Internet's Invisible Enemy: Detecting and Measuring Web Cache Poisoning in the Wild

Yuejia Liang
Tsinghua University
Beijing, China
liangyj21@tsinghua.org.cn

Jianjun Chen*
Tsinghua University; Zhongguancun Laboratory
Beijing, China
jianjun@tsinghua.edu.cn

Run Guo
Tsinghua University
Beijing, China
gr15@tsinghua.org.cn

Kaiwen Shen
Tsinghua University; Clouditera Inc
Beijing, China
kaiwenshen17@gmail.com

Hui Jiang
Tsinghua University; Baidu Inc
Beijing, China
jianghui01@baidu.com

Man Hou
Zhongguancun Laboratory
Beijing, China
houman@zgclab.edu.cn

Yue Yu
Beijing University of Posts and Telecommunications
Beijing, China
yuyue_999@bupt.edu.cn

Haixin Duan
Tsinghua University; Quancheng Laboratory
Beijing, China
duanhx@tsinghua.edu.cn

ABSTRACT

Web cache poisoning (WCP) has posed significant threats to Internet security by causing the cache server to deliver malicious responses to innocent users. This results in widespread denial of access to website resources and potential injection of harmful payloads. However, prior works on WCP vulnerability have been fragmented and conducted in a case-by-case form, lacking a systematic analysis of the threat landscape. In this paper, we fill this research gap by conducting a systematic evaluation of WCP vulnerabilities at scale. We propose *HCache*, a novel testing methodology to facilitates the widespread identification of WCP vulnerabilities. We evaluated our methodology against Tranco Top 1000 domains and their subdomains, and found that over 1,000 websites across 172 domains, representing 17% of the evaluated domains, are vulnerable to WCP. In particular, we have identified 7 new attack vectors stemming from previously unexplored caching headers. We have responsibly disclosed the vulnerabilities to the affected websites and received acknowledgements and bug bounties from world-famous companies, such as Alibaba, Adobe, Huawei, and Microsoft.

CCS CONCEPTS

• **Networks** → **Network measurement**; • **Security and privacy** → *Network security*; *Web application security*.

KEYWORDS

Network Security, Measurement, Web Cache, Web Cache Poisoning

*Corresponding author.

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690361

ACM Reference Format:
Yuejia Liang, Jianjun Chen, Run Guo, Kaiwen Shen, Hui Jiang, Man Hou, Yue Yu, and Haixin Duan. 2024. Internet's Invisible Enemy: Detecting and Measuring Web Cache Poisoning in the Wild. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3690361

1 INTRODUCTION

To prevent unnecessary Internet traffic and enhance data transmission efficiency, web caching facilities are extensively used. They store frequently requested data resources, reducing the need for repeated data transfers. Given web cache's advantages, it has become a critical infrastructure component of the Internet. However, when compromised by malicious actors, web caching facilities pose significant risks to the Internet. Research indicates that issues with web caching can lead various security consequences, such as Denial-of-Service (DoS), Cross-site scripting (XSS), and information leakage [4, 16, 17, 19, 28].

Attacks against web cache typically fall into two categories, the *web cache deception (WCD)* and the *web cache poisoning (WCP)* [24, 25]. WCD aim to deceive the cache into making confidential information publicly available online, whereas WCP involve poisoning the cache with harmful payloads that are then distributed to unsuspecting users. In recent years, Mirheidar et al. [24, 25] studied the severity of WCD by measuring Alexa Top websites, demonstrating the widespread threats on the Internet. However, due to the complexity, WCP have been studied in a case-by-case form [4, 16, 17, 19, 24, 28], focusing on revealing the specific vulnerabilities while lacking a global Internet view of the severity. As the WCP poses a severe threat to the Internet, it is urgent to detect and prevent the vulnerabilities ahead of the attacker on the global scale.

In this paper, we aim to fill this gap by performing a systematic detection of WCP vulnerabilities at scale. To achieve this goal,

we need to address three research questions: (1) How can we generate testing requests that systematically probe cache poisoning vulnerabilities? (2) How can we accurately detect potential web cache poisoning? (3) How can we assess the impact of web cache poisoning while minimizing disruption to normal users?

To address these questions, we introduce a novel testing methodology, *HCache*, designed to detect WCP vulnerabilities. For the first question, we employ a *cache-key-aware* approach that systematically generates and mutates requests to identify fields not included in cache keys, thereby exposing potential inconsistencies. For the second question, we utilize a three-step detection strategy involving the issuance of a normal request, an attack request, and a validation request. This strategy allows us to analyze differences in the response's status code, content, and length to detect potential WCP vulnerabilities. For the third question, we incorporate *cache buster* variables in our request parameters, ensuring that our testing does not disrupt normal website operations while maintaining the efficacy of our detection approach.

We evaluated *HCache* against Tranco Top 1,000 domains involving 22,114 subdomains with 51,596 distinct URL links. Our evaluation discovers more than 1,000 websites across 172 domains, constituting 17% of the domains evaluated, are vulnerable to WCP. Moreover, we identify 7 new attack variants to trigger WCP, including HTTP protocol headers, scope requests, conditional requests, and so on. Meanwhile, we investigated the caching differences between HTTP/2 and HTTP/1.1 and found that the WCP problem is also prevalent in HTTP/2. Therefore, WCP is still a serious problem, and network operators and caching service providers should take appropriate measures to solve this problem. To the best of our knowledge, this study represents the first systematic, large-scale evaluation of WCP within a scientific framework. We reported the vulnerabilities to the affected websites and received acknowledgements from over 15 companies, including globally renowned ones like Adobe, Alibaba, Huawei, and Microsoft. Additionally, we received bug bounties totalling over $1,000 from these entities.

In summary, we make the following contributions:

- We introduced a novel testing methodology for large-scale evaluation of websites for WCP on the Internet, along with a practical detection system named *HCache*[1].
- We carried out a comprehensive analysis of the Tranco Top 1,000 domains and their subdomains, discovering over 1,000 websites across 172 domains vulnerable to WCP, indicating that 17% of measured domains are at risk.
- We discovered 7 new attack vectors that can cause WCP attacks and found the WCP issues are still prevalent in HTTP/2. We have responsibly reported the vulnerabilities to the affected websites and received acknowledgements and over $1,000 bug bounties from many companies such as Adobe, Alibaba, Huawei, and Microsoft.

2 BACKGROUND

2.1 Web Cache

Web cache reduces network traffic and optimizes application performance by caching frequently used network resources. It can be categorized into private caching and shared proxy caching. Private caches are caching mechanisms within the web client itself (e.g., the browser cache[30]) and within the web server (e.g., the WordPress plugin cache[36]). Shared proxy caching mainly includes various proxy servers and CDNs.

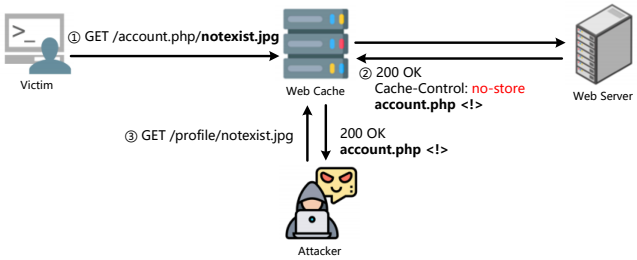

Figure 1: Process of web cache deception

The reports released by the three major CDN providers, Akamai, Cloudflare, and Fastly, indicate that a significant amount of network traffic passes through caching proxy communication each year [34]. A measurement study by Guo et al. [6] shows that among the top 1,000 domains in the Alexa ranking list, 74% of websites utilize CDN services for content distribution and network acceleration. Additionally, there are many independent caching proxies (such as Squid [32], Varnish [33]) and caching servers (such as Apache [10], Nginx [26]) distributed throughout the Internet, indicating that web caching devices have become critical infrastructure for the Internet.

Cache servers typically store static and commonly accessed resources like HTML, JS, CSS, images, and other media. Most web caches, due to their shared nature, do not cache dynamic, personalized, or sensitive content. The HTTP/1.1 specification's "Cache-Control" header directs caching devices on handling responses, such as "Cache-Control: no-store" to prevent storage. Despite RFC mandates for adherence to these headers, some caching devices and CDNs offer options to bypass them. A prevalent caching strategy involves rules based on resource paths and extensions, like caching only JPG, ICO, CSS, or JS files.

2.2 Web Cache Attack

As an important infrastructure in the Internet, web cache requires utmost security. There are primarily two attack vectors targeting cache servers based on their caching characteristics [25].

Web Cache Deception (WCD) is an attack that tricks the application into storing sensitive content belonging to other users in the cache. Subsequently, the attacker retrieves this content from the cache. Figure 1 shows the process of WCD: 1)The attacker tricks the victim into visiting a URL that requests */account.php/nonexist.jpg*. 2)The request reaches the web server and ignore the non-existent part of the URL. Web server send back a successful response with account.php, which has victim's private account. The web cache store the response, interpreting it as a static image. 3)The attacker visits the same URL accessing the victim's information stored in the cache.

Web Cache Poisoning (WCP) is to induce the application to store malicious content in the cache. The normal requests from

[1]https://github.com/phantomnothingness/HCache

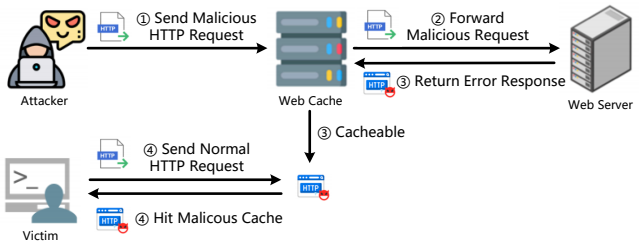

Figure 2: Process of web cache poisoning

other users may hit the cache, resulting in access to malicious content. Figure 2 shows the process of WCP: 1)The attacker sends a carefully crafted malicious HTTP request. 2)The cache server fails to filter the malicious request and forwards it to the web server. 3)The malicious request triggers an exception at the web server, resulting in a harmful response that the cache server stores. 4)A normal request initiated by the victim hits the cached malicious resource.

These two attacks have two main differences. (1) The attack techniques are different: WCD achieves its goal by constructing abnormal URLs, it requires the victim to click on the malicious URL to deceive the cache. WCP can exploit various parts of the HTTP request to poison the cache, directly resulting in the victim receiving abnormal responses. (2) The attack objectives are different: WCD aims to steal sensitive information from the cache, whereas WCP aims to make the victim access error responses in the cache. Researchers have conducted extensive measurement studies on WCD [24, 25]. However, there is currently a lack of large-scale measurements regarding WCP. This study focuses on the research gap in the deficit of a global WCP threat overview, by designing and implementing the *HCache* to study the severity on the Internet.

WCP has the merit of a wide-range attacking impact with just a simple attack. Specifically, attackers only need to send a single attacking request, while affecting numerous global Internet users. The larger the traffic of a website, the greater the impact it can cause. In the entire attack chain, WCP can be conducted in conjunction with other attacking techniques to broaden the attack surface, and their final impact closely depends on the injected malicious payloads. If an error response is returned, it can lead to a Denial of Service (DoS) attack. If the response is dynamically generated, injection of JavaScript code can result in Cross-Site Scripting (XSS) attacks. If the location of redirect responses can be manipulated, arbitrary page replacement can occur. In a word, when combining WCP with other attack methods, the severity can be further expanded.

2.3 Limitation of Existing Research

Current studies share a common limitation as they are all case-by-case investigations heavily reliant on empirical knowledge. Chen et al. proposed a new method for WCP by exploiting the Host header, termed "Host of Trouble" [1]. James Kettle introduced a novel technique to execute such attacks using HTTP request fields, including X-Forwarded-Host, request parameters, fat get request [16, 17]. Nguyen et al. proposed CPDoS, using three methods to conduct a DoS attack [28]. Mirheidari et al. conducted large-scale measurements on the impact of WCD on the Internet [24, 25].

These studies have two main limitation: (1) They are case-by-case studies and do not systematically analyze the cache poisoning vectors that may result from different HTTP fields, which could miss many new attack vectors, as we demonstrate later; (2) They lack large-scale measurements. Existing studies have either only conducted manual testing for CDNs and HTTP implementations, or only conducted small-scale testing for certain attack types, leading to many vulnerability instance undiscovered. Therefore, there is an urgent need for a systematic tool capable of conducting large-scale measurements to identify WCP vulnerabilities.

3 OVERVIEW

3.1 Threat Model

Essentially, web cache poisoning (WCP) attacks stem from the problem with *cache key*. The cache key serves as the unique identifier to locate and isolate cached objects, determining whether a request hits the cache or not. Figure 3 presents an example of cache keys in HTTP requests. It typically consists of the request method, hostname, and URI. A cache hit occurs when a new request matches the cache key of a previous stored object that still remains valid within the cache; if not, the resource is retrieved from the web server.

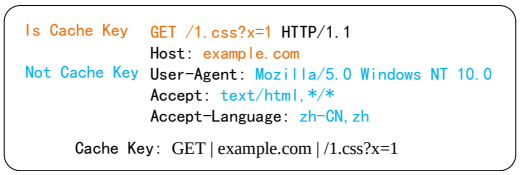

Figure 3: A example schema for cache keys in HTTP request

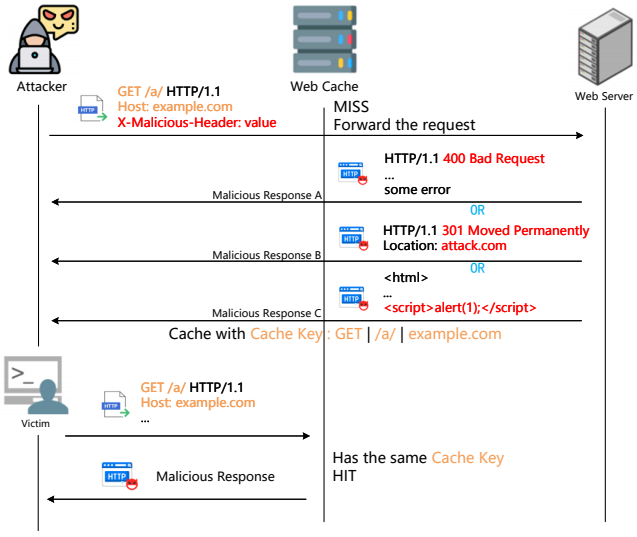

Figure 4: An example of web cache poisoning

Figure 4 presents a example of WCP, where an attacker constructs a malicious request with evil content in the headers. The

cache server forwards this request, triggering a malicious response from the web server. Malicious responses could be an error page, a redirection to a 3rd-party website controlled by the attacker, or a page containing malicious content. Finally, the cache server then caches this evil response, and victim requests with the same cache key hit the poisoned cache, leading to a WCP attack. While WCP has posed a severe threat to the Internet, there is lack of systematic evaluation of such vulnerabilities at scale.

3.2 Methodology

In this paper, we present a novel testing methodology to detect WCP on the Internet. However, developing such a methodology needs to answer the following research questions.

Q1: How can we generate testing requests to systematically probe web cache poisoning vulnerabilities?

Previous works [24, 25, 27] usually utilize manual approaches or collect known exploits to generate testing requests, and do not systematically explore various HTTP fields and specific caching behaviors. This can lead to incomplete testing and the potential oversight of new attack vectors. To address this, we have developed a cache-key-aware approach to systematically generate and mutate HTTP requests to uncover WCP vulnerabilities. We start with standardized HTTP requests to incorporate typical header fields by leveraging syntax rules derived from HTTP RFCs. We then enumerate different HTTP fields such as request line, headers, and body to uncover those fields not included in cache keys. Then we mutate non-cache-key fields and body of requests to probe inconsistencies between web caches and web servers, aiming to uncover potential exploits. This allows for a more targeted and systematic generation of test cases for essentially identifying potential WCP issues.

Q2: How can we detect Web Cache Poisoning accurately?

We design a three-phase testing approach to detect WCP accurately. First, we send a normal request to establish a baseline response. This is followed by a especially crafted request, where potential vulnerabilities are systematically tested. The response to this request is then compared to the baseline response, identifying discrepancies that may indicate a successful poisoning attack. Finally, a validation request is sent to confirm the initial assessment of WCP vulnerability. This approach allows us to pinpoint the exact conditions under which WCP can occur, providing a reliable means of assessment.

Q3: How can we assess the impact of WCP while minimizing disruption to normal users?

Minimizing the impact on normal users while assessing WCP is crucial. To achieve this, we employ *cache buster* variables in our request parameters to isolate web caches. These variables, crafted as unique random values and cache keys, ensure that normal user requests do not intersect with our crafted testing requests, thereby preventing access to potentially poisoned caches. This technique ensures that our testing process does not disrupt the normal operations of the website or the access of legitimate users, while still maintaining the high efficacy of WCP detection.

4 HCACHE: DESIGN AND IMPLEMENTATION

4.1 Workflow

Based on the above methodology, we developed *HCache*, a large-scale detection system to detect web cache poisoning (WCP), depicted in Figure 5. *HCache* comprises three core modules: the *Preprocessing Module*, the *Test-case Generation Module*, and the *Cache-poisoning Detection Module*.

(1) The *Preprocessing Module* processes the seed domain list through expansion, survivability checks, deduplication, and cacheable URL identification, outputting detectable URLs.

(2) The *Test-case Generation Module* identifies cache keys, produces standard requests, and generates test cases for potential WCP.

(3) The *Cache-poisoning Detection Module* synthesizes the prior modules' outputs to craft attack requests and assesses WCP vulnerabilities using varied attack payloads.

The following paragraphs present detailed information on the related working steps and specific modules.

4.2 Stage A. Preprocessing

First, the list of URLs to be tested needs to be determined before the following real-world measurement. Thus, the *Preprocessing Module* includes the initial three steps, including subdomain extension, target URL finding, and URL deduplication.

Step A.1) Subdomain Extension. Starting from initial domains, this process recursively crawls related HTTP/HTTPS pages to gather subdomains with a 200 status code, thereby expanding the domain list for further steps. Domains that do not return a 200 status code are disregarded, as they are not typically accessed by web clients. The next step then generates the initial set of URLs for testing based on the collected subdomains.

Step A.2) Target URL Finding. This component is a website crawler that uncovers URL resources through deep traversal and automates website visits using Python's Requests library. To enhance efficiency for large-scale detection of popular websites, it operates with multiple concurrent threads. In summary, the program sequentially crawls the target domain's homepage, extracting static resources such as JavaScript, images, and videos.

Relevant studies indicate that using the HTTP header fields in the response (e.g. 'age', 'x-cache') to determine whether a page is cached is a relatively accurate method[25]. Pages detected using this approach form a true subset of all cached pages, as certain websites may omit cache-related information in their responses. We referenced official documentation from major caching vendors to understand the specific caching behavior of different cache identity headers. Additionally, the crawler discovers numerous related URLs on third-party websites, including OSS storage, JS hosting, and self-built CDN services, and automatically adds these domains into the domain discovery list.

Step A.3) URL Deduplication. The deduplication module enhances the efficiency of large-scale cache-poisoning detection. Many web applications generate customized pages based on query strings or URL path parameters, leading to similar URL structures being cached together with the same vulnerabilities. Exhaustive testing of each URL is time-consuming and resource-intensive. To avoid redundant detection of similar URLs, obtained URL lists are processed. Utilizing the SimHash algorithm [31], we developed a program for

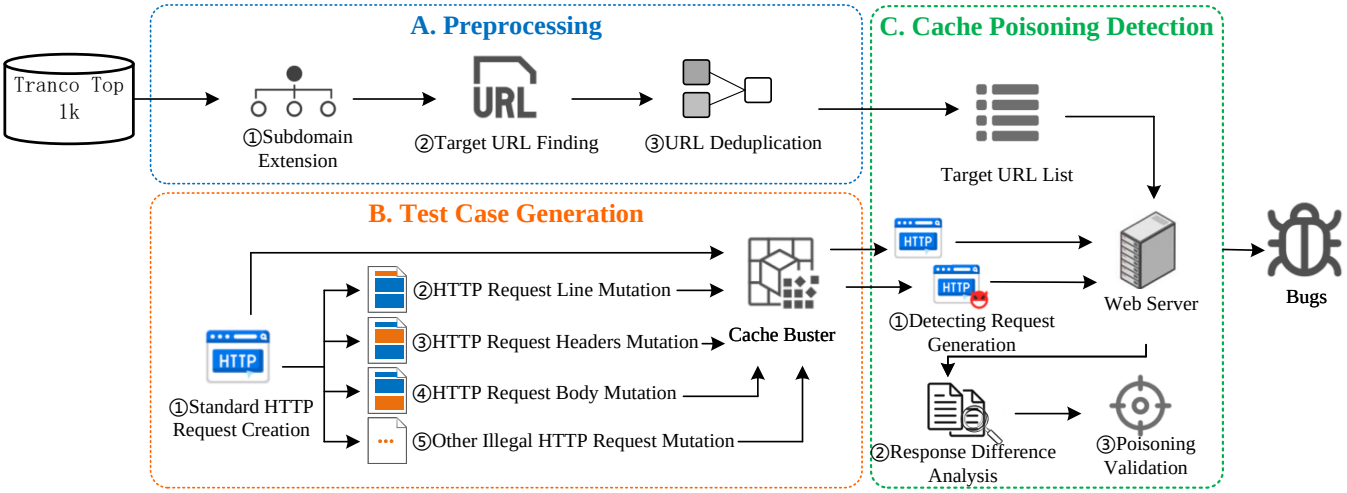

Figure 5: Overview of our large-scale measurement system:*HCache*

fuzzy matching and URL similarity calculation to consolidate similar URLs.

For example, *example.com/users/bob/blog1* and *example.com/users /alice/article2* may exhibit high similarity. Initially, we generalize them based on letters (represented by C), numbers (represented by D), and special characters (represented by S): *example.com/CCCCC /CCC/CCCCD*. Subsequently, we assign weights according to the hierarchical levels of the path, where higher-level directories have greater weights. Next, we use a directory of different levels as keywords to calculate feature vectors. We compute similarity by utilizing the Hamming distance between feature vectors, and URLs with excessively high similarity are deduplicated. In the end, this process yields a set of URLs for testing, and filtering out URLs in this manner significantly reduces the testing workload. It also avoids overconsumption of the target server's resources with redundant scans.

4.3 Stage B. Test Case Generation

The test case generation is the core module of *HCache* that outputs different request variations to comprehensively cover different WCP methods. It includes standard HTTP request generation, cache key detection, and multiple request mutation methods.

Step B.1) Standard HTTP Request Creation. Informed by expert insights and traffic analysis, we've crafted standard HTTP request templates for common methods like HEAD, GET, and POST. These templates are designed to avoid rejection by mimicking normal HTTP traffic, including typical header fields like *'Host'*, *'User-Agent'*, *'Cookie'*, and *'Accept-Encoding'*, with the 'Host' field adapting to the target domain automatically. This equips *HCache* with a basic suite of HTTP requests.

Step B.2) HTTP Request Line Mutation. The HTTP request *Line*, comprising the *Method*, *URI*, and *Protocol Version*, is often a cache key, thus we explore the impact of different fields of non-cache keys, such as method case insensitivity, parameter changes, and protocol version arbitrarily specified variants. WCP can occur when a non-cache key field affects content generation or causes server errors. For parameter mutation, we collect a list of common parameters, which *HCache* utilizes to mutate HTTP request parameters.

Step B.3) HTTP Request Headers Mutation. The request header includes fields both from standard RFC specifications and popular web servers and CDN vendors. This complexity, coupled with variations between middlebox and web server, often leads to inconsistencies and potential WCP vulnerabilities. It also brings a great challenge to the detection of WCP: how to cover as many types of attacks as possible? To this end, we propose the following variants based on the characteristics of different headers.

i.Request Headers Scanning: Some fields in the HTTP request header may also affect the web server's execution logic. A common trick is to utilize forwarding headers (e.g., *'X-Forwarded-Host'*, *'X-Forwarded-Scheme'*, *'X-Forward-Port'*), which are often used to pass information among multi-hop HTTP servers. WCP occurs when the cache server uses these fields for routing without adding them to the cache key. Similarly, web server that fetches cookie fields to generate readback data dynamically is vulnerable. Meanwhile, numerous real-world headers may dynamically affect the caching results, and different CDN vendors have their customized headers for access control. This method involves gathering common request headers on the Internet and systematically altering HTTP requests with these headers to evaluate their effect on WCP.

ii.Special Headers Scanning: Certain HTTP request headers, as defined in RFCs, have specific value requirements, such as the *'If-Unmodified-Since'* header specifying a date format. Besides, web servers will format the header of a request, if a header's value does not conform (e.g., a random string), it's disregarded by web servers, hindering WCP detection. To address this, we generate syntax-compliant values that adhere to RFC specifications for testing.

iii.Blacklist HTTP Request Mutation: While WAFs block scanners or crawlers by common filtering mechanisms (eg. return *403 Illegal Access Response* when detecting *'User-Agent'* as SQLMap), some cache servers may not include *'User-Agent'* in the cache key, creating an opportunity for WCP. *HCache* employs a blacklist-based

mutation mechanism that assesses the impact of security scanners (e.g. Nuclei) and web crawlers (e.g. PyCurl) on the cache. Additionally, it tests the cache's resilience to malicious *'Referer'* messages from phishing sites and common blacklist strings used by WAFs (e.g. *<script>alert(1)</script>*).

Step B.4) HTTP Request Body Mutation. While GET requests typically lack a body, some HTTP services process bodies in GET requests, causing abnormal behaviors like redirects or 400 error responses. Additionally, rewriting methods like *'X-HTTP-Method-Override'* can extend the attack payload. When a cache server transparently forwards such requests, and the web server responds with an exception consequently, it becomes susceptible to WCP.

Step B.5) Other Illegal HTTP Request Mutation. Beyond mutating the three main components of the HTTP request, we crafted other illegal HTTP requests to probe WCP vulnerabilities, examining the effects of overly long headers and invalid characters.

Cache Buster. To finalize the test requests for WCP, we employed a *cache buster* with two objectives: on the one hand, modifying the value of the *cache buster* avoids interactions between targeting the same URL and prevents invalidation caused by new attack requests hitting the previous cache. On the other hand, it ensures that normal user requests do not trigger responses poisoned by our tests, as they do not carry our randomly generated *cache buster*.

4.4 Stage C. Cache Poisoning Detection

Under this component, *HCache* first initiates WCP detection for each URL in the pending list, then analyzes the response to identify vulnerabilities. *HCache* performs multiple rounds of WCP testing rapidly using multi-threading, encompassing request generation, response analysis, and cache poisoning validation.

Step C.1) Detecting Request Generation. This module is used to generate three HTTP requests, which are normal request, attack request and validation request. The normal request is obtained by adding the request parameter A to the standard request generated in Step B.1), which aims to check whether the cache buster is effective and collect the normal response of the target website for subsequent analysis. The attack request is obtained by adding the different request parameter B from the test cases generated in the previous stage. The validation request is similar to the normal request, the only difference is it has the same request parameter B as the attack request.

Step C.2) Response Difference Analysis. *HCache* identifies potential WCP by analyzing differences between the response returned by a normal request and an attack request. It assesses three types of information: a) whether the status code of the HTTP response has changed; b) whether the length of the HTTP response body has changed; c) whether the HTTP response contains additional content of the poisoning request compared with the normal request. If one of the above conditions occurs, *HCache* determines that the target server may be threatened by WCP.

Step C.3) Poisoning Validation. When *HCache* finds a website that may have WCP vulnerabilities, it will use the validation request to verify if the cache will be poisoned. This validation request is sent within 1 second to verify that the WCP vulnerability can be successfully exploited. If the website is vulnerable, the validation response matches the last poisoned content, and the cache identity field should display *"HIT"*.

False positives in the measurement process are caused by multiple similar requests from the same client being rejected by the web server. When both an attack request and a validation request return the same error response, *HCache* mistakenly assumes that the error request was cached. In order to eliminate false positives, *HCache* will initiate two subsequent tests of the potential WCP vulnerability detected after a certain period. And all discovered potential vulnerabilities will be cross-validated on clients in different regions. Finally, we also manually verified the discovered WCP vulnerabilities.

5 MEASUREMENT STUDY AND FINDINGS

5.1 Data Collection

Our work use Tranco Top 1,000 domains as seeds, and extracts a total of 114,560 subdomain information, among which 31,350 surviving websites can be accessed via HTTP(S). On this basis, more resource is crawled on these websites by the crawler, thus expanding the target domains to 4,427,590 different URL links. To increase the testing efficiency, URLs with similar paths are de-emphasized during the experiment, and finally, 1,417,004 URL links are obtained. Then, the websites that contain the cache identity header in the HTTP response packet are selected as targets for testing. A total of 22,114 domains containing 51,596 different URL links were tested in this chapter. Then we conducted detection measurements from 7 different VPS servers across the world, such as New York, Frankfurt, Sydney and Tokyo. For each detected case, multiple repeated experiments are conducted across different geolocations to eliminate accidental false positives that may arise. In the end, more than 1,300 websites were found to have web cache poisoning (WCP) vulnerabilities, containing 1,556 different URL links.

5.2 Cache Key Detection

To prevent the poisoned cache from affecting normal users during testing, we use a cache buster to isolate the cache. The test request must carry a crafted cache key different from the normal user's request, and the cache key used for the cache buster should be "irrelevant" and its modification must not affect the normal response content. To this end, we designed a pre-experiment on cache key detection to find the best cache buster.

We determine which fields are commonly used as cache keys by modifying different parts of the HTTP request. From all the cachable URLs detected, URLs were randomly selected for each accessible domain of the Tranco top 1,000. In most cases, if the parameter cannot be recognised by the server, it will ignore without affecting the corresponding content, indicating that the request parameter is a kind of effective cache buster. It will be used in the subsequent large-scale cache poisoning measurement to avoid affecting the normal user's access.

5.3 Overview

We conducted large-scale WCP detection experiments on popular websites on the Internet, and found 1,354 WCP vulnerabilities, affecting some world-famous websites, which have high Tranco

Table 1: Newly discovered attack vectors by *HCache*

Type	Common Attack payloads	Vulnerable Websites*
Internal Route Header Attack	X-Request-Id: 123456789	wikia.com
	Fastly-Client-Ip: 123456789	fandom.com
	Gpt-Tags-Enabled: 123456789	ipage.com
	X-Amz-Request-Id: 123456789	stanford.edu
	Fastly-Soc-X-Request-Id: 123456789	domain.com
	X-Amz-Website-Redirect-Location: 123456	marriott.com
HTTP Identification Header Attack	Auth-Key: 123456789	sinaimg.cn
	X-Auth-User: 123456789	bing.com
	Authorization:123456789	wsimg.com
	X-Authorization: 123456789	ziffdavis.com
	Client-Proxy-Auth-Required:123456789	ccmbg.com
HTTP If Header Attack	If-Match: 123456789	usa.gov
	If-Range: 123456789	aig.com
	If-None-Match: 123456789	bluehost.com
	If-Modified-Since: 123456789	starbucks.com
HTTP Protocol Header Attack	X-Forwarded-SSL: on/off/nonsense	pcmag.com
	X-Forwarded-Scheme: nothttps/http(s)	cisco.com
	X-Forwarded-Proto: http(s)/ssl/nonsense	mashable.com
	X-Forwarded-Protocol: http(s)/nothttps/nonsense	getflywheel.com
HTTP Range Header Attack	Range: bytes=cow	stats.com
	Range: bytes=9-4	miele.co.nz
	Range: bytes=-1024,0	starbucks.com
	Range: bytes=0-,0-,0-,0-	chiltondiy.com
HTTP Upgrade Header Attack	Upgrade: 123456789	lefigaro.fr
	Upgrade: HTTP/0.9	smtp2go.com
	Upgrage: Websocket, RTA/x11	salesforce.com
	Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9	disney.io
HTTP Coding Header Attack	Accept: 123456789	fcc.gov
	Accept-Encoding: 12345	house.gov
	Transfer-Encoding: error	europa.eu
	zTRANSFER-ENCODING: asdf	landmarkcinemas.com

*: The vulnerable websites in the table only show the base domain. The subdomains and paths were redacted for ethical considerations.

Table 2: Detection datasets and vulnerable websites statistics

	Initial domain name	Domain name extension	Cache pages	Cache Poisoning Vulnerabilities
Number of domain names	1,000	114,560	22,114	1,354
Number of URLs	-	1,417,004	51,596	1,556

rankings and a large amount of web traffic, as shown in Table 1 and Table 2. Besides, some websites may even have more than one vulnerabilities. Once an attacker compromises these websites through one of the identified WCP vulnerabilities, it will affect a large number of global Internet end-users.

We compare our detection results with existing studies in Table 3 and Table 4. Compared with previous work, our study is more systematic and comprehensive in terms of attack vector coverage and measurement scale, with many new attack methods and vulnerabilities discovered. In total, 14 types of attack techniques are discovered by *HCache*, 7 of which are newly discovered vectors.

Figure 6 shows the percentage of different attacks, from which we can find that known attacks still account for more than half of the websites found to have WCP vulnerabilities, indicating that various vendors are still not in place to protect against known WCP attacks. In addition to the known issues, we also found that many other new HTTP fields may lead to WCP. This suggests that any non-cache key could potentially be at risk of WCP. Protection against a single attack method is not enough to fully defend against the effects of WCP.

Figure 7 presents the distribution of vulnerable websites with respect to their Tranco ranks, exhibiting a fairly uniform. This suggests that Web Cache Poisoning is pervasive among the websites in our dataset with no strong connection to their popularity ranking.

Moreover, we tested the impact of WCP in HTTP/2, using the same variant of the scanning test on websites deployed with HTTP/2. We found that all the vulnerabilities that existed in HTTP/1.1 still existed in HTTP/2. About 90% of the websites share caches between

Table 3: Number of websites with ≥ 1 vulnerabilities found by *HCache*

	Attack type	Number
New Attack Vectors	Internal Route Header Attack	237
	Identify Header Attack	118
	If Request Attack	79
	Protocol Header Attack	69
	Range Request Attack	46
	Upgrade Request Attack	25
	Coding Header Attack	19
Vectors in CPDoS	HTTP Header Oversize (HHO)	269
	HTTP Method Override (HMO)	149
	HTTP Meta Character (HMC)	56
Vectors in Blogs	Forwarded Header Attack	96
	HTTP Parameter Attack	84
	Fat Get Request Attack	67
	Blacklist Attack	40

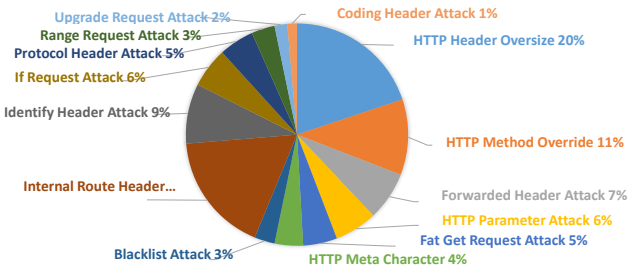

Figure 6: Impact ratio of different attack vectors

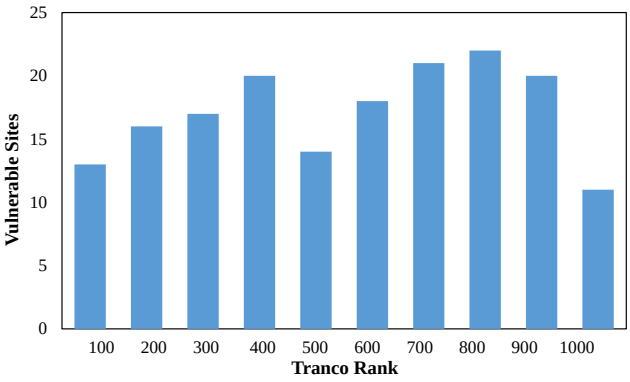

Figure 7: Distribution of vulnerable websites in Tranco ranking

HTTP/1.1 and HTTP/2, i.e., after sending an HTTP/1.1 request to poison a cache, a normal HTTP/2 request afterward will still hit the poisoned cache, and vice versa. This suggests that an HTTP/2 to HTTP/1.1 transition may have occurred, implying that attacks targeting HTTP/1.1 could affect services utilizing HTTP/2.

5.4 Findings

We present an overview of our findings about attack vectors. We identified 14 types of attack vectors that could lead to cache poisoning, among which 7 types are newly discovered. Table 1 shows the new attack vectors we discovered and lists some specific payloads that can cause poisoning as well as the affected websites.

Internal Route Header Attack. A CDN is a large distributed network with a large number of internal nodes that perform different transmission and caching functions. Therefore, CDNs also implement some special headers to pass routing information during internal transmission. Attackers can abuse these headers to trigger CDNs to throw exceptions, ultimately leading to WCP. These headers include *Fastly-Client-Ip, Fastly-Soc-X-Request-Id, X-Amz-Website-Redirect-Location, X-Amzn-CDN-Cache,* etc. This is the attack found to affect most websites besides the HTTP Header Oversize Attack, with 234 websites affected.

HTTP Authentication Header Attack. In certain APIs or gateway systems, authenticating HTTP requests is a common requirement. Some services use headers like *Authorization, X-Auth-User* and *Auth-Key* for this purpose. An attacker can exploit this by sending a request to the cache server with these headers. The cache server forwards them to the web server. The web server finds that the value of the header is illegal and returns a response with a denial of access. The cache server retains the incorrectly cached resource, returning it for equivalent requests. *HCache* found 118 websites have this problem.

HTTP Protocol Header Attack. Cache servers use headers like *X-Forwarded-SSL, X-Forwarded-Scheme, X-Forwarded-Proto*, and *X-Forwarded-Protocol* to identify client connection protocols. However, these headers may impact web server processing. Some servers respond with a 301 redirect. If the redirect request retains these headers and redirects to the URL itself, it causes a DoS attack due to excessive redirects. As per the HTTP standard, 301 responses are cached, leading victims to hit the cache. In this scenario, if an attacker utilizes headers such as *X-Forwarded-Host* to control the redirected link address, it becomes easy to direct victims to a malicious site for subsequent attacks. A total of 69 websites are vulnerable.

HTTP Range Header Attack. Clients utilize the Range header to request specific portions of a resource, widely supported by most intermediate servers for tasks like multi-threaded downloads. However, certain web servers lack support, leading to potential semantic differences with cache servers. Some web servers may support Range requests but report errors when processing malformed ones (e.g. *Range: bytes=100-90*). *HCache* found 46 websites have this problem.

HTTP If Header Attack. HTTP standard headers like *If-Match, If-Range*, and *If-Modified-Since* determine if a web server meets specified conditions. However, *HCache* discovered some web servers generate 4xx or 5xx errors when processing these requests. If the cache server caches this status code, it will result in WCP. HCache found 79 websites have this problem.

HTTP Upgrade Header Attack. HTTP protocol allows upgrading an established connection to a new, incompatible protocol using mechanisms like *Upgrade: Websocket*. If an attacker initiates

Table 4: Comparison with existing research

Research	Attack Vector	Target	Vulnerable Websites
CPDoS [27]	HHO HMO HMC	Alexa top 500	12
Redefining Unexploitable blog[16]	Forwarded Attack	Manual testing	11*
Novel Pathways to Poisoning blog [17]	Para, Blacklist, and Fat GET	Manual testing	8*
Our work	14 types of attack	Tranco top 1,000 domains and their subdomains	1,354

*: The authors did not fully disclose the number of vulnerabilities in their blogs, and the statistics in the table are derived from the cases in their report.

an unsupported upgrade request (e.g., *Upgrade: HTTP/3.0*) or a malformed one (e.g., *Upgrade: HTTP/0.9*), web server may return an incorrect status code, potentially leading to a WCP. *HCache* found 25 websites have this problem.

HTTP Coding Header Attack. The HTTP protocol uses headers like *Accept, Accept-Encoding*, and *Transfer-Encoding* to identify encoding formats. If an attacker sets a malformed or illegal value in these headers, it may trigger an exception at the web server, potentially resulting in WCP. *HCache* found 19 websites have this problem.

What's more, *HCache* also found many websites have known attacks. Although these attacks have been presented in previous articles[16, 17, 28], they still account for more than half of all vulnerabilities, so it is necessary to analyze how such attacks are exploited.

HTTP Header Oversize Attack. The HTTP protocol standard does not impose a limit on the length of the request header. Therefore, different Web middleboxes implement different restrictions. A DoS attack may exist if the request length allowed by the cache server exceeds the limitations of the web server. An attacker can initiate an HTTP request with a length between the cache server and web server. The cache server forwards the malicious request to the web server, and an error response triggered at the web server that would have resulted in a DoS attack had it been cached by the cache server. Although this vulnerability is a known one and has been disclosed for many years, it still affects the most targeted websites with a total of 269.

HTTP Method Override Attack. HTTP defines request methods like GET, POST, DELETE, and PUT. Some systems only support GET and POST. To overcome this, web frameworks use helper headers like *X-HTTP-Method-Override*. Attackers may exploit this by sending a GET request with an override field set to DELETE. If the server doesn't handle DELETE requests, it returns a 405 error. As per RFC9110, cache servers cache this error, causing subsequent equivalent requests to result in a DoS attack. A total of 149 websites were found to have this issue.

HTTP Meta Character Attack. This attack utilizes a request header with harmful metacharacters, exploiting semantic differences between the cache server and the web server. The cache server may tolerate certain special characters, forwarding them, while the web server, processing the request, triggers an error page, resulting in a DoS attack. Metacharacters involved could include control characters like newline (\r), carriage return (\n), or any Unicode control character. Attackers leverage this to launch WCP against vulnerable websites. *HCache* found 56 websites vulnerable to this attack.

Fat GET Attack. Cache servers usually cache GET requests by default, excluding the HTTP request body as a cache key. Despite the HTTP standard prohibiting GET requests from having a body, some web applications parse fat GET request bodies, allowing dynamic responses. This opens the door to WCP. *HCache* enhances detection with headers like *X-HTTP-Method-Override*, expanding the attack vector. The web server, influenced by *X-HTTP-Method-Override*, treats the request as a POST, attempting to generate a dynamic link from the body. The cache server, ignoring this, uses the cache key of the original GET request and URL. When a user triggers a regular request hitting the attacker's tainted cache, content hijacking occurs. *HCache* found 67 websites has this problem.

HTTP Parameters Attack. There are many applications that choose to extract parameter values from requests to dynamically generate response content. If the web server uses the values in the request parameters to dynamically generate content, and the web server does not perform any filtering on the string, an attacker can construct an XSS attack payload to launch an attack. If the cache server's cache key does not contain the request parameter fields in the URL, the cache is hit when a normal user initiates a request, resulting in malicious cache samples being distributed to the client, ultimately causing an XSS attack. Similar flaws were found on 84 websites.

HTTP Forwarded Header Attack. Reverse proxies (e.g., load balancers, CDNs) rely on routing host information to determine the web server for fetching web resources. RFC7239 introduces the Forward header for this purpose. However, headers like *Host, X-Forwarded-Host, X-Forwarded-Port*, and *Forwarded* are commonly used by reverse proxies to identify the original routing host. This can be exploited for WCP. Attackers can manipulate these headers to control the cache server's routes back to the source, potentially causing the cache server to read malicious data or the web server to reject responses. As these headers are not part of the cache key, victims may unwittingly hit the attacker's poisoned cache, leading to an attack. *HCache* found 96 websites have this vulnerability.

Blacklist Attack. WAFs often use blacklists to block malicious traffic. *HCache* explores three blacklisting mechanisms: manipulating User-Agent with security scanners (e.g., sqlmap) and crawlers (e.g., Crawler), inserting known phishing site domain names (e.g. spam.com) into the Referer header, and randomly adding common attack payloads (e.g. *<script>alert(1)</script>*) to certain headers. *HCache* exploits inconsistencies in blacklist support between cache

server and web server. An attacker sends a request with a malicious string, triggering an exception on the web server. The web server's WAF responds with a 403 Forbidden Access. The cache server incorrectly caches the resource, blocking even normal users from accessing the target site. *HCache* found 40 websites have this problem.

6 THREAT ANALYSIS

The victim website can suffer from various losses, such as reputation degradation, supply chain attacks, or even monetary loss. In this paper, we further categorized these WCP vulnerabilities based on the specific attacking threats. Table 5 shows the vulnerabilities that can result from different types of attacks.

Table 5: Threats that stem from different attack vectors

	DoS	XSS	AUR*
Internal Route Header Attack	✓		
Identify Header Attack	✓		
If Request Attack	✓		
Protocol Header Attack	✓		✓
Range Request Attack	✓		
Upgrade Request Attack	✓		
Coding Header Attack	✓		
HTTP Header Oversize (HHO)	✓		
HTTP Method Override (HMO)	✓		
HTTP Meta Character (HMC)	✓		
Forwarded Header Attack	✓	✓	✓
HTTP Parameter Attack		✓	
Fat Get Request Attack		✓	
Blacklist Attack	✓		

*AUR: Arbitrary URL Redirection

Cache Poisoned Denial of Service. DoS attack is the most basic attack that can be caused by web cache poisoning (WCP). It can be caused by simply constructing an attack request that triggers an error at the web server. We found that even though CPDoS has been disclosed for many years, there are still many websites that are subject to such attacks, such as *harvard.edu, taobao.com, mail.ru*, and *huawei.com*.

We have also found many other HTTP headers that can lead to DoS attacks. All of the 7 new attack vectors discovered by us can cause service inaccessibility on subdomains of *adobe.com, intuit.com, skype.com,* and *sina.com.cn*, etc. A common feature of this type of attack is that the cache server does not comply with the RFCs and caches error status codes that should not be cached. Even if the RFCs were followed, *X-HTTP-Method-Override: NONSENSE* can be used to poison*visualstudio.microsoft.com* with 405 Method Not Allowed. *nvidia.com* and *sap.com* will return 404 Not Found when processing a request with *X-Forwarded-Host: attack.com*. Both 404 and 405 response status codes are heuristically cacheable in RFC.

Cache Poisoned Cross-Site Scripting. An attacker can exploit these WCP vulnerabilities to launch beyond DoS attacks on victim websites. When exploited in conjunction with other attacking techniques, it may also lead to more severe damage. In fat GET attack and request parameter attack, the web server dynamically generates a response using the request parameters or request body, but the cache server caches these dynamically generated responses as static pages. Therefore, the attacker can inject malicious XSS payloads into the response. *HCache* found that some websites will include parameters or request body in the response. Our further validation revealed that some websites do not filter request content and can inject XSS payloads. *edu.sina.com.cn, in.ign.com* and *blackfriday.com* have such vulnerabilities.

Cache Poisoned Arbitrary URL Redirection. Previous research found X-Forwarded-Host can be used to control the actual response page, but our results show that when websites receive these headers containing unknown URLs (such as *X-Forwarded-Host: attack.com*), they will ignore them or return an error response such as 400 Bad Request. It suggests that many websites have already fixed this vulnerability. However, we newly discovered X-Forwarded-Proto header can re-establish the connection, and will return 301 redirect responses. Combined with the X-Forwarded-Host header, the redirected page can be controlled, resulting in an arbitrary page redirect attack. The attacker can implement subsequent higher-order attacks if the victim accesses the attacker-controlled page.

Take one of the subpages in *themeforest.net* as an example. First, we establish an HTTPS connection with it. Then we can send a request with *X-Forwarded-Scheme: http* and *X-Forwarded-Host: attack.com*. The former changes the protocol to HTTP and returns a redirection response, while the latter specifies the response's location, redirecting to an attacker-controlled website. 301 Moved Permanently is cached by the cache server, causing subsequent victim requests to be redirected to attacker-controlled pages as well.

7 DISCUSSION

7.1 Responsible Disclosure

We try our best to responsibly disclose the related vulnerabilities to the vendors of affected websites. First, we actively contacted the affected vendors through several third-party vulnerability disclosure platforms (e.g., Hackerone, Bugcrowd, and Intigriti), discussing the security issues and related mitigations. Second, we have sent notification emails to the administrators of the affected websites, disclosing the vulnerabilities and the specific detection methodologies. According to the rank of vulnerable websites, we summarize the related responses to responsible disclosure below:

Microsoft *microsoft.com (6th in Tranco Top 1,000)*: Responded that they have shared the report with the inner responsible team, and they will take appropriate actions as needed to help their customers be well protected.

AliBaBa *taobao.com (23th in Tranco Top 1,000)*: Confirmed and patched the discovered DoS attack vulnerability, assessed the vulnerability as *Medium Critical*, providing a vulnerability bounty of about 100$.

Adobe *adobe.com (65th in Tranco Top 1,000)*: Confirmed the vulnerability and discussed the scope of the attack. They responded that they are evaluating the vulnerability internally and will provide a fix for the vulnerability in the near future.

NetEase *163.com (187th in Tranco Top 1,000)*: Rated the vulnerability as *Medium Critical*.

Yelp *yelp.com (207th in Tranco Top 1,000)*: Thanked for the results of this research and acknowledged the issues identified in this paper. They will continue to monitor the subsequent impact of the vulnerability and fix the issue when appropriate.

Mashable *mashable.com (426th in Tranco Top 1,000)*: Confirmed the vulnerability and highly praised the work, and suggested looking deeper into the potential harm of the attack, such as using the 'X-Forwarded-Host' header to discover more vulnerable assets internally.

HuaWei *huawei.com (537th in Tranco Top 1,000)*: Confirmed this problem and agreed that it was caused by irregularities in the Nginx cache configuration. They rated our reported vulnerability as *Medium Critical* and awarded about 200$ for the vulnerability.

SAP *sap.com (969th in Tranco Top 1,000)*: SAP has released the fix for this issue, and they offer acknowledgment by publishing our team information on its webpage.

Knowyourteam: They specifically thanked the researcher for the vulnerability report and have started the vulnerability remediation process. Also added our team to the list of vulnerability-fixing acknowledgments and gave some vulnerability bounty 100$.

Street Context: They rated the vulnerability found in this paper as *Medium Critical* and awarded about 300$ for the vulnerability.

VidaXL: They thanked the work of this paper and considered it valuable research. They evaluated our discovered vulnerabilities as *High Risk* and gave a vulnerability reward of about 300$.

BlackFriday, Asana, YoYoGames, Ziff Davis, Nutanix, Starbucks, WP Engine: Acknowledged and thanked us for the vulnerability report and advised that "The issue has been identified internally and is in the process of being fixed".

7.2 Mitigation

WCP is a complex and severe security problem, it is not a vulnerability within a single caching system, but rather the vulnerability of parsing differences between multiple caching systems. As a result, traditional static analysis and white-box testing techniques on a single system are difficult to detect and eliminate the problem. A recommended solution is to employ several methods together in production environments to minimize the cache poisoning problem.

Add additional headers as the cache key: From our discoveries, when exploiting headers that have not been implemented as the cache keys within the caching systems, such as '*X-HTTP-Method-Override*' and '*X-Forwarded-Host*', a successful web cache poisoning (WCP) happens. Therefore, it is applicable and beneficial to enforce these headers as the cache keys within the caching systems. With this mitigation, even if the attacker has successfully poisoned the cache with an error response, this poisoned cache is only private to the specific request with the problematic headers. As a normal request does not contain the problematic header, it will not hit the poisoned cache thus invalidating the attack.

Adhere to the RFC specifications: Most vulnerabilities found by the *HCache* are caused by the caching of error responses that are maliciously triggered by attackers, while these caching behaviors are implementation-specific and not specified by the related RFCs. Therefore, the effective mitigation is to strictly follow the RFC standards, only caching the error status codes that are allowed by the RFCs, and returning other status codes directly for requests that should not be cached.

Enhance exception handling at the web server: Based on our findings, an attacker can proactively trigger error responses at the website web server, which results in WCP. Thus, to avoid returning an error response for malformed HTTP requests, we suggest the website server enhance a good exception handling design, which just ignores the problematic request headers and returns a benign response instead, or directly returns an error code indicating that the response should not be cached by any on-path cache servers. Thus, normal users still obtain the correct response, invalidating WCP.

Disable caching of dynamic resources: Web caching should only be applied to accelerate static resources, not dynamically generated pages. Therefore, caching should be disabled for resources that need to be dynamically generated according to request parameters. *HCache* has found that, although resources (such as CSS and JS) are normally categorized as static resources, some websites generate these resources using dynamic templates, actually turning these static-looking resources into dynamic resources. Hence, the best way to fix this is to directly change these resources to static resources. If this dynamic generation feature is essential for the website operation, we suggest clearly indicating the dynamic nature of these resources to disable the caching behavior. Besides, the website can also add various XSS filters to proactively defend against WCP resulting from the dynamically generated web content.

Reduce the caching time of error pages: The caching system can also reduce the impact by only caching error response within a short time, such as 1 second. This approach can proactively limit the effective time of WCP and greatly increase the attacking difficulty.

7.3 Limitation

Due to the complexity of WCP and the scale of our measurements, our work still has the following limitations, which can be further optimized in future works.

Testing scope. Our research only analyzes individual websites from the top 1,000 Tranco domains. However, our proposed tool *HCache* is also applicable to wider measurement, which apparently can further reveal the severe threat of WCP on the Internet.

Detection on caching behaviors. HCache detection presupposes that the target caching system adopts relevant header identifiers for cache operations. However, there are still some cache servers in real web environments that do not use such identifiers. Therefore, the websites covered by the tests in this chapter are a subset of websites running cache servers in the real world.

Evaluation on web pages with crucial functionality. HCache does not consider user permissions in detecting WCP. Commonly, websites have service-critical or data-sensitive pages that are only accessible to users that require log in, while which are not included in our work. We believe more severe threats can be discovered when further works incorporate the detection of login-related web pages.

Measurements of poisoning techniques. Our work mainly focuses on the well-known WCP that mostly threatens the Internet, thus *HCache*'s request variant module is based on four types of variant patterns defined by expert knowledge. Although our framework

has attempted to include various attacking techniques as much as possible, there may still be WCP that *HCache* does not cover.

7.4 Ethical Consideration

In this study, we have taken our uttermost care to avoid any ethical concerns both in the design and implementation.

For security concerns, the exception requests generated by our tests conform to the HTTP syntax specification and are only likely to cause the web server to return an incorrect response and then close the connection, without affecting the normal operation of the server. We use cache buster to avoid the impact on normal users; the request parameters of the test request are randomly generated, and normal user requests will not hit the poisoned cache because of the different cache key. In addition, our tests found that most of error responses have much shorter cache times relative to normal responses, and unlike an attacker continuously sending attack requests to poison the cache, our experiments sent only one attack request, thus the poisoned caches will not survive for more than 10 minutes according to the cache time in all experiments. Further, to ensure that the poisoned caches will not continue to exist, we sent normal requests to each potentially poisoned site after our experiments, to make sure that the caches had been refreshed to normal responses. For performance concerns, we filtered a large number of target URL links using URL similarity detection. We strictly limit the request rate, a single URL to 5 requests per second, which will not place an excessive performance load on the websites and CDNs.

For privacy concerns, only URL information related to cache poisoning was captured and analyzed, and no privacy data of the target website was saved locally, nor was any content of the target website indexed and otherwise made public. In addition, we use an HTTP header (User-Agent) embedded with our research purpose and contact information during the scanning process. If website administrators notice any adverse effects caused by the automated scanning on their websites, they can timely contact us, and we will promptly cease the automated scanning of the target website. We strictly followed the principle of responsible disclosure to report discovered vulnerabilities to affected websites, by actively contacting through various channels such as email and third-party security disclosure communities. The case mentioned in the article has already been fixed.

8 RELATED WORK

Our research focuses on web cache poisoning (WCP) caused by non-cache keys in HTTP requests, and delves deep into various details of actual poisoning attacks and exploits. In addition, there are several other attacking tricks to perform WCP or to exploit cache flaws for other purposes. Host-of-Trouble attack exploits inconsistencies in the parsing of the host header in HTTP requests between the cache server and the web server, to perform WCP and WAF bypassing[1].

HTTP Desync Attack poisons the cache by smuggling an additional request to disrupt the responses with malicious payloads[14, 18, 20]. WCD tricks a web cache into erroneously storing sensitive content, thereby making it widely accessible on the Internet[4, 24, 25].

Two detection tools are most relevant to the work in this paper. One is *Param Miner*[15], designed by James Kettle, which is used to scan whether some headers and parameters are included in cache keys to detect potential WCP. Another one is *Web Cache Vulnerability Scanner*[9], which summarizes some of the previously proposed methods of WCP, and allows for the detection of known attack methods. *HCache* works as a superset of these two tools, it analyses the request line, request header, and request body of an HTTP request to generate corresponding test cases that can comprehensively test the different aspects of WCP.

Since James Kettle demonstrated the severity and prevalence of request smuggling in 2019, researchers have come up with several tools to detect attacks on request smuggling[2, 5, 29]. T-reqs is a novel grammar-based differential fuzzer to test HTTP request smuggling[12]. Frameshifter aim to discover the security implications of HTTP/2-to-HTTP/1 conversion anomalies[11]. Large-scale measurements of web cache[27], HTTP(S)[3, 23], CDN[13] e-mails[35] or other web attacks[25], provides insights into the current security problems on the Internet, allowing us to better address potential security risks. To our knowledge, our work is the first large-scale examination of the WCP attack, revealing the prevalence of this threat on the Internet.

In addition to WCP, cache servers, especially CDN, have other security issues. Its working mechanisms can also bring WCP [1, 24], DoS attacks[8], or other forms of attacks[7, 21, 22]. Compared to the above research on *"forwarding"*, our work focuses on *"caching"*, revealing the pervasive security risks posed by the inconsistent processing of requests between websites and cache servers. Our work highlights this widespread systemic problem, which can motivate cache vendors and webmasters to properly implement and configure the caching, strictly adhering to HTTP standards specifications.

9 CONCLUSION

Web cache poisoning (WCP) has been a significant threat on the Internet, however, it still lacks a global view of the severe impact at scale. We have proposed a systematic measuring platform *HCache*, which enables a large-scale evaluation of WCP threats on the real-world Internet. Based on Tranco Top 1K domains and their subdomains, we have discovered more than 1,000 websites across 172 domains (17% of measured domains) with WCP vulnerabilities. Our work first reveals that WCP threat is a widespread security issue on the Internet, and discloses that WCP threat still exists in the new incoming protocols. We have responsibly reported the vulnerabilities to the affected websites, receiving acknowledgments and over $1,000 bug bounties from world-famous companies such as Adobe, Alibaba, Huawei, and Microsoft.

ACKNOWLEDGMENTS

We sincerely thank all anonymous reviewers and our shepherd for their insightful and constructive feedback to improve the paper. This work is supported by the National Natural Science Foundation of China (grant #62272265).

REFERENCES

[1] 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, pages 1516–1527, 2016.
[2] Evan Custodio. Smuggler. https://github.com/defparam/smuggler, 2020.
[3] Zakir Durumeric, James Kasten, Michael Bailey, and J Alex Halderman. Analysis of the https certificate ecosystem. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 291–304, 2013.
[4] Omer Gil. Web cache deception attack. *Black Hat USA*, 2017, 2017.
[5] Mattias Grenfeldt, Asta Olofsson, Viktor Engström, and Robert Lagerström. Attacking websites using http request smuggling: empirical testing of servers and proxies. In *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 173–181. IEEE, 2021.
[6] 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. IEEE, 2018.
[7] 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 *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6185–6202, 2023.
[8] 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 *NDSS*, 2020.
[9] Hackmanit. Web cache vulnerability scanner. https://github.com/Hackmanit/Web-Cache-Vulnerability-Scanner.
[10] Apache http server project. caching guide. https://httpd.apache.org/docs/2.4/caching.html.
[11] Bahruz Jabiyev, Steven Sprecher, Anthony Gavazzi, Tommaso Innocenti, Kaan Onarlioglu, and Engin Kirda. {FRAMESHIFTER}:fram security implications of {HTTP/2-to-HTTP/1} conversion anomalies. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1061–1075, 2022.
[12] 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*, pages 1805–1820, 2021.
[13] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. Unveil the hidden presence: Characterizing the backend interface of content delivery networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2019.
[14] James Kettle. Http/2: The sequel is always worse. https://portswigger.net/research/http2.
[15] James Kettle. Parem miner. https://github.com/PortSwigger/param-miner.
[16] James Kettle. Practical web cache poisoning: Redefining 'unexploitable'. https://portswigger.net/research/practical-web-cache-poisoning.
[17] James Kettle. Web cache entanglement: Novel pathways to poisoning. https://portswigger.net/research/web-cache-entanglement.
[18] James Kettle. Http desync attacks: Smashing into the cell next door. *Black Hat USA*, 2019.
[19] Amid Klein. Divide and conquer. *HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics, Sanctum whitepaper*, 2004.
[20] Amit Klein. Http request smuggling in 2020–new variants, new defenses and new challenges. *Black Hat Briefings USA*, 8, 2020.
[21] 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. IEEE, 2020.
[22] 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. IEEE, 2014.
[23] Abner Mendoza, Phakpoom Chinprutthiwong, and Guofei Gu. Uncovering http header inconsistencies and the impact on desktop/mobile websites. In *Proceedings of the 2018 World Wide Web Conference*, pages 247–256, 2018.
[24] 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, 2020.
[25] 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, 2022.
[26] Nginx. Nginx content caching. https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/.
[27] Hoai Viet Nguyen, Luigi Lo Iacono, and Hannes Federrath. Mind the cache: large-scale explorative study of web caching. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 2497–2506, 2019.
[28] 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*, pages 1915–1936, 2019.
[29] PortSwigger. Exploiting http request smuggling vulnerabilities. https://portswigger.net/web-security/request-smuggling/exploiting, 2020.
[30] Mike Reddy and Graham P Fletcher. An adaptive mechanism for web browser cache management. *IEEE Internet Computing*, 2(1):78–81, 1998.
[31] Caitlin Sadowski and Greg Levin. Simhash: Hash-based similarity detection, 2007.
[32] Squid. Squid: Optimising web delivery. http://www.squid-cache.org/.
[33] Varnish. Varnish http cache. https://varnish-cache.org/.
[34] w3techs. Cloudflare vs. akamai vs. fastly usage statistics. https://w3techs.com/technologies/comparison/cn-akamai,cn-cloudflare,cn-fastly.
[35] Chuhan Wang, Kaiwen Shen, Minglei Guo, Yuxuan Zhao, Mingming Zhang, Jianjun Chen, Baojun Liu, Xiaofeng Zheng, Haixin Duan, Yanzhong Lin, et al. A large-scale and longitudinal measurement study of {DKIM} deployment. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1185–1201, 2022.
[36] WordPress. Wp super cache. https://wordpress.org/plugins/wp-super-cache/.

A CACHE IDENTIFICATION FIELD

Table 6: Common caching status fields used by major service vendors

Cache Service/Software	Response Header	Hit	Miss
Azure	X-Cache	TCP_HIT	TCP_MISS
Fastly	X-Cache	HIT	MISS
Akamai	X-Cache, Server-Timing	desc=HIT	desc=MISS
CDN77	X-Cache, X-77-Cache	HIT	MISS
CloudFront	X-Cache	Hit from cloudfront	Miss from cloudfront
UDomain	X-Cache-Status	HIT	MISS
KeyCDN	X-Cache	HIT	MISS
Cloudflare	CF-Cache-Status	HIT	MISS
GCoreLabs	Cache	HIT	MISS
ChinaCache	X-cc-via	*[H,*]	*[M,*]
Github Pages	X-Cache	HIT	MISS
Google Cloud	cdn_cache_status	hit	mis
Incapsula CDN	X-Iinfo	...0CNN...	...PNNN...
AlibabaCloud	X-Cache	HIT TCP_IMS_HIT	MISS TCP_MISS
Tencent Cloud	X-Cache-Lookup	Hit From * / Cache Hit	Cache Miss
HUAWEI CLOUD	X-Cache-Lookup	Hit From *	Miss From *
Baidu AI Cloud CDN	X-Cache-Status	HIT	MISS
Apache Traffic Server	X-Cache	HIT	MISS
Squid	X-Cache	Hit From *	Miss From *
Varnish	X-Cache	HIT	MISS
Nginx	Cache_status, X-Proxy-Cache	HIT	MISS
Apache	X-Cache	HIT	MISS
Rack Cache	X-Rack-Cache	Hit	Fresh/Miss

B KNOWN ATTACK VECTORS

Table 7: Examples of known attack vectors discovered by *HCache*

Type	Common Attack payloads	Vulnerable Websites*
HTTP Header Oversize	X-Oversized-Header-[1-N]: Big-Value-000000000000...000000000000	taobao.com nvidia.com mail.ru dropbox.com
HTTP Method Override	X-HTTP-Method: PUT	house.gov
	X-HTTP-Method: TRACE	bmw.com
	X-Method-Override: TRACE	mailchimp.com
	X-HTTP-Method-Override: POST	huawei.com
	X-HTTP-Method-Override: DELETE	microsoft.com
HTTP Meta Character	Header\uffff:1234	aadcoinst.com
	X-Metachar-Header: \0	house.gov
	X-Metachar-Header: \b	house.gov
Fat GET	GET /?id=1 HTTP/1.1	nih.gov
	X-HTTP-Method-Override: POST	sina.com.cn
	...	gouvernement.lu
	attack=<script>alert(1);</script>	adobe.com
HTTP Parameters	/app?config=<script>alert(1);</script>//	ign.com
	/base.css?exp=<script>alert(1);<script>	hotelscombined.com
	/index.js?utm_medium=x;callback=alert(1)//	cdlvr.net
HTTP Forwarded Header	Host: example.com:1337	grab.careers
	Forwarded: Host=attack.com	bing.com
	X-Forwarded-Host: attack.com	blackfriday.com
	X-Forwarded-Port: 1337	yoyogames.com
Blacklist	Referer: spam.com	yelp.com
	Referer: <script>alert(1)</script>	alipayobjects.com
	Any-Header:.burpcollaborator.net	salesforce.com
	User-Agent: sqlmap/1.3.11#stable	jfrogchina.com
	User-Agent: Nmap Scripting Engine	alipay.com

*: The vulnerable websites in the table only show the base domain. The subdomains and paths were redacted for ethical considerations.