery.

[63] Linkai Zheng, Xiang Li, Chuhan Wang, Run Guo, Haixin Duan, Jianjun Chen, Chao Zhang, and Kaiwen Shen. Reqsminer: Automated discovery of CDN forwarding request inconsistencies and dos attacks with grammar-based fuzzing. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024.

[64] Mingwei Zheng, Danning Xie, and Xiangyu Zhang. Large language models for validating network protocol parsers. In *2025 IEEE Security and Privacy Workshops (SPW)*, pages 56–64. IEEE, 2025.

Table 9: The Amplification of RangeAmp on Every CDN.

Test Type	Metric	Aliyun CDN	Aliyun ESA	Baidu	Cloudflare	Cloudfront	Fastly	Huawei	Tencent CDN	Tencent EdgeOne
Single_Range / If-Range	Client Response Code	206	206	206	206	206	206	206	206	206
	Client Response Data (B)	1	1	1	1	1	1	1	1	1
	Origin Response Data (B)	524288	400717.6*	542713*	29552200	1048576	2513443	524288	1	4097
Multiple_Range	Client Response Code	206	206	206	206	206	200	200	206	206
	Client Response Data (B)	1	1	1505	1473	1689	2513443	29552200	1681	1681
	Origin Response Data (B)	524288	468256.2*	609378*	29552200	1048576	2513443	29552200	28673	12289
Wrong_Range	Client Response Code	416	416	416	416	200	200	416	416	416
	Client Response Data (B)	0	0	206	0	29552200	2513443	194	0	0
	Origin Response Data (B)	327368*	417596*	373503*	29552200	29552200	2513443	524288	0	0
If-Match / If-Unmodified-Since	Client Response Code	412	412	412	200	412	412	412	412	412
	Client Response Data (B)	182	182	182	29552200	182	182	170	182	182
	Origin Response Data (B)	182	182	490767*	29552200	182	182	524288	182	182
If-None-Match / If-Modified-Since	Client Response Code	304	304	304	304	304	304	304	200	200
	Client Response Data (B)	0	0	0	0	0	0	0	29552200	29552200
	Origin Response Data (B)	29552200	29552200	633937*	29552200	179957.6*	2513443	524288	29552200	29552200

Table 10: Overview of attack vectors which present in HTTP/2 and HTTP/3

Attack Vector	HTTP Version	Alibaba CDN	Alibaba ESA	Baidu	Cloudflare	Cloudfront	Fastly	Huawei	Tencent CDN	Tencent EdgeOne
RangeAmp	HTTP/3	✓	✓	✓	✓	✓	✓	✓		
	HTTP/2			✓	✓	✓	✓	✓		
CondAmp	HTTP/3	✓	✓	✓	✓	✓	✓	✓		
	HTTP/2	✓	✓	✓	✓	✓	✓			
HTTP Slow Attack	HTTP/3	✓	✓	✓		✓		✓	✓	✓
	HTTP/2	✓						✓	✓	
URI-based CP	HTTP/3							✓	✓	
	HTTP/2								✓	
Multi-Host-Based CP	HTTP/3								✓	✓
	HTTP/2	✓			✓	✓		✓	✓	✓
Foo-CRLF Attack	HTTP/3							✓		
	HTTP/2									
Meta Characters	HTTP/3	✓	✓					✓	✓	✓
	HTTP/2									

A Amplification Statistics

We measured the actual impact, i.e. the approximate bandwidth amplification factor, of the two attacks reported in Sections 5.1.1 and 5.1.2. To capture representative behavior, we repeated each packet transmission five times for every CDN. We then calculated the average length of the DATA frames received by the client and the average volume of data sent by the origin server. Using the experimental setup described in Section 4.1, we requested a 29,552,200-byte PNG image from the origin server. The final statistics are shown in Table 9. We observed that the response behaviors of both the CDNs and the origin server were generally stable. However, some CDNs prematurely closed the connection to the origin, preventing full data transmission. Consequently, the amount of data sent varied across attempts. We marked these unstable instances with an asterisk (*) in the table.

Specifically, the Range-related fields are configured as follows: **Single_Range** uses `range: bytes=0-0`; **Multiple_Range** uses `range: bytes=0-0, 1-1, 7-15, 428-1035, 1050-1258, 4095-4096, 524288-524290`; and **Wrong_Range** uses `range: bytes=100-90`. The settings for the five conditional fields are shown in Figure 4.

We observe that CDN behavior exhibits high similarity when handling **Single_Range** and requests with **If-Range**. Similar patterns also appear for **If-Match** versus **If-Unmodified-Since**, and for **If-None-Match** versus **If-Modified-Since**. Therefore, we merge these into three groups in our results.

B HTTP/2 vs HTTP/3

We present the occurrence of conversion anomalies in HTTP/3-to-HTTP/1.1 and HTTP/2-to-HTTP/1.1 across among CDNs in Table 10.