A Methodology for Web Cache Deception Vulnerability Discovery

Filippo Berto[a], Francesco Minetti[b], Claudio A. Ardagna[c] and Marco Anisetti[d]
Department of Computer Science, University of Milan, Milan, Italy

Keywords: Web Cache Deception, Web Cache, Web Security.

Abstract: In recent years, the use of caching techniques in web applications has increased significantly, in line with their expanding user base. The logic of web caches is closely tied to the application logic, and misconfigurations can lead to security risks, including the unauthorized access of private information and session hijacking. In this study, we examine Web Cache Deception as a technique for attacking web applications. We develop a solution for discovering vulnerabilities that expands upon and encompasses prior research in the field. We conducted an experimental evaluation of the attack's efficacy against real-world targets, and present a new attack vector via web-client-based email services.

1 INTRODUCTION

Content distribution is a common problem in modern web applications, as there are rapidly increasing numbers of users who access the same resources. In the recent years, many software products have integrated web caching technologies and services to enhance the performance of their infrastructure. The increasing popularity of these techniques has prompted researchers to investigate the associated security aspects, leading to the identification of a new type of attack: Web Cache Deception (WCD) (Gil, 2017; Mirheidari et al., 2022; Mirheidari et al., 2020; Nguyen et al., 2019a)[1]. These attacks exploit vulnerabilities in caching services to exfiltrate information, bypassing access control features and obtaining stored data intended for other users (Mirheidari et al., 2022). Researchers have identified several alternative attacks, revealing a variety of vulnerabilities in enterprise content distribution systems (Mirheidari et al., 2020). Although some solutions have been developed to automatically scan for WCD vulnerabilities, they do not consider all the possible attack vectors. Our contribution with this paper is threefold: i) a novel solution for detecting WCD vulnerabilities capable of covering a wide range of important novel cases affecting modern applications; ii) a new attack vector using web mail client, iii) experimentally evaluate the effectiveness of our solution in real scenarios..

1.1 Motivation and Goals

Existing tools, including the one proposed in (Mirheidari et al., 2022), do not cover a number of relevant cases and are not automated. In most of the cases they do not cover scenarios where there are no caching-related HTTP headers or where there are responses with different content for the same cached resource, such as cases where the vulnerable application uses the Cloudflare email obfuscation. In addition, they do not cover specific cases of certain software product versions, such as the case having advisory CVE–2020–15151[2]. Also rare cases in which the web application caches all the resources that have an HTTP 200 response code are not covered as well as cases in which the classic payloads cannot be used but a simple unique query string must be used. Our solution aims to cover all the above cases providing certain degree of automation. In addition, we propose a novel attack vector for WCD vulnerabilities exploiting web mail clients automatically loading web contents. To the best of our knowledge, this approach has never been discussed before in literature or in public domain resources.

[a] https://orcid.org/0000-0002-2720-608X
[b] https://orcid.org/0009-0007-1272-956X
[c] https://orcid.org/0000-0001-7426-4795
[d] https://orcid.org/0000-0002-5438-9467
[1] Practical Web Cache Attacks: https://portswigger.net/research/practical-web-cache-poisoning

[2] https://nvd.nist.gov/vuln/detail/CVE-2020-15151

Berto, F., Minetti, F., Ardagna, C. and Anisetti, M.
A Methodology for Web Cache Deception Vulnerability Discovery.
DOI: 10.5220/0012692000003711
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 14th International Conference on Cloud Computing and Services Science (CLOSER 2024)*, pages 231-238
ISBN: 978-989-758-701-6; ISSN: 2184-5042
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

1.2 Paper Structure

The rest of the paper is organized as follows: Section 2 analyses the literature related to web cache vulnerabilities and exploitation techniques, later focusing on WCDs. Section 3 describes the background of WCD, explaining how web caches work and how their misconfiguration can be exploited, and the motivation of this paper. Section 4 shows our proposed methodology, describing how our solution works and the new edge cases being covered. Section 5 describes the cases covered by our solution and introduces a novel attack vector. Section 6 contains our experimental validation of the developed solutions against real web services. Finally, in Section 7 we discuss the results obtained and report our conclusions.

2 RELATED WORK

The security of web caches is a well known problem without comprehensive solution yet. It is grounded on the difficulties of finding the correct middle ground between caching every content and none, it is strictly linked to the application logic and securing it is generally a hard task. Initial works on the security of web caches date back to the end of the last century, when the rapid growth in popularity of Internet required systems that could handle large scale distribution of contents (Chankhunthod et al., 1996; Smith et al., 1999).

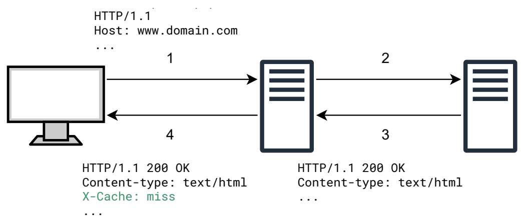
The configuration of web caching services varies according to the requirements of the target web application, as it manages the caching logic for its content. Correctly configuring web caches is error-prone: several considerations must be taken into account depending on the application, such as the type of content returned by the web application, the resource path, and the HTTP code returned. Attacks against web caches usually fall into two classes:

1. **Cache poisoning**, where the attacker sends a crafted request to a vulnerable web cache, forcing it to store a particular version of the server's response. Eventually, a user who sends a request that matches the attacker's caching keys will receive the cached content. This type of attack can be used as a form of Denial of Service (DOS), preventing the user from retrieving the correct content, or even as misinformation, providing the user with outdated information (Ghaznavi et al., 2021; Nguyen et al., 2019b; Nguyen et al., 2019a).
2. **Cache deception**, covered in this paper, which occurs when web caches are incorrectly configured, allowing an attacker to deceive users into storing private content within the cache. The attacker can then retrieve the data using a matching caching key.

In both cases, the misconfiguration of the web cache allows the attacker to generate false positive matches with other users' caching keys.

WCD's literature is still in its infancy, with only few available related publications. Omer Gil was the first describing the attack, showing how misconfigured web caches and Content Distribution Networks (CDNs) may incorrectly store users responses' data, mistaking them for static files such as stylesheets and scripts (Gil, 2017). In this scenario the attacker can then send a key-matching request, retrieving the cached response, possibly containing sensitive information or session data. Mirheidari et al. introduced a WCD search methodology focusing on markers in HTTP responses (Mirheidari et al., 2020). Furthermore, in this article the authors have proposed innovative WCD payloads, characterized by the possession of special characters that block the parsing of the path by the origin application servers, while they are entirely parsed by the intermediary nodes. These payloads have broadened the spectrum of possible WCD scenarios. These techniques have been given the name of *path confusion* as they confuse the origin application server into believing that the requested resource is the correct one, while letting the intermediary nodes carry out the complete parsing of the path, letting them believe that a static resource has been requested, and therefore that it can be saved locally. Mirheidari et al. also implemented a WCD detection tool whose computation is based on the presence and semantics of the HTTP caching headers. They also conducted the largest large-scale experiment in WCD detection. To do so they developed an automation tool that does not require a manual registration phase on the website to be tested. The tool relies on the fact that if a web application mistakenly saves non-authenticated resources in the web cache, it will probably do the same with authenticated resources. This gave them a solution that is well suited to large-scale experiments (Mirheidari et al., 2022). Nguyen et al. conducted a large-scale exploration of commonly used cache systems' security and identified many vulnerabilities that can be attributed to misconfiguration, misinterpretation of standards, and bypassing of security features (Nguyen et al., 2019a).

Several solutions for the security of web caches and CDNs have been adopted in literature. In (Jabiyev et al., 2021), Jabiyev et al. have proposed a fuzzing-based approach to caches vulnerability discovery. Anomaly detection techniques have been proposed to counteract crafted requests (Yang et al., 2022; Ghaz-



```
HTTP/1.1
Host: www.domain.com
...
```

```
HTTP/1.1 200 OK
Content-type: text/html
X-Cache: miss
...
```

```
HTTP/1.1 200 OK
Content-type: text/html
...
```

Figure 1: Web caching mechanism.

navi et al., 2021; Zolfaghari et al., 2020). Others have proposed automated techniques for identifying cache poisoning vulnerabilities (Hildebrand, 2021) or substituted standard HTTP-based caches with ones based on custom network protocols (Lin et al., 2022).

Different network stacks have implemented various approaches to the security of CDNs. For instance, networks based on the Information Centric Networking (ICN) paradigm require producers to sign all content, which mitigates the risk of cache poisoning. Additionally, automated verification solutions have been implemented to verify non-functional properties, such as security and performance (Anisetti et al., 2021; Anisetti et al., 2022). Similar solutions could be applied transparently to traditional web cache services, reducing the risk of misconfiguration.

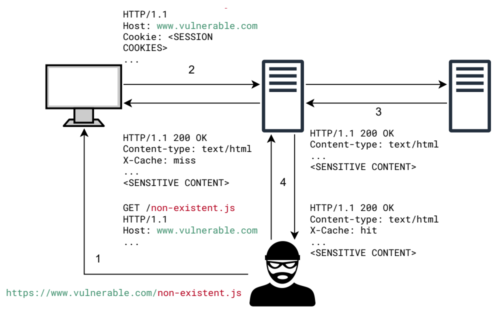
3 BACKGROUND

In this section we summarize the main concepts of web caching and web cache deception vulnerabilities.

3.1 Web Caching

Web caching refers to the process through which HTTP responses are saved by intermediary nodes of a multi-level web infrastructure and then served by them if needed. This process ensures optimization of HTTP traffic, reducing latency and network usage and improving performance. Figure 1 represents the behavior of a generic web caching mechanism.

In steps 1 and 2 a client requests a resource from a server using an HTTP GET request. The request travels from the client to the originating application server, passing through an intermediary node that is providing the caching service. Subsequently, the origin application server replies with an HTTP 200 response, which will travel up to the intermediary node. The intermediary node will now save a copy of the response locally and forward the response to the client. At a later time, if a client requests the same resource, it will be returned directly by the intermediary node, thus eliminating useless traffic between the intermediary node and the originating application



```
HTTP/1.1
Host: www.vulnerable.com
Cookie: <SESSION
COOKIES>
...
```

```
HTTP/1.1 200 OK
Content-type: text/html
X-Cache: miss
...
<SENSITIVE CONTENT>
```

```
GET /non-existent.js
HTTP/1.1
Host: www.vulnerable.com
...
```

```
HTTP/1.1 200 OK
Content-type: text/html
...
<SENSITIVE CONTENT>
```

```
HTTP/1.1 200 OK
Content-type: text/html
X-Cache: hit
...
<SENSITIVE CONTENT>
```

Figure 2: Web cache deception mechanism.

server. In a real environment, the intermediary is an edge server of a CDN or a reverse proxy at the edge of a Demilitarized zone (DMZ). HTTP response codes and headers may differ from the standard ones shown in Figure 1. Resources saved in web caches are identified through configuration variables called *cache keys* which value is generated using various parts of HTTP requests and responses related to a given resource. Depending on the deployment configuration this duty of continuously managing the keys handling mechanism is accomplished either by the organization's or the CDN provider's system administrators, as complete automation often leads to false positives.

3.2 Web Cache Deception

Web cache deceptions are vulnerabilities that arise from an incorrect configuration of the web caching mechanisms. Figure 2 shows a diagram of the WCD attack mechanism.

In step 1 the attacker tricks the victim user into sending a request to a crafted URL. This can be achieved through Cross Site Scripting (XSS) (Cui et al., 2020; Liu et al., 2019) or phishing (Barron et al., 2021; Gupta et al., 2016) or other common attack vectors. The URL consists of a first part, indicated in green in Figure 2, which points to a resource that exists on the application server, concatenated to a second part, the WCD payload indicated in red, which points to a non-existent static resource on the server. In step 2, the victim, authenticated on the vulnerable site, opens the malicious link and sends an HTTP GET request with session cookies to the origin application server. In step 3, after the server has received the HTTP request, it will reply with an HTTP 200 response containing the victim user's private information. The response will travel up to the intermediate node, which will save the response locally, detecting the end of the URL path as a static file due to its misconfiguration. Finally, the content is forward it to the client. As last step, the attacker, requests the same resource with the same cache keys, fetching Personally

Identifiable Information (PII) of the victim user.

WCD vulnerabilities have the following characteristics:

- require a simple HTTP GET request with session cookies. Other web application vulnerabilities usually have stricter requirements, e.g. Reflected XSS vulnerabilities described in (Cui et al., 2020; Shrivastava et al., 2016) where the response must also be interpreted by the browser;
- are widespread, as many web infrastructures today use one or more web caching mechanisms internally;
- have impact that varies according to the type of content mistakenly saved in web caches;
- have a finite attack surface, given by the combination of the set of resources returned by the application server that contain private user information with the set of WCD payloads.

Given the above characteristics, the process of searching for these vulnerabilities can be partially automated.

4 METHODOLOGY

In this section we describe our scanning methodology, which is subdivided into 3 phases.

Registration. In the first phase, we register a new user on the target application. Normally this step requires manual intervention by the researcher, inputting specific markers in the registration that will later be searched by the program. Then the researcher logs onto the target website and copies the session cookies to the scanning tool using the provided browser extension.

Crawling. During the second phase, our solution will perform an authenticated recursive crawling of the domain using the provided cookies and a headless browser. We found this to provide the most effective results even with single page applications. The tool will save all the links it finds within anchor elements and forms, creating a representation of the application attack surface. Algorithm 1 describes the steps of the crawling phase.

Detection. The third and final phase, focuses on detection of WCD vulnerabilities using the collected URLs. For each collected link and for each WCD payload, the tool performs the following actions:

```
def getPageLinks(url, depth: int = 0)
    if isInLinkList(url) and
      isAmplitudeAllowed(url) then
        save(url);
        if depth < MAX_DEPTH and
          totalLinks < MAX_LINKS then
            hrefs ← getAllAnchorHref();
            actions ← getAllFormAction();
            foreach href ∈ hrefs do
                getPageLinks(link, depth+1);
            end
            foreach action ∈ actions do
                getPageLinks(action,
                  depth+1);
            end
```

Algorithm 1: Crawling phase algorithm.

- send an HTTP GET request with session cookies to the given link concatenated with the WCD payload;
- send the same previous request, but without the session cookies;
- finally, check if there are cookies, markers or Cross Site Request Forgery (CSRF) tokens of the victim user in the response to the unauthenticated request and, if so, it will warn the user that it has found a possible WCD.

Algorithm 2 describes the steps of the detection phase.

```
def detection(payloadList)
    foreach p ∈ payloadList do
        authResponse ←
          sendAuthRequest(p);
        unauthResponse ←
          sendUnauthRequest(p);
        if isAuthContent(unauthResponse)
          then
            possible WCD found
    end
```

Algorithm 2: Detection phase algorithm.

We note that, although not implemented in our solution, the first phase could also be fully automated by programmatically recognizing *sign up* and *sign in* forms and inserting the appropriate tokens and credentials. We also note that some sites require two-factor authentication or obstruct robots and scrapers with detection and prevention techniques, such as captchas, often placed precisely in conjunction with the sign-in and sign-up forms. It has recently been demonstrated how it is possible to bypass these anti-bot puzzles in an automated way with the help of neural networks (Mirheidari et al., 2022; Ma et al., 2020).

5 COVERED CASES

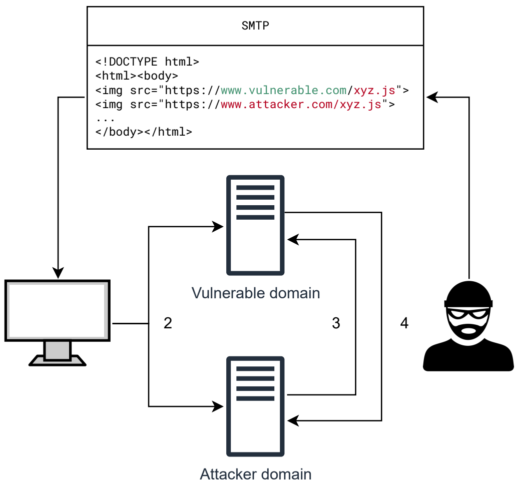
Our solution covers the classic cases with and without HTTP headers related to caching, including path confusion techniques. Furthermore, it covers cases in which it receives responses where the body differs for the same resource saved in the web cache, e.g. in web applications that use Cloudflare email obfuscation.

Our solution also covers a specific case of certain versions of the Content Management System (CMS) OpenMage LTS, based on Magento, with advisory CVE–2020–15151. For certain versions of this CMS, the default installation includes a WCD-vulnerable web cache local to the application server. Specifically, with the default configuration, all 404 responses related to a request with the path ending with a static extension are saved in the cache. The problem is that these 404 responses contain the CSRF token of the victim user who requested it. As a result, an attacker could steal the victim's CSRF token and, if the same-site attributes of the cookies allow it, it could perpetrate a CSRF attack. These versions of the CMS require a cookie called X-Magento-Vary[3] with a precise value (and the same for all users) to access the web cache. Our tool detects whether the X-Magento-Vary cookie is present in the session cookies, saves its value and subsequently for each HTTP request made, the cookie in question will be included in the requests cookies. This case, like the others mentioned previously, could not have been identified using current tools.

5.1 Novel Attack Vector: Web Mail Client

We identified web mail clients as possible vectors for WCD-type attacks. The hypothesis is that web mail clients can, under certain conditions, send HTTP GET requests with session cookies while loading email contents. At the time of writing, we are unaware of any previous discussion in literature or public domain resources on the topic.

The attack is summarized in Figure 3. In step 1, an attacker sends an email with two images in the body to the victim's email address. The first image will have the `src` attribute with a URL dedicated to the WCD attack, while the second image will point



```
<!DOCTYPE html>
<html><body>
<img src="https://www.vulnerable.com/xyz.js">
<img src="https://www.attacker.com/xyz.js">
...
</body></html>
```

Figure 3: Web mail client as attack vector.

to a server owned by the attacker. When the victim user opens the malicious mail in a web mail client that does not filter third-party content in the body of the mail, two HTTP GET requests are sent. The first request will initiate the WCD attack, and second request will notify the attacker's server that the WCD attack has started and with what payload value it was performed. At this point the attacker's server will be able to request the resource erroneously stored in the cache and steal the victim's private information. It should be noted that throughout this process the interaction of the victim user is almost non-existent, as they will only have to view an email, without clicking any link. We have experimentally demonstrated that in order to perpetrate the attack just described, 3 conditions must be valid:

- the site vulnerable to WCD must have session cookies with the same-site attribute set to none and the secure flag set to true;
- the web mail client must not filter in the body of the emails content that could generate GET requests to third-party sites;
- the victim user must be using the Chrome browser, which as of this writing has not yet implemented state partitioning.

Moreover, we have demonstrated how before the insertion of the same-site attribute of cookies in browsers (before mid-2020) it was possible to perpetrate the attack with any site vulnerable to WCD and with any browser. This was demonstrated by testing the attack with a mid-2020 standalone release of the Firefox browser and a popular web mail client that does not filter third-party content in the body of emails. Furthermore, flaws in the filtering of content in the body of emails that can generate HTTP GET

[3] Magento's default caching policies: https://devdocs.magento.com/guides/v2.4/extension-dev-guide/cache/page-caching/public-content.html

requests can be exploited by an attacker to use web mail clients that normally could not be used as an attack vector. This experiment has shown how with the advent of new types of vulnerabilities a privacy problem, such as filtering third-party content in the body of emails, can also become a security problem.

6 EXPERIMENTS

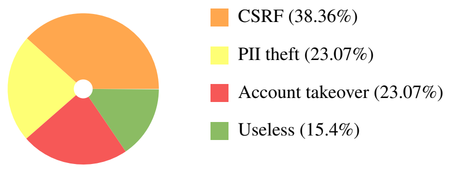
This section describes the experiments carried out to validate the efficacy of the methodology, focusing on detection of WCD vulnerabilities, their classification and exploitation of the vulnerable target. Following, additional experiments on the exploitation of web mail clients as vectors for WCD attacks.

6.1 Detecting WCD

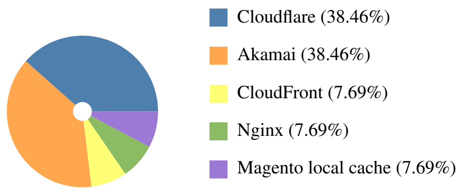
The developed tool has been tested on 100 domains, chosen from organizations that explicitly allow security testing, e.g. by providing bug bounty programs. Furthermore, these domains have been selected in such a way that they all have private content returned directly in HTML by the application server. Of these 100 domains, 13 were affected by WCD. Figures 4a, 4b and 4c show pie charts plots representing respectively the consequences of the WCDs vulnerability, the web caching technologies found to be misconfigured and the HTTP response codes of the resources erroneously saved in the web caches.

As shown in Figure 4a, the impact of the WCDs found is variable and ranges from a CSRF attacks, allowing changes to the personal information of the victim account, to the theft of the victim account in the worst case. Figure 4b shows how most of the web caching software found vulnerable belonged to CDNs (Cloudflare, Akamai, CloudFront) while a smaller part to reverse proxies (Nginx, Magento). Finally, Figure 4c shows that the HTTP response codes of resources erroneously saved in the cache are mainly 200 and 404. In smaller numbers, cases with responses erroneously saved in the cache with HTTP 410 response code were also detected.

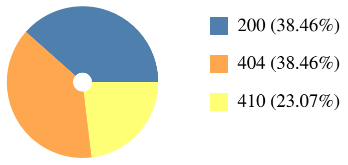
A separate analysis was carried out regarding CVE–2020–15151. First, 15 domains using the vulnerable version of OpenMage LTS were identified, these domains were chosen from a different pool of domains from those mentioned in the previous experiment. In all 15 domains the default configuration was active and all were vulnerable to WCD, allowing exfiltration of the session token. Of these 15 domains, 5 did not have the same-site session cookie attribute set, allowing an attacker to perpetrate a CSRF attack.



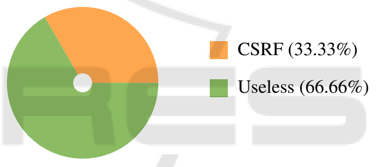
(a) WCD consequences.



(b) Misconfigured technologies.



(c) Cached HTTP response codes.



(d) OpenMage LTS WCD.

Figure 4: WCD detection experimental results.

The other domains had the session cookies' same-site attribute set to LAX or STRICT, thus preventing the attack. Figure 4d contains a pie chart plot representing the consequences of the WCDs found during this analysis.

The same domains were also tested manually in order to verify the effectiveness of the tool. We identified false negative cases, where the tool could not detect the vulnerability as it was blocked by anti-bot software checking the JavaScript `navigator.webdriver` property, thus identifying the Selenium driver and halting the crawling phase. We also noticed false positives cases in which cookies with unimportant values, such as the domain name or domain URLs, were returned by the application server, misleading the tool into detecting a possible WCD with cookie exfiltration. The issue has been corrected by filtering the returned cookies keeping only the ones with more than 16 characters and which differ in the initial part from a variant of the domain name (such as `domain.com`, `http://domain.com`, `https://domain.com`).

Following the detection of WCDs vulnerabilities in authenticated mode, the tool was run on a list of 750 domains in unauthenticated mode. In this execution mode, the tool only checked for the presence of CSRF tokens that were erroneously stored in the web cache. This experiment is based on the hypothesis that if the tool detects the presence of CSRF tokens mistakenly stored in the web cache for a particular anonymous session, it will most likely do the same for authenticated users. Of these 750 domains, only 3 were found to be vulnerable. This lower number of positive cases is mainly due to the fact that the research targeted a specific subset of possible WCDs cases. In addition, the tool uses simple regular expressions to match against CSRF tokens, therefore it is likely that some were missed. We have found that for large-scale analysis of the unauthenticated type, it is less computationally expensive and more effective to use an approach based on the analysis of the HTTP caching headers. We note that all the experiments conducted in this study solely targeted web applications of organizations that granted explicit permission for security testing. Additionally, the tool developed generated minimal network overhead in comparison to a normal browser, preventing accidental flooding of the target, and is available on our GitHub repository under the Creative Common license[4].

6.1.1 Global Scale Attacks

Various researchers have tackled the issue of exploiting WCDs in a CDN-like environment in a globally scalable way. The limit they encountered, from the attacker's point of view, was that of being able to retrieve the resources that were initially saved in a precise edge server in the globe, without knowing in advance in which geographical region the victim is located. We found that the attacker can easily overcome the issue by inducing the user in connecting to one of their servers using attacks similar to the ones used for WCD (e.g. XSS or phishing). When the victim user connects to the attacker's server, their IP is collected and the victim is redirect to the URL of the WCD attack. The attacker can then identify the victim location using IP-to-Location services. Alternatively, an attacker could simultaneously send several HTTP requests to each CDN edge server in the victim supposed region.

[4]Source code of the proposed tool: https://github.com/SESARLab/WCD_prober

6.2 Discussion

In terms of defense against WCD vulnerabilities, the primary solution relies on a proper configuration of web caching technologies, preventing the caches from storing private information. This implies specific considerations on the handling of HTTP requests by the application server, even for non-existing endpoints. It is paramount to check both the test and production environments as the version or configuration of the application or the caching technology changes.

Considering the web mail clients as vectors for WCD attacks, a substantial defense level can be achieved by filtering any content in the body of the emails that could automatically generate HTTP GET requests to third-party sites, such as images and styling files. Current widespread behavior of allowing whitelisting of entire (sub)domains could pose a threat in the event of changes to the cache technology configuration. Finally, some vendors, such as Cloudflare, have implemented software products that mitigate WCD vulnerabilities by performing content type checks on HTTP responses, verifying that the *content-type* header matches the one declared in the path, if any, and thus deeming it suitable for being stored.

7 CONCLUSION

In this paper we presented a novel methodology for detecting WCD vulnerabilities, experimentally evaluate its effectiveness covering the largest possible number of WCD cases. Our detection solution demonstrated better reliability for authenticated analyses, compared to the unauthenticated ones. Finally, a novel attack vector for WCDs using web mail client has been proposed and experimentally verified. We also have verified how some privacy-preserving techniques introduced by default in web browsers in mid-2020 have accidentally reduced part of the attack surface of WCDs.

ACKNOWLEDGEMENTS

The work was partially supported by the projects i) MUSA – Multilayered Urban Sustainability Action – project, funded by the European Union – NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.5: Strengthening of research structures and creation of R&D “innovation ecosystems”, set up of “territorial leaders in R&D” (CUP

G43C22001370007, Code ECS00000037); ii) SERICS (PE00000014) under the NRRP MUR program funded by the EU – NextGenerationEU; iii) 1H-HUB and SOV-EDGE-HUB funded by Università degli Studi di Milano – PSR 2021/2022 – GSA – Linea 6; and iv) program “piano sostegno alla ricerca” funded by Università degli Studi di Milano.

REFERENCES

Anisetti, M., Ardagna, C. A., Berto, F., and Damiani, E. (2021). Security Certification Scheme for Content-centric Networks. In *2021 IEEE International Conference on Services Computing (SCC)*, pages 203–212, Chicago, IL, USA. IEEE.

Anisetti, M., Ardagna, C. A., Berto, F., and Damiani, E. (2022). A Security Certification Scheme for Information-Centric Networks. *IEEE Trans. Netw. Serv. Manage.*, 19(3):2397–2408.

Barron, T., So, J., and Nikiforakis, N. (2021). Click This, Not That: Extending Web Authentication with Deception. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS ’21, pages 462–474, New York, NY, USA. Association for Computing Machinery.

Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., and Worrell, K. J. (1996). A Hierarchical Internet Object Cache. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, volume 164 of *ATEC ’96*, page 13, USA. USENIX Association.

Cui, Y., Cui, J., and Hu, J. (2020). A Survey on XSS Attack Detection and Prevention in Web Applications. In *Proceedings of the 2020 12th International Conference on Machine Learning and Computing*, ICMLC ’20, pages 443–449, New York, NY, USA. Association for Computing Machinery.

Ghaznavi, M., Jalalpour, E., Salahuddin, M. A., Boutaba, R., Migault, D., and Preda, S. (2021). Content Delivery Network Security: A Survey. *IEEE Communications Surveys & Tutorials*, 23(4):2166–2190. Conference Name: IEEE Communications Surveys & Tutorials.

Gil, O. (2017). Web Cache Deception Attack. In *Proceedings of Black Hat 2017 US*.

Gupta, S., Singhal, A., and Kapoor, A. (2016). A literature survey on social engineering attacks: Phishing attack. In *Proceedings of 2016 International Conference on Computing, Communication and Automation (ICCCA)*, pages 537–540.

Hildebrand, M. (2021). *Automated Scanning for Web Cache Poisoning Vulnerabilities*. PhD thesis, Technische Universität Dortmund.

Jabiyev, B., Sprecher, S., Onarlioglu, K., and Kirda, E. (2021). T-Reqs: HTTP Request Smuggling with Differential Fuzzing. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’21, pages 1805–1820, New York, NY, USA. Association for Computing Machinery.

Lin, S., Xin, R., Goel, A., and Yang, X. (2022). InviCloak: An End-to-End Approach to Privacy and Performance in Web Content Distribution. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’22, pages 1947–1961, New York, NY, USA. Association for Computing Machinery.

Liu, M., Zhang, B., Chen, W., and Zhang, X. (2019). A Survey of Exploitation and Detection Methods of XSS Vulnerabilities. *IEEE Access*, 7:182004–182016. Conference Name: IEEE Access.

Ma, Y., Zhong, G., Liu, W., Sun, J., and Huang, K. (2020). Neural CAPTCHA networks. *Applied Soft Computing*, 97:106769.

Mirheidari, S. A., Arshad, S., Onarlioglu, K., Crispo, B., Kirda, E., and Robertson, W. (2020). Cached and Confused: Web Cache Deception in the Wild. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*, pages 665–682.

Mirheidari, S. A., Golinelli, M., Onarlioglu, K., Kirda, E., and Crispo, B. (2022). Web Cache Deception Escalates! In *Proceedings of the 31st USENIX Security Symposium (USENIX Security 22)*, pages 179–196.

Nguyen, H. V., Iacono, L. L., and Federrath, H. (2019a). Mind the cache: large-scale explorative study of web caching. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC ’19, pages 2497–2506, New York, NY, USA. Association for Computing Machinery.

Nguyen, H. V., Iacono, L. L., and Federrath, H. (2019b). 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, pages 1915–1936, New York, NY, USA. Association for Computing Machinery.

Shrivastava, A., Choudhary, S., and Kumar, A. (2016). XSS vulnerability assessment and prevention in web application. In *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, pages 850–853.

Smith, J., Calvert, K., Murphy, S., Orman, H., and Peterson, L. (1999). Activating networks: a progress report. *Computer*, 32(4):32–41. Conference Name: Computer.

Yang, L., Moubayed, A., Shami, A., Heidari, P., Boukhtouta, A., Larabi, A., Brunner, R., Preda, S., and Migault, D. (2022). Multi-Perspective Content Delivery Networks Security Framework Using Optimized Unsupervised Anomaly Detection. *IEEE Transactions on Network and Service Management*, 19(1):686–705. Conference Name: IEEE Transactions on Network and Service Management.

Zolfaghari, B., Srivastava, G., Roy, S., Nemati, H. R., Afghah, F., Koshiba, T., Razi, A., Bibak, K., Mitra, P., and Rai, B. K. (2020). Content Delivery Networks: State of the Art, Trends, and Future Roadmap. *ACM Comput. Surv.*, 53(2):34:1–34:34.