Cross-Origin Web Attacks via HTTP/2 Server Push and Signed HTTP Exchange

Pinji Chen*, Jianjun Chen*†‡, Mingming Zhang†, Qi Wang*, Yiming Zhang*, Mingwei Xu*, Haixin Duan*
*Tsinghua University, †Zhongguancun Laboratory
{cpj24, qi-wang23}@mails.tsinghua.edu.cn, {jianjun, zhangyiming, xumw, duanhx}@tsinghua.edu.cn,
zhangmm@mail.zgclab.edu.cn

***Abstract*—In this paper, we investigate the security implications of HTTP/2 server push and signed HTTP exchange (SXG) on the Same-Origin Policy (SOP), a fundamental web security mechanism designed to prevent cross-origin attacks. We identify a vulnerability introduced by these features, where the traditional strict SOP origin based on URI is undermined by a more permissive HTTP/2 authority based on the SubjectAlternativeName (SAN) list in the TLS certificate. This relaxation of origin constraints, coupled with the prevalent use of shared certificates among unrelated domains, poses significant security risks, allowing attackers to bypass SOP protections. We introduce two novel attack vectors, CrossPUSH and CrossSXG, which enable an off-path attacker to execute a wide range of cross-origin web attacks, including arbitrary cross-site scripting (XSS), cookie manipulation, and malicious file downloads, across all domains listed in a shared certificate. Our investigation reveals the practicality and prevalence of these threats, with our measurements uncovering vulnerabilities in widely-used web browsers such as Chrome and Edge, and notable websites including Microsoft. We responsibly disclose our findings to affected vendors and receive acknowledgments from Huawei, Baidu, Microsoft, etc.**

I. Introduction

The Same-Origin Policy (SOP) is a cornerstone of web security, designed to prevent malicious scripts on one website from accessing data on another, thereby safeguarding user data against cross-origin attacks. Despite its foundational role, we find that recent advancements in HTTP protocol, specifically HTTP/2 server push and signed HTTP exchange (SXG), pose challenges to this web security paradigm. These new HTTP features introduce novel security concerns that could potentially allow the circumvention of SOP, undermining the security of web content across domains.

At the core of this vulnerability lies two critical issues. First, the notion of "authority" in HTTP/2 and HTTP/3 is looser compared to the "origin" definition of the browser's SOP. Browser's SOP defines the origin of a resource as the URI scheme/host/port tuple [1], while RFC 9110 [2] states that in HTTP/2 and HTTP/3, any domain listed in the SubjectAlternativeName (SAN) of a TLS certificate is recognized as the authority. Second, HTTP/2 server push and SXG both allow to specify the origin of resources that they deliver. Combining these two characteristics, a domain can spoof the origin of another domain within the SAN list of TLS certificates to deliver resources and browsers will accept them as legitimate same-origin resources. This shift from a traditional strict URI-based origin to a more permissive SAN list-based origin could weaken the traditional boundaries of browser origin security.

Even worse, the situation is exacerbated when this permissive boundary meets the problematic ecosystem of the shared certificate, whose SAN list contains multiple domains. Previous work has shown that multiple-domain shared certificates are common and these domains are often managed by disparate entities [3]. It is common on today's web to connect to a website whose certificate is shared with an attacker-controlled domain [4]. Such shared certificates create an avenue for attackers to exploit these features for malicious purposes.

In this paper, we propose two novel cross-origin web attacks via server push and SXG (CrossPUSH and CrossSXG). These techniques enable an off-path attacker with access to a shared certificate to execute a range of web attacks, such as arbitrary XSS, cookie manipulation, and malicious file downloads, across all domains listed in the certificate. These attacks are particularly severe because they remain effective even against victim websites that adhere to HTTPS best practices and implement strict SOP security measures like Content Security Policy (CSP).

We further introduce several strategies to demonstrate the practicality and impact of CrossPUSH and CrossSXG attacks. First, we demonstrate that acquiring the attack condition (i.e., shared certificates) is feasible in practice via simple vulnerabilities, such as by domain reselling or domain takeover. Second, we explore practical exploitation using validation reuse to extend attack duration and the Add Grace Period (AGP) to reduce attack cost. Furthermore, we introduce a technique to make illegitimately acquired shared certificates difficult to revoke by legitimate domain owners, ensuring the persistence of the attack.

To evaluate the real-world impact of these two attacks, we conduct both client-side and server-side measurement studies. Our client-side measurement results show that: (1) 11 out of 14 popular browsers, including Chrome and Edge, are

‡Corresponding author.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.231086
www.ndss-symposium.org

vulnerable to at least one of our attacks; (2) Numerous popular mobile applications, including Instagram and WeChat, are also affected by our attacks; (3) Some vulnerable libraries, such as Chrome-Net, can expose browsers to our attacks. Our server-side measurement results show that: (1) More than 10K Tranco Top 1M domains, including many corporate websites, have been resold and potentially exposed to our attacks; (2) About 5K dangling domains, including a subdomain of *windowsupdate.com*, can be exploited for our cross-origin web attacks; (3) About 85% of Tranco Top 1K domains are included in the shared certificates of domains ranked out of Tranco Top 1M. That risky dependency allows attackers to attack the top websites via less secure sites. We have responsibly reported our findings to the affected providers and received acknowledgments from Huawei, Baidu, Microsoft, etc.

Contributions. In this paper, we make the following contributions.

- We present CrossPUSH and CrossSXG attacks that broadly threaten web security. We demonstrate this threat is practical and severe, enabling off-path attackers to conduct a range of web attacks by exploiting HTTP/2 server push and SXG, thus circumventing SOP protections.
- We conduct comprehensive measurements to evaluate the real-world impact of the attacks, revealing vulnerabilities across numerous popular browsers and websites.
- We propose four approaches to mitigate CrossPUSH and CrossSXG attacks. We have also responsibly disclosed issues to affected vendors and received positive feedback.

II. Background

A. HTTP/2 Server Push

Server push is one of the new characteristics of HTTP/2, which was first defined in RFC 7540 [5]. Without server push, traditional website access follows the request-then-response pattern that a client initially requests for an HTML document, and then the web server replies with the assets. Afterward, the browser parses the HTML and discovers other referenced assets, such as scripts, images, and style sheets. Due to the discoveries, the browser needs to request these assets again, which takes at least two round-trip times (RTT) to get critical resources. Before the browser gets all of the resources, the user can merely see a blank page on his screen. Server push solves this problem more efficiently. When the server responds with HTML, it can also proactively push what the users are going to need for the requested pages. Consequently, pushed assets prevent the browser from requesting resources again, the whole process only costs one RTT to load the contents if the server preemptively sends all critical assets, which greatly improves the performance of website access. Specified in HTTP/2 by the IETF working group in 2015, server push has been going on ever since and continued to be kept in HTTP/3 [6] in 2022.

HTTP/2 server authority. In HTTP/2 and HTTP/3, a client attributes authority to a server if the URI origin's host matches any of the hosts present in the server's certificate [2]. Although server authority can be lots of origins in the certificate, the client always attributes server authority to the current requesting URI tuple. However, the HTTP/2 server push is unique here. It can indicate the authority of a push stream by specifying `:authority` pseudo-header. Moreover, the browser can accept the push stream as long as its `:authority` matches any origins in the certificate according to the specification of RFC 9113 [7].

B. Signed HTTP Exchange

Signed HTTP exchange (SXG) is a delivery mechanism that makes the browser authenticate the origin and integrity of a resource without considering how it was delivered [8]. With the SXG, a publisher can safely make their content portable, i.e., available for redistribution by third parties, while still keeping the content's integrity and attribution. The most common practice of SXG is to prefetch and serve the contents signed by the publisher through a cache. This boosts cross-origin navigations from referer sites while also ensuring the untampered contents as well as the proper attribution. For instance, Google Search crawls and caches SXGs when available and prefetches SXG that the user is likely to visit, e.g., the first search result, to provide users with a faster page load experience from the search results page. In 2018, Google announced the support of SXG in Chrome 73 and later [9]. Since then, many other Chromium-based browsers have begun to be available for SXG as well, e.g., Edge 79 and Opera 64 [10]. Some Content Delivery Network (CDN) vendors can also configure SXG service [11].

Origin of SXG. Browser attributes the origin of an SXG to `request-url` header as long as this URL is same-origin with `validity-url` and the certificate in `cert-url` can authenticate `validity-url`. Similar to server push, SXG providers can also indicate the origin of a resource to any domain in the certificate by specifying these headers.

C. Shared Certificate

SSL/TLS certificate is the digital identity card of a website. In an HTTPS connection, one party can ensure the identity of another party via the certificate issued by the trusted certificate authorities (CAs), reducing the risks of man-in-the-middle attacks. Nevertheless, it is inconvenient if one certificate can merely authenticate one domain. Some organizations have plenty of domains, which is a heavy burden to issue certificates for each one. To issue, manage, and revoke an SSL/TLS certificate more efficiently, the certificate can be shared by multiple subjects. Specifically, a shared certificate uses the SAN extension to include multiple alternate domains, and each of these domains can be authenticated by the certificate. For example, a certificate with a SAN list [**.google.com*, *android.com*] would be accepted by both *www.google.com* and *android.com*.

Shared certificate issuance. The most crucial step in issuing a certificate is domain ownership validation (DOV). When issuing a shared certificate with multiple domains, CAs need to verify the ownership of every domain. There are three ways to perform DOV: (1) DNS mode. Add appointed DNS records

to the authoritative nameservers. This is the only case to issue the certificate of wildcard domains; (2) HTTP mode. Write a given file to the root directory of the website, e.g., */.well-known/pki/*; (3) Email mode. Click the received verification link in the website administrator's email. Besides, issuing a certificate that can sign an SXG additionally needs the CA to support the CanSignHttpExchanges extension. To the best of our knowledge, only two CAs (*Digicert* and *Google*) are now available for the SXG certificate and ask the domains to configure the DNS CAA records[1].

Certificate expiration. The expiration time is important for both attackers and owners because it is the final backstop of the compromised certificates. Although the CA/Browser forum suggests that setting the lifespans of certificates to no more than 398 days offers several benefits [12], the specific validity period depends on issuers and the implementations of CAs. In addition, the draft of SXG limits the lifespan of the certificate with CanSignHttpExchanges extension to 90 days [13].

III. CrossPUSH and CrossSXG Attack

This paper introduces a novel vulnerability that adversaries can utilize shared TLS certificates in server delivery processes to bypass SOP. The threat is brought by two server delivery mechanisms, HTTP/2 server push and SXG, and we refer to them as *CrossPUSH* and *CrossSXG* attacks, respectively. In this section, we first present the threat model (Section III-A) and concept (Section III-B). Then, we delve into the novel security implications (Section III-C) and methodology (Section III-D) of these attacks.

A. Threat Model

In this paper, we assume an off-path adversary with two capabilities: *First, an attacker can acquire a TLS certificate shared with a victim's websites.* This requirement is feasible in practice via common vulnerabilities such as unsecured file uploads in the "*/.well-known*" directory [14] or an email provider's oversight in protecting the domain's administrative email addresses [15]. Besides, we propose two practical methods for attackers without any intrusion abilities to acquire multi-domain certificates shared with victim's websites in our study: (1) Domain reselling. An attacker can register numerous domains and issue a shared certificate for these domains, then resell domains to victims while retaining the certificate (Section IV-A Method-1); (2) Domain takeover. An attacker can take over the dangling domains whose DNS records pointed to discontinued CDN services, de-provisioned VPS IPs, or expired domain names, and then issue a shared certificate (Section IV-A Method-2). *Second, the attacker can lure the victim to visit the attacker's website.* This can be accomplished through methods such as phishing or embedding an iframe on a popular site. Notably, we do not require the attacker to be in-path or on-path to intercept and sniff encrypted traffic.

For the victims, all users visiting attackers' websites with vulnerable browsers (Chrome, Edge, and others in Section VI-A3) could be affected. They will be susceptible to arbitrary XSS, cookie manipulation, and other attacks in the context of the websites that are included in the attacker's shared certificate.

[1]https://github.com/google/webpackager/wiki/Certificate-Authorities

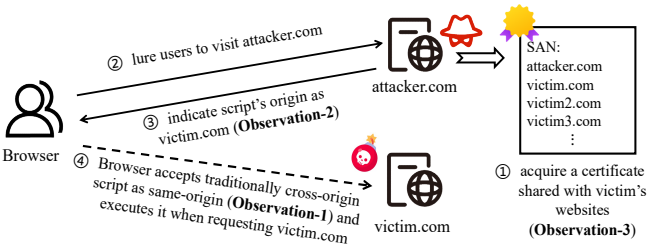


Fig. 1: Attack concept of CrossPUSH and CrossSXG.

B. Attack Concept

CrossPUSH and CrossSXG attacks take advantage of three key observations to bypass browser SOP protections, thus enabling various cross-origin web attacks.

Observation-1: HTTP/2 server authority is less restrictive than the browser's same-origin policy. While both policies indicate the resource's origin, browsers define the same origin based on identical URI tuples, whereas HTTP/2 considers any domains listed in the SAN of the TLS certificate to be from the same authority. Therefore, this discrepancy could compromise the browser's SOP protection when the browser's origin is reassigned to match the HTTP/2 authority.

Observation-2: Server push and SXG can specify the authority of the resource in HTTP responses. Typically, the authority of an HTTP response aligns with the request's URI. However, server push and SXG, two new features in the server delivery process, can utilize the `:authority` pseudo-header and `request-url` to reassign the resource's authority to any domains listed in the certificate's SAN. Thus, a domain can spoof the origin of another domain within the SAN list of shared certificates to deliver resources and browsers will accept them as legitimate resources under the target domain.

Observation-3: Misalignment between certificate and domain ownership exacerbates attacks. Previous work has shown that multiple-domain shared certificates are common and these domains are often managed by disparate entities [3]. In addition, there is no coercive measure to keep the certificate and domain owner in line. Attackers can exploit common web vulnerabilities or scenarios presented in Section IV-A to acquire a certificate shared with others' domains, thereby launching our attacks to numerous victim websites.

As illustrated in Figure 1, based on the above observations, the attack flow for both CrossPUSH and CrossSXG can be summarized into four steps: (1) the attacker acquires a shared certificate containing both attacker-controlled domains (e.g., *attacker.com*) and victim domains (e.g., *victim.com*), as detailed in Section IV-A; (2) the attacker can set up an HTTP/2 server or an SXG server on *attacker.com* using the shared certificate, and lure users to visit; (3) the attacker delivers a malicious cross-origin resource (e.g., a malicious script) via server push or SXG, specifying the origin as *victim.com*;

(4) browsers check the authority of the attacker's scripts and recognize them as resources from *victim.com*. Consequently, when users revisit *victim.com*, their browsers execute the malicious scripts within the context of the *victim.com*.

C. Novel Security Implication

Notably, our attacks introduce novel security concerns compared to previous works, as outlined below.

Implication-1: Enabling more practical web attacks with illegitimate certificates. Previously, acquiring a certificate would only facilitate man-in-the-middle (MitM) attacks, which requires the attacker to be on-path, but now it is straightforward for an off-path attacker to execute web attacks via server push and SXG, making it much easier to launch an actual attack against users.

Implication-2: Increasing the attack surface of high-profile sites. For instance, if an adversary wants to attack *google.com*, he might typically look for vulnerabilities on this strictly protected site. However, our attacks expose more attack surfaces by enabling the adversary to execute various web attacks against *google.com* by compromising another domain, such as *admob-cn.com* or a subdomain, that shares certificates with *google.com*. Many of these domains are low-ranked websites, which, as previous studies [16] have shown, respond more slowly to security events like Heartbleed, thereby exposing more risks of certificate breaches to enable our attacks.

Implication-3: Extending the attack duration time of some short-lived vulnerabilities. Previous vulnerabilities like dangling domain takeover will be invalid after the victim deletes the stale DNS record (from hours to days). In contrast, our attacks extend the attack duration to the certificate expiration (796 days after removing the dangling records as we demonstrate in Section IV-B). An attacker can issue a shared certificate of dangling domains, and then leverage CrossPUSH to continuously attack the dangling domains for more than two years. Further, we also find a technique (Section IV-D) to make victims difficult to revoke the acquired certificate themselves which results in a persistent attack.

D. Attack Methodology

CrossPUSH and CrossSXG attacks exhibit fundamental similarities but differ in execution details. Below, we elaborate on the specific methodologies of each attack. Our threat model assumes that an attacker has already obtained a certificate sharing between *attacker.com* and *victim.com* and we would demonstrate how an attacker can acquire the shared certificate in Section IV-A.

CrossPUSH attack. Typically, browsers accept resources that are pushed from the same origin as the requests. The authority of the pushing server is determined by verifying the SAN domains in the SSL/TLS certificates during TLS connections. Furthermore, browsers assess the `:authority` pseudo-header in HTTP responses to confirm the authenticity of the resources. However, a malicious server could push resources for other domain names sharing the same TLS certificates by crafting a forged `:authority` header.

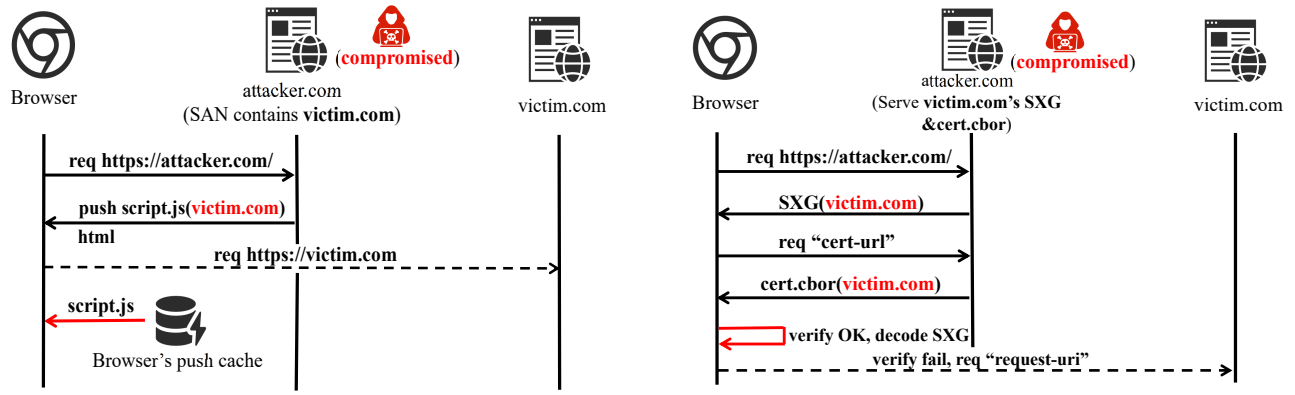
Key entities. Figure 2(a) presents the *CrossPUSH* attack model, which comprises three key components: (1) A browser equipped with a push cache; (2) An attacker in control of a website, *attacker.com*, sharing a TLS certificate with victim websites; (3) A victim website *victim.com*.

Attack details. To conduct a *CrossPUSH* attack, the attacker first tricks users into visiting *https://attacker.com*, possibly via phishing or embedding an iframe on a popular site. Then, while sending a standard HTML response, the attacker uses an HTTP/2 server push to send script.js, claiming its authority as *victim.com*. This could be done by using Node.js's HTTP/2 framework to directly set the stream's `:authority` [17]. Since the authority of script.js (*victim.com*) aligns with the origin specified in the shared certificate but does not match the requested host (*attacker.com*), the browser accepts and stores it in the push cache but does not apply it to the current page. Afterward, the attacker can use the HTML `<meta>` `http-equiv` attribute in the response stream to make the browser request *victim.com*, leading the attack to take effect. The dotted line in Figure 2(a) indicates that the following request will not arrive at *victim.com* since the browser will check the push cache first and find that the requesting host matches the authority of script.js. Consequently, via CrossPUSH, the browser loads the script.js from the cache that is actually sent by *attcker.com*. Although HTTP/2 server push may be designed to permit cross-origin pushes, this feature can be exploited by malicious attackers with access to shared certificates to falsely present trusted authority for harmful content, such as malicious scripts, thereby circumventing the browser's SOP protection.

CrossSXG attack. SXG is designed for publishers to distribute their web content via third parties or caches, enabling browsers to securely serve cross-origin resources. Here, we should claim that rather than discuss the aforementioned capability of *serving* others' cross-origin content via SXG, we elaborate on how a website administrator can *sign* the SXGs of other websites. Our threat model reveals a pressing concern, akin to the CrossPUSH scenario: an illegal entity could create and sign malicious web content as SXG, falsely presenting it as originating from any victim domains in the SAN list of a shared TLS certificate. This is achieved by manipulating the `request-url` and `validity-url` headers.

Key entities. Figure 2(b) depicts the *CrossSXG* model, which also comprises three parts: (1) A browser enabled with SXG support; (2) A compromised website, *attacker.com*, serving *victim.com*'s SXG and cert.cbor. Importantly, cert.cbor represents the shared certificate in concise binary object representation (CBOR) format [18], which is essential for verifying the victim's SXG; (3) A victim website *victim.com*.

Attack details. First, the attacker prepares malicious web content signed as an SXG for *victim.com*. Then, they entice users to visit *https://attacker.com* and deliver the SXG to the browser. The browser then requests the `cert-url` from the SXG headers for certificate validation, which, controlled by the attacker, points back to *attacker.com*. This allows the attacker to respond with cert.cbor containing *victim.com*



(a) CrossPUSH attack. The attacker forges the authority of the push stream to victim.com. Since the browser accepts all of the resources that are authorized in the scope of SAN, the browser loads the script.js that is actually sent by attacker.com from the push cache when the user requests victim.com next time.

(b) CrossSXG attack. The attacker forges two headers of the SXG, "request-url" and "validity-url" to victim.com. Since the browsers accept all of the SXGs whose aforementioned headers are same-origin with the domains in the SAN of the certificate, the browser decodes the SXG as the victim's web content but is actually crafted by the attacker.

Fig. 2: Details of CrossPUSH attack and CrossSXG attack

in the SAN. The browser checks if this certificate matches the `validity-url` and whether the `request-url` is the same origin as the `validity-url`. If verified, the browser displays *victim.com* in the address bar and loads the attacker's content. If not, it would typically request *victim.com*. However, as the attacker can tailor the SXG content, the verification usually succeeds, rendering the request to the victim's site impossible, as indicated by a dotted line in the figure. Consequently, the attacker with access to a shared certificate (cert.cbor) can successfully sign an SXG for *victim.com* and execute malicious content on the victim website's context.

IV. Attack Practicality

In this section, we introduce a series of techniques to make these attacks more practical and dangerous in the real world. We propose (1) two common and exploitable cases to reduce the demands on the attacker's capability to acquire shared certificate (Section IV-A); (2) a novel insight on validation reuse to extend attack duration time (Section IV-B); (3) an abuse of AGP to reduce attack cost (Section IV-C); (4) a technique to make illegitimate certificate irrevocable (Section IV-D).

A. Acquiring Shared Certificates without Intrusion

We propose two broadly applicable and exploitable methods that enable even a basic attacker, lacking any intrusion capabilities, to meet the shared certificate condition necessary to initiate our attacks.

Method-1: Domain reselling—issuing a multi-domain certificate and then reselling part of the included domains. Due to the absence of mechanisms aligning certificate ownership with domain ownership, domain reselling allows attackers to retain the shared certificate while reselling domains included in that certificate to another party. Thus, an attacker can obtain a shared certificate through the following steps:

① The attacker purchases many domains (e.g., victim.com, attacker.com) and issues shared certificates for them.
② The attacker popularizes domains included in the shared certificate with many methods to increase the reselling probabilities, such as exploiting blackhat search engine optimization (SEO) platforms [19], promoting websites via social media and domain resale marketplaces, or improving rankings in top domain name lists [20].
③ The attacker waits for these domains to be bought by any potential victims. If a victim acquires the resold domains before the certificate expires, the attacker successfully creates attack conditions.

While domain reselling has become prevalent[2], this resale process not only grants all original domain owners access to the attack conditions but also exposes anyone purchasing the resold domains to our described attacks.

Method-2: Domain takeover—taking over dangling domains and issuing shared certificates. Domain takeovers have been demonstrated as a potent method for hijacking domain names [21], [22]. This occurs when domain names point to inactive cloud services, de-provisioned VPS IPs, or expired domain names. As these targets are publicly available, attackers can seize control of some victim domain names without authority. Previous work [23] shows that once a domain is taken over, attackers can circumvent domain ownership validation (DOV) via HTTP mode when obtaining SSL/TLS certificates. This enables them to issue a shared certificate that includes both the hijacked domain and an attacker-controlled domain, setting the stage for our attacks. We illustrate attack requirements for different scenarios to launch domain takeover in Appendix A Table V. The key steps to obtain a shared certificate through domain takeover are shown below:

[2]https://www.name.com/zh-cn/blog/3-easy-ways-to-get-started-reselling-domains

TABLE I: Validation reuse period of celebrated CAs

Digicert	GlobalSign	Entrust	Let's Encrypt	Google
397	397	398	30	90

[1] The certificate with SXG extension has validation reuse too. Although there is no document to expose how long it is, our experiment on Google CA shows that the reuse period lasts at least 30 days.

① The attacker locates the victims (e.g., *victim.com*) by scouting for dangling domains whose DNS records are pointing to inactive cloud services (e.g., CDN services), de-provisioned VPS IPs, or expired domain names.
② For each scenario, the attacker applies for the stale resources (e.g., CDN-assigned endpoints like victim.cdnservice.com) that are pointed by dangling DNS records of *victim.com* and then configures the endpoints to route traffic from *victim.com* to the attacker's server. Consequently, the DNS resolution would be as follows:

victim.com ⇒ *stale resources* ⇒ *attacker's web server*

③ With the dangling DNS records, a CA could connect to the attacker's server to fetch the challenge token during the DOV of victim.com in HTTP mode. This enables the attacker to issue a multi-domain certificate that covers both victim.com and attacker-controlled domains, thereby successfully creating the attack condition.

B. Extending Attack Duration via Domain Validation Reuse

Attack duration time significantly affects the practicality of our attacks. Although the recommended validity period for public TLS certificates has been reduced to less than 398 days [24], or in some cases even 90 days [25], our recent findings indicate that a user-friendly approach, known as domain validation reuse [26], can potentially extend the duration of our attacks to a maximum of 796 days.

Domain validation reuse allows a certificate issuer to bypass DOV by reusing previously validated domain ownership information. This approach, automatically activated after the issuer has successfully passed validation in the recent past, aims to ease the issuer's burden and enhance usability. The CA/Browser Forum has set the maximum period for validation reuse at 398 days [26]. However, individual certificate authorities (CAs) have considerable discretion when implementing this policy. The specific validation reuse periods for several well-known CAs are detailed in Table I.

Nevertheless, it has been overlooked that validation reuse not only extends the certificate's lifespan but also potentially grants the domain owner extended control over the certificate. This aspect significantly increases the potential duration of our proposed attacks. As shown in Figure 3, we assume that the attack time span would be 398 - T days in the absence of validation reuse, where "398" is the certificate lifespan and "T" is the time an attacker waits for a victim to purchase or gain control of the domains included in the shared certificate. However, when employing validation reuse, the

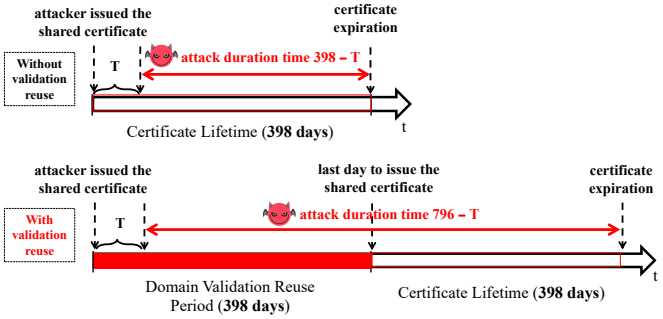


Fig. 3: Extending attack duration time. Validation reuse enables the attacker to prolong the attack window to 796 - T days, where T is the period that the attacker waits for victims to get control of the domains in the shared certificate, e.g., buying the attacker's resold domains.

attacker can issue shared certificates without DOV for another 398 days (solid red bar in Figure 3) following the initial issuance. During this period, the shared certificate remains firmly under the attacker's control. Notably, domain owners even cannot completely revoke the shared certificate, because the attacker can continually reissue it as long as their account remains within the validation reuse period. Moreover, if the attacker reissues a new shared certificate on the last day of the validation reuse period, the duration of the attack can potentially be extended to a maximum of 796 days[3].

Even worse, there is no established security protocol or joint effort between domain registrars and certificate authorities (CAs) in this context. To test its feasibility, we conducted a real-world experiment. We registered a test domain from *Godaddy* and issued a shared certificate for this domain authorized by *Let's Encrypt*. We then released the domain, making it available to other customers. When the domain was available again[4], we acted as a potential victim and purchased the domain through *Namecheap*. We then issued certificates using another account via *ZeroSSL* and *Let's Encrypt*, respectively. Our findings are concerning: (1) Even when two accounts from the same CA (e.g., *Let's Encrypt*) control the same domain simultaneously, this does not disrupt the process of validation reuse; (2) Despite later revoking all issued certificates, the original account can still bypass DOV to reissue certificates due to validation reuse.

C. Reducing Attack Cost via AGP Abuse

In a domain reselling scenario (Method-1 in Section IV-A), an attacker needs to acquire many domain names for attacks. Therefore, minimizing the cost of these domain purchases significantly enhances the attack's viability. We find that Add Grace Period (AGP) [27], a user-friendly feature of domain registration, can be abused by attackers to facilitate this

[3]This is 120 days for CrossSXG attack because the lifespan (90 days) and validation reuse period (30 days) of the certificate with CanSignHttpExchanges are shorter.

[4]The domain releasing time from a registry is tested to be 4 days in our experiment.

process. AGP, supported by ICANN and domain registrars, provides registrants the opportunity to return domain names within 5 days without cost. Two typical cases of abuse are domain tasting [28] and domain kitting [29]. Due to a rise in abuse, ICANN implemented the AGP Limits Policy in 2008 [30], significantly curbing such incidents. However, the rules are not strictly followed, and certain domain registrars, including Godaddy[5], Google[6], and Namecheap[7], still permit a limited amount of refunds.

For the first time, we examine the security risks associated with AGP abuse, specifically in the context of shared certificate issuance. Attackers can purchase numerous domains from registrars offering refunds, then issue shared certificates for these domains. Subsequently, they release the domains during the AGP and receive refunds. This strategy enables attackers to acquire a shared certificate with multiple released domains at no cost.

D. Making Illegitimate Certificate Irrevocable

A fundamental prerequisite for CrossPUSH and CrossSXG attacks is the use of an illegitimately acquired shared certificate. If such a certificate is identified and subsequently revoked by the victim, the attack becomes ineffective. However, we discover that it is possible to hinder the victim's ability to revoke the certificate without the attacker's consent.

Commonly, when requesting a CA to revoke a shared certificate, users must either be able to pass DOV for all domains included in the certificate [31] or possess the private key of the certificate[8]. If the attacker issues a new shared certificate that encompasses both the victim's and the attacker's domains, the victim would be unable to meet either of the revocation conditions, rendering the illegitimate certificate non-revocable. We have verified this problem leveraging a trusted CA (i.e., *ZeroSSL*) and our test domain names. Despite reporting our security concerns about the existence of spoofing certificates beyond our control through the CA's official problem reporting platform, the shared certificate could not be revoked. This demonstrates that due to this technique, even if the victims identify an illegitimate certificate, they cannot prevent CrossPUSH and CrossSXG threats.

V. Real-World Exploitation

In this section, we discuss exploiting CrossPUSH and CrossSXG. Once an attacker has a shared certificate, they can launch a malicious website to target other domains in the certificate. Since server push and SXG are rendered in HTTP responses, the attacker can manipulate both HTTP body and header to initiate various cross-origin web attacks by exploiting CrossPUSH and CrossSXG.

[5]https://www.godaddy.com/en-sg/legal/agreements/refund-policy
[6]https://support.google.com/domains/answer/6000754?hl=en
[7]https://www.namecheap.com/legal/general/refund-policy/
[8]https://letsencrypt.org/docs/revoking/

A. Leveraging HTTP Body

HTTP bodies determine the main contents exhibited on web pages. By crafting HTTP bodies, attackers could execute UXSS or conduct phishing attacks.

Exploit-1: Universal cross-site scripting (UXSS) attacks. Given the ability to forge origins and control front-end web pages, a straightforward exploitation is the XSS attack. An attacker can add a script tag into the response body, e.g., `<script>alert(document.cookie)</script>`, to let the browser run arbitrary JavaScript on the victim's website. Such exploitation can steal a user's cookie and initiate a money transfer to an attacker's account. Besides, our attack is the most harmful UXSS which is independent of the target website. Regardless of whether the target website has vulnerabilities, is offline, or no longer exists, our attack still works. It is worth noting that security policies like Content Security Policy cannot prevent such attacks because browsers consider the scripts from the same origin.

Exploit-2: Website phishing. The attacker can customize the whole HTTP body to a phishing HTML, inducing victims to hand over usernames, passwords, or other secret information on a familiar website. It should be noted that our phishing is hard to detect. For instance, if the attacker wants to phish the users of *example.com*, he can disguise his malicious website as *example.com*'s portal, leading users to click. Then, through CrossPUSH or CrossSXG, victims will see the address in the navigation bar rewritten to the *example.com* with the certificate status showing secure (a lock). In addition, the page can be spoofed to look exactly like *example.com* with only modifying, e.g., the address of the submission form to the servers of the attacker, thus completing the phishing attack imperceptibly.

B. Leveraging HTTP Header

Many HTTP headers carry security properties, with some having been analyzed in previous works [32]. In our attacks, exploitable headers should satisfy three basic conditions: 1) response header, 2) cacheable, and 3) security-related. Upon thorough inspection, we identified two stateful headers (i.e., `Set-Cookie`, `Strict-Transport-Security`) as exploitable, given their ability to modify client states or behaviors.

Exploit-3: Setting arbitrary cookies. The `Set-Cookie` HTTP response header is used to send a cookie from the server to the user agent, so that the user agent can send it back to the server later to notify the state of the user, such as login status or personal profile. This header has many attributes, the attacker can set `Domain` to the victim domain, `path` to `/`, and `Expire` to a distant future, to sustainably set the user's cookie. As the cookie is commonly used in today's web access, an attacker who is capable of setting an arbitrary cookie to the victim websites can lead to disastrous impacts like session fixation attack [33].

Exploit-4: Bypassing HTTP Strict Transport Security (HSTS). The `Strict-Transport-Security` header informs browsers that the site should only be accessed using HTTPS, and any future attempts to access it using

HTTP should automatically be converted to HTTPS. This header is a pivotal security strategy that avoids SSL strip and other types of man-in-the-middle (MITM) attacks, however, it can be bypassed by our attacks. The adversary can set `Strict-Transport-Security: max-age=0` to delete the HSTS of the victim domains on browsers. Consequently, the browser can use HTTP to access the victim's websites, breaking the security practices of the websites.

Additionally, in the case of SXG, signing a file with the aforementioned stateful headers is disallowed for security reasons. However, we discover that the implementation of this restriction is flawed and can be circumvented. Rather than having browsers check for stateful headers, this verification is performed by the official SXG web package provided by WICG[9]. This approach overlooks the possibility of an attacker not using the official tool but instead developing their own. Therefore, we modified the web package for SXG and created an SXG generator that does not check for stateful headers, enabling us to launch CrossSXG by leveraging HTTP headers.

C. Leveraging Body and Header

Besides exploiting the HTTP body and header respectively, the attacker can also use HTTP headers to control how the body is parsed, combining them together to launch an attack.

Exploit-5: Malicious file download. An attacker can leverage the `Content-Disposition` header to indicate if the content is expected to be displayed inline (as a web page, default) or as an attachment. Since the attacker can also control the HTTP body, he is able to arbitrarily specify the content of the file and let the victim browser download it as an attachment. More importantly, this header has an attribute `filename`, which is capable of specifying the file name as well as the file extension. As a result, the attacker can use the ransomware or trojan as the HTTP body and the `Content-Disposition: attachment; filename="instruction"` as the header to make the victim's browser automatically download a file named "instruction". From the victim's perspective, this file is downloaded from an intended website with a reasonable name, which seems to be secure. If the victim clicks the file, his computer will be possibly implanted with a virus and controlled by the attacker.

D. Attacking Multiple Targets with Single Click

Furthermore, the high concurrency of HTTP can be exploited to attack multiple targets with a single click. HTTP/2 supports numerous concurrent streams whose maximum number is limited by *MAX_Concurrent_Stream* (set as 128 in default). For the CrossPUSH attack, an attacker can initiate multiple push streams simultaneously, and each authority header of the streams is forged to be one of the victim websites in the shared certificate, enabling attacking numerous targets at once. Regarding CrossSXG, it's easier to achieve this goal since CrossSXG merely needs the static *request-uri* to indicate the target website, rather than the authority of the stream. Therefore, an attacker can directly sign the SXGs of different domains in the shared certificate and serve them on distinct paths. At last, the attacker can use plenty of iframe to let the browser request all of the SXGs, attacking multiple targets with a single click.

[9]https://github.com/WICG/webpackage

VI. Large-Scale Evaluation

In this section, we conduct a large-scale measurement to evaluate the real-world impact of our attack. The evaluation mainly falls into two categories: one is the client-side browsers' behaviors for CrossPUSH or CrossSXG, and the other is the server-side affected websites.

A. Client-side: Vulnerable Browsers

CrossPUSH and CrossSXG attacks are constrained by browser behavior, as different browsers may react distinctively to these two attacks. Additionally, the diversity and quantity of vulnerable browsers greatly affect the impact of the attacks. To find out how many client-side browsers are vulnerable, we conduct a large-scale measurement of browsers' behaviors.

1) Measurement Challenge and Solution: Considering the wide variety of browsers used in our daily lives, it is common to have multiple browsers installed on different devices such as smartphones, tablets, and computers. Furthermore, the multitude of browser versions can lead to behavioral inconsistencies across different versions. Thus, manually collecting all the different browsers is a formidable task, posing significant challenges for comprehensively evaluating client-side browsers. This motivates us to develop *PSChecker*, an integrated platform to conduct the client-side measurement and help to analyze the browsers' behaviors on server push and SXG. By deploying the platform on high-traffic websites, visitors will obliviously help us to automatically test their browsers' behaviors, increasing the range and diversity of the browsers to complement our unmeasured test objectives.

Nevertheless, *PSChecker* may raise ethical concerns if we directly use it to measure CrossPUSH and CrossSXG attacks in the wild. Moreover, *PSChecker* only provides coarse-grained access data, which may require a few supplements. To address the above issues, we determined to accomplish client-side measurement through a two-step process. First, we utilize *PSChecker* to broadly measure the browsers' feature support for server push and SXG without any cross-origin attack behaviors. The user's access data will enable us to discover more testing objectives and give us an overview of browsers' support for these two features. Second, we launch a strictly controlled web environment to block other users' requests and download all suspicious browsers locally to test their behaviors on CrossPUSH and CrossSXG attacks. We also manually supplement some data, such as the latest version of the vulnerable browser, the older version of the invulnerable browser, and the high-ranked browsers that were not tested, etc., to enhance the comprehensiveness of the measurement at a fine-grained level.

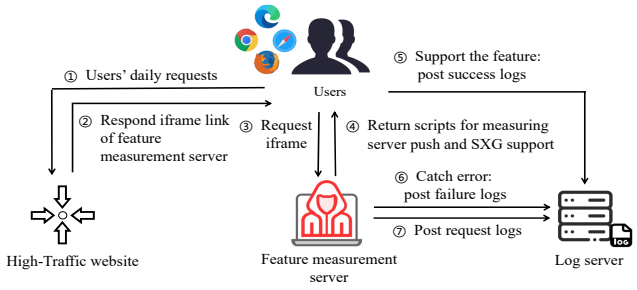


Fig. 4: Overview of *PSChecker*

2) PSChecker and Local Test: *PSChecker* takes full consideration of ethics and does not exploit or steal any credentials from its users. It just uses the legal server push and SXG to measure the feature support and records `User-Agent` (UA) to help us identify browser varieties and versions. Figure 4 shows the overview of *PSChecker*. It consists of three parts: (1) High-traffic website. *PSChecker* needs more visitors to include as many browsers as possible, so we embedded our test site as an iframe on a high-traffic website from our industrial partners; (2) Feature measurement server. An HTTP/2 server returns scripts via server push and SXG for measuring feature support when users request it. Besides, it records some basic information like UA and sends logs to log servers; (3) Log server. An HTTP server that processes post requests and records success log, failure log, as well as the full log. These logs will be used in the following analysis.

***PSChecker* workflow.** First, users request our industrial partner's high-traffic website through their browsers. As a traffic honey pot, the website not only returns the normal HTML but also adds an iframe within the `<body>` tag of the HTML, with the link source set to the feature measurement server. The width and height of the iframe are set to 0 to make it invisible, which aims to avoid affecting users' browsing experience. Second, the browser receives the response HTML and then accesses the iframe link, directing the traffic to the feature measurement server. After receiving the request, the feature measurement server initiates server push and SXG with scripts for measuring feature support. If the browser supports the feature, it will execute the script and send the success log to the log server. If the browser does not support the feature, e.g., reject the push stream, the feature measurement server will receive an error event. We can catch the error and post the failure log to the log server. Last, since not all failures result in an error event, for example, browsers without SXG support do not have a corresponding error event. Therefore, the feature measurement server will post all user requests to the log server to maintain a full log for comparison. If the behavior of a browser is neither in the success log nor in the failure log, then this browser failed without an error event.

Subsequent local tests. After we automatically get the result of browsers' support for server push and SXG from *PSChecker*, we then locally test the behaviors of suspicious browsers on our attacks in a lab environment. We launch an attack server (like *attacker.com* in Figure 1) to initiate CrossPUSH and CrossSXG attacks with an XSS script. The origin of the XSS script is set to another domain in the shared certificate. Besides, in CrossPUSH, whether the IP address of the requesting domain is the same as the authority header will affect browsers' behaviors. So, we further indicate the origin of the pushed script to cross-ip domain and same-ip domain, dividing CrossPUSH into cross-ip and same-ip CrossPUSH. Moreover, we identify browsers from UAs in the result of *PSChecker* and supplement uncovered top-used browsers reported by Statcounter[10] to the testing list. We download different versions of browsers in the list to three leading platforms (Windows, iOS, Android) from official websites. At last, we use browsers to request the attack server and analyze the behaviors.

3) Client-side Result Analysis: We deployed *PSChecker* for 30 days, from Nov 15 to Dec 15, 2023. There were 10,072 visits to our *PSChecker* in 30 days. Measurement results cover 2,873 different user agents and more than 30 different types of browsers after de-duplication. We tried our best to figure out the browser type from each UA and supplement the missing top-used browsers in Statcounter. Finally, we identified and systematically tested 26 browsers at a granular level across different software versions and operating systems in total. They were categorized into three groups: (1) Top-used browsers, (2) Default browsers on mobile, and (3) Celebrated apps.

Top-used browsers: 11/14 of them are vulnerable to at least one of our attacks. The upper part of Table II includes top-used browsers reported by Statcounter and other well-known browsers identified from UA. We evaluated these browsers' behaviors on three popular platforms, i.e., Windows, iOS, and Android. The result shows that the latest versions of most browsers (11/14) by the time of testing are vulnerable to at least one of our two attacks on at least one platform. Browsers' behavior on iOS is the most special case: except for the UC browser being fully vulnerable to CrossPUSH (we will analyze this finding later), other browsers are either partially vulnerable or not vulnerable. That is because Apple Inc. requests all browsers on iOS to use Webkit kernel, making browsers on iOS react the same as Safari. In terms of CrossSXG, the latest versions of most browsers are vulnerable. Safari, Firefox, and IE are not vulnerable since they do not support the SXG feature now. Whether this feature will be developed in the future and suffered from our attack is unknown. For CrossPUSH, UC, QQ browser (Win), and Instabridge (Android) are fully vulnerable, while the rest of the browsers are partially vulnerable. The reason is that most developers of these browsers use Chromium implementation, and Google removed the server push feature in Chromium106. So, the latest version does not support server push, but the old versions are vulnerable. Webkit-based safari only suffered from same-IP CrossPUSH until version 17, which required that the IP address of the victim website and the currently

[10]https://gs.statcounter.com/browser-market-share

TABLE II: Browser behaviors on CrossPUSH and CrossSXG

Top-used Browsers						
	CrossPUSH			CrossSXG		
Browser	Win	iOS	Android	Win	iOS	Android
Chrome	◐	◐	◐	●	○	●
Safari	-	◐	-	-	○	-
Edge	◐	◐	◐	●	○	●
Firefox	○	◐	○	○	○	○
Opera	◐	◐	◐	●	○	●
UC	●	●	●	○	○	○
360	◐	◐	●	●	○	●
IE	○	-	-	○	-	-
Yandex	◐	◐	◐	●	○	●
QQ browser	●	◐	◐	●	○	●
CocCoc	◐	-	◐	●	-	●
Whale	◐	◐	◐	●	○	●
Instabridge	-	◐	●	-	○	●
Xunlei	●	◐	◐	●	○	●
Default Browsers on Mobile						
Samsung	-	-	◐	-	-	●
Huawei	-	-	●	-	-	●
Oppo	-	-	●	-	-	●
Xiaomi	-	-	◐	-	-	●
Vivo	-	-	●	-	-	●

[1] The red bar symbolizes that the browser's latest version is vulnerable to at least one of our attacks.
[2] ●denotes that the browser is vulnerable in the latest version;
[3] ○denotes that the browser is not vulnerable;
[4] ◐denotes that the browser is partially vulnerable, e.g., vulnerable in the old versions or just vulnerable to same-IP CrossPUSH;
[5] - denotes that the browser is not supported on this platform.

connected website should be the same, otherwise, it would not use cross-origin pushed assets. Besides, iOS also no longer supports the server push feature in the latest version 17. In summary, invulnerable browsers or partially vulnerable browsers are caused by the temporary lack of support for SXG or server push, while supported versions other than Firefox and IE are all vulnerable, which is alarming.

Default browsers on mobile: five leading mobile browsers are vulnerable. Mobile browsers contribute more traffic than desktop browsers worldwide[11], so we systematically tested the behaviors of default browsers on mobile. We collected 5 leading mobile phone manufacturers in the world[12] (Apple Safari has been presented in the upper part), as shown in the lower part of Table II, all of the tested default browsers' latest versions are vulnerable to at least one of our attacks, posing a significant threat to the mobile users. It should be noted that many mobile applications will open the URL link in the default browser, which increases the possibility of the users being attacked.

Celebrated applications: vulnerable Webviews spread the threat to apps. Apart from the browser, our *PSChecker* found that lots of apps also have built-in browsers that are susceptible to our attacks. Thus, we further tested them at a fine-grained level. As shown in Table III, while all of them are vulnerable, Baidu and Quark behave distinctively compared to other apps, which we will analyze later. Notably, the rest of the apps are partially vulnerable to CrossPUSH because these apps' behaviors vary from devices in our test (e.g., succeed on Huawei Mate 50 but fail on Xiaomi 10 pro). We think the reason is these apps depend on the Webviews of the user's devices. Unfortunately, the default browser on mobile reflects the behaviors of Webview in general, considering that many devices have insecure default browsers, our attacks will affect a large number of apps.

TABLE III: Celebrated apps

	CrossPUSH		CrossSXG	
App	Android	iOS	Android	iOS
Baidu	●	●	●	○
Quark	●	●	○	○
Instagram	◐	◐	●	○
Wechat	◐	◐	●	○
QQmail	◐	◐	●	○
Weibo	◐	◐	●	○
Tiktok	◐	◐	●	○

[1] All symbols have the same meaning as Table II.

Software libraries: some libraries and software supply chains are vulnerable. We observed that UC, Baidu, and Quark are vulnerable to the CrossPUSH attack on iOS, behaving distinctively compared to other browsers or apps, even though they indeed use the Webkit kernel that is forced by Apple. We consider the main reason is that the network requests are taken over from Webkit by the developers. This viewpoint is verified by the developers of the Baidu App, who used the Chrome-Net library to take over the network requests. This critical finding reveals that vulnerable libraries may also lead browsers to suffer from our attacks, and the software supply chains of browsers may be vulnerable.

B. Server-side: Affected Websites

1) Measurement Objectives: In this part, we measure how many websites are potentially threatened by our attacks. Concretely, within our attacks, the websites that can be included in the shared certificates of attackers, are the victim websites. However, given the diverse methods of acquiring shared certificates, we cannot evaluate all situations. As a result, we make clear our measurement objectives as the following three typical categories of affected websites.

Reselling domains. These affected domains indicate the domains once resold from attackers to victims. As we describe in Section IV-A Method-1, an attacker can issue a multiple-domain shared certificate first and resell domains to victims. Subsequently, these resold domains can be directly targeted by the original domain owners with our attacks until the shared certificates expire.

[11] https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet
[12] https://www.globalbrandsmagazine.com/top-10-mobile-brands-in-world/

Dangling domains. These affected domains indicate the domains whose DNS records point to applicable public services that can be hijacked by any attackers. As shown in Section IV-A Method-2, an attacker can take over these dangling domains and issue the shared certificate. Then, these dangling domains will be under our attacks for more than two years even after the dangling records are removed.

Cert-sharing domains. These affected domains indicate the top websites that are included in the existing certificates' SAN lists of other peer websites. For instance, *google.com* is a cert-sharing domain since it is included in the certificate of other websites like *admob-cn.com*. Because peer websites hold the shared certificate, these cert-sharing domains can be directly attacked by peer websites using CrossPUSH and CrossSXG. More importantly, attackers can compromise a less secure peer website first, get the shared certificate, and launch our attacks against these cert-sharing domains.

2) Measurement Methodology: We measure three types of affected websites in different ways, as the following:

Measuring reselling domains. We use WHOIS history data to measure how many of the Tranco Top 1M domains have experienced reselling within our attack window (796 days), disclosing domains that are subject to potential exploitation by the original domain owners. Firstly, since we cannot analyze anonymous WHOIS, we excluded as much anonymous WHOIS data as possible through keywords [3]. Secondly, we use three strict conditions to filter out the reselling domains: (1) Three main registrant's information has been changed; (2) Levenshtein similarity [34] between old registrant organization and new one was less than 0.3, which aims to prevent some tiny changes in one entity, e.g., from Google to Google LLC; (3) Two reselling period (createdDate to expiresDate) does not overlap and the interval between them is less than our attack window (796 days). Domains that meet these three strict conditions are considered as reselling domains affected by our attacks.

Measuring dangling domains. To identify dangling domains in the wild, we utilized the state-of-the-art tool, HostingChecker [22], for large-scale measurement. This tool primarily targets public hosting services susceptible to domain takeovers. However, some of the services (e.g., OSS) are not exploitable for issuing TLS certificates and launching our attacks. Through manual analysis, we identify three scenarios feasible for forging certificates: (1) Dangling DNS records pointed to CDN services; (2) Dangling `A` records pointed to applicable virtual private server (VPS) IPs; (3) Dangling canonical name (`CNAME`) records pointed to unregistered domains. To this end, we extend HostingChecker to assess these specific situations. For case (1), we supplement with more vulnerable CDN services by manually testing all of the CDN vendors from CDN planet[13] since HostingChecker misses some vendors and several CDNs like Bunny do not allow certificate issuance in our tests. By adding the outdated CDN service fingerprints to the config of HostingChecker, we can directly measure this case. For case (2), we utilize HostingChecker to resolve every domain to IP address. Then, we check whether the IP belongs to three main VPS providers, i.e., AWS, Azure, and Aliyun. At last, we refer to the liveness detection method in previous work [23] to find if the pointed VPS is applicable. For case (3), we just make a WHOIS request to every `CNAME` record resolved by HostingChecker and inspect whether there is an unregistered domain.

[13]https://www.cdnplanet.com/

TABLE IV: Overall result on server-side affected websites

	Reselling	Dangling	Cert-Sharing
Scope	Top 1M	Top 1M and subdomains	Top 1K
Count	11741	4919	829

Measuring cert-sharing domains. We scope this measurement to the cert-sharing domains in Tranco Top 1K. At a high level, we find out which website's TLS certificate contains Tranco Top 1K websites and whether there are high-ranked domains included in the shared certificates of low-ranked ones, revealing the security degradation of top websites. Given the extensive domain landscape, acquiring and analyzing all of the domain's certificates proves inefficient. As a result, we should quickly locate the relevant domains of the Top 1K sites and continue the measurements within this pruned range. Our key insight is that the domains in the SAN list of Top 1K websites' certificates must have a strong association and be possible to share certificates with Top 1K. So, we take measurements in the following steps: (1) Collecting "related" domains in certificates' SAN of Top 1K sites. We first establish a connection on port 443 for each Top 1K site and utilize the Python SSL library to extract the certificates from the context, capturing the domains in the SAN from each certificate as the fundamental dataset; (2) Constructing relevant domain dataset. Secondly, We expand the data in the form of wildcards from the fundamental dataset by recursively crawling subdomains. For subdomain crawling, we employ various methods to crawl as many subdomains as possible, including searching for subdomains in HTTP response, Certificate Transparency log (crt.sh), and passive DNS (from our industrial partners). We use this augmented dataset as the relevant domain dataset; (3) Constructing the Top 1K dependency dataset. Then, we use the method of step (1) again to retrieve certificates of relevant domains and inspect whether their certificates' SAN lists contain Top 1K websites; (4) Acquiring rank dependency. Consequently, we search the rank of the domains that share certificates with the Top 1K websites from both Tranco Top 1M and Tranco Top 1M (with subdomains) to find the rank dependency phenomenon. We supplement the Tranco rank (with subdomains) to ensure the fairness of our evaluation since lots of domains are not apex domains.

3) Server-Side Overall Results: Overall results are shown in Table IV. Specifically, at least 11,741 Tranco Top 1M domains were once resold within our attack window, which could be attacked by original domain owners. In a month,

4,919 dangling domains (including domains from celebrated companies, like Microsoft as we present in Section VI-C) can be compromised by us and we can launch CrossPUSH to these websites even for two years after they are not "dangling". Besides, 829 (82.9%) of Tranco Top 1K domains are cert-sharing domains that are suffering security downgrades.

4) Result Analysis of Reselling Domains: **Finding-1: Domain reselling is a widespread practice, with even many high-ranking domains having been resold.** By carefully analyzing the WHOIS history data, we identified 1 domain in Tranco Top 1K, 59 domains in Top 10k and 11,741 domains in Top 1M have been resold before. They all have been changed hands from one person/organization to another within our attack window (796 days). Notably, domain reselling can also threaten corporate websites. For example, our measurement showed that the owner of ftstatic.com (Tranco rank 3895 on Nov 18, 2023) once changed from foodtraders.com.au Pty Ltd (an Australian food company) to Flashtalking (an American advertising agency). Even worse, the community is unaware of the security risks posed by domain reselling at all, and has even developed numerous domain reselling platforms[14]. These discoveries demonstrate that our attacks will pose a novel and serious threat to the prevalent domain reselling.

5) Result Analysis of Dangling Domains: **Finding-2: The amount of exploitable dangling domains is large.** We measured the dangling domains of Tranco Top 1M and its subdomains 12 times (a month) to reveal how many websites were threatened by our attacks due to temporary dangling DNS records. For (1) dangling DNS records pointed to applicable CDN services, 19 domains can be exploited by us. For (2) dangling `A` records pointed to applicable VPSs, 3,952 domains were exploitable, of which 2,513 domains' `A` records were pointed to AWS VPSs, 788 to Azure, and 651 to Aliyun. For (3) dangling `CNAME` records pointed to unregistered domains, we found that 948 domains' `CNAMEs` were pointed to unregistered domains. Attackers can acquire the shared certificate of these dangling domains by merely applying for the services or buying the unregistered domains. As a result, dangling domains are very common, which demonstrates that a large number of websites are affected by our attacks.

Finding-3: The duration of our attack is commonly longer than traditional domain takeover. Moreover, we explored the existence duration of these exploitable dangling domains. For example, by delving into the 948 dangling domains whose `CNAME` records pointed to unregistered domains, we found that they are pointing to 162 unregistered apex domains after de-duplication. Through analyzing the WHOIS historical data, we statistically obtained that the median number of days from the expiry time of each unregistered domain to the start time of our measurements was 148 days. At the end of the one-month measurements, only 2 once-dangling domains were still pointing to unregistered domains, and almost all other dangling DNS records were removed or the domains they were pointing to were registered. This implies that most dangling domains cannot be hijacked after six months, which is also demonstrated by the previous study on dangling domain lifecycle [22]. Such results prove that our attack poses a more significant threat to dangling domains, as we extend the duration of domain takeover attacks, which usually tend to be invalid within six months, to up to 796 days.

6) Result Analysis of Cert-Sharing Domains: The result painted a bleak picture of certificate sharing and security dependency. Although these issues cannot let an attacker obtain the shared certificate directly, the analysis of cert-sharing domains demonstrated how CrossPUSH and CrossSXG can increase the attack surface of high-profile sites. For example, instead of looking for vulnerabilities in strictly protected top sites, the adversary can compromise less secure websites that share certificates with these sites first, and launch our attacks against these high-profile cert-sharing domains with the acquired certificates. Therefore, we analyze the current situation of shared certificates below.

Finding-4: 829 Tranco Top 1K websites are included in the shared certificates of low-ranked websites. By crawling the subdomains, we discovered 99,827 fully qualified domain names (FQDNs) potentially relevant to Tranco Top 1K domains and eventually confirmed that there were 45,930 domains sharing certificates with the top 1K websites. Among these peer domains, 16,950 domains were the subdomains of the top 1K domains and the rest were the non-subdomains. It shows that many high-profile websites not only rely on the security of their subdomains but also rely on non-subdomains, which may belong to acquired companies, subsidiaries, or even distinct organizations. In addition, as shown in Figure 5, most websites sharing certificates with Tranco Top 1K were out of Tranco 1M. For instance, 43,594 / 45,930 (94.9%) domains were out of Tranco (only apex) 1M while this was 39,436 / 45,930 (85.9%) in Tranco (withsub). Actually, 953 and 829 of the Top 1K websites were included in the certificates of domains ranked out of the Top 1M according to these two domain rankings.

This existing ecosystem of certificate-sharing raises two major concerns: (1) Many companies do not always update their certificates when their acquired companies have changed ownership, which was confirmed by Baidu. After receiving our report, they noticed that some of their acquired companies became independent but still held certificates shared with Baidu's domains, which brought potential threats. (2) Certificate sharing leads top-ranked sites to suffer security degradation. Generally, the lower-ranked sites are relatively less secure, since security practices may act as ranking signals[15] and previous study [16] demonstrated that low-ranked websites react more slowly than high-ranked sites to security events like Heartbleed. Therefore, attackers are more likely to compromise these less secure sites, acquire certificate private keys, and launch attacks on high-profile sites.

[14]https://www.resellerspanel.com/domain-names/

[15]https://developers.google.com/search/blog/2014/08/https-as-ranking-signal

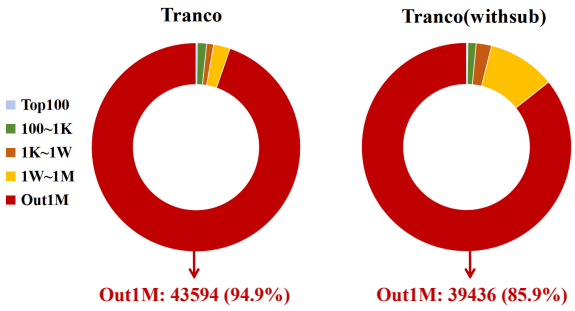


Fig. 5: Rank distribution of the domains that share certificates with Top 1K websites. Most of the domains sharing certificates with Top 1K are ranked out of 1M.

C. Case Study: Attack Microsoft's Domain

To demonstrate the threat of our attacks in the real world, we analyzed a domain from Microsoft. We found that the `CNAME` record of 14.au.www.download.windowsupdate.com (a Microsoft's domain) was pointing to an unregistered domain au.download.windowsupdate.qtlcdnect.com. So, we bought qtlcdnect.com from Godaddy. Due to ethical considerations, we did not continue the exploitation. If we were malicious attackers, we could set the `A` record of au.download.windowsupdate.qtlcdnect.com to our server. Then, we can issue a certificate of our own domain shared with 14.au.www.download.windowsupdate.com in HTTP mode. With the shared certificate, we can successfully launch CrossPUSH attacks on this domain. As this domain serves as a download site for Windows update files, we can initialize the malicious file download attack to let the victims who request the malicious website (disguised as a Windows update portal) receive a malicious file. Because the browser shows that the file is indeed downloaded from Microsoft's official website, victims are then likely to execute it, thus being implanted with a trojan horse or virus. Remarkably, compared to traditional domain takeover, our attacks will remain effective for a maximum duration of 796 days, even after Microsoft removes the dangling DNS records.

VII. Discussion

A. Mitigation

The root cause of our attacks stems from two primary factors: (1) Relaxation of the browser's origin in HTTP/2 and HTTP/3. As server push and SXG can indicate the origin of the resources to all domains in the SAN list of multi-domain shared certificates, the boundary of the browser's origin is changed on these two features. This relaxation can compromise the SOP protection, which leads to various attacks. (2) Misalignment between certificate and domain ownership. Currently, there are no robust mechanisms to ensure that certificate ownership aligns with domain ownership. In our analysis, there are various ways in which an attacker can get a certificate for a domain that does not belong to him, making the attack condition of shared certificates can be fulfilled in practice. As server push and SXG overly trust the certificate owner's control of the domains in the certificates, the attacks can seriously threaten the real world. To defend against our attacks, we propose the following mitigations:

Enforcing consistent authority in browsers to mitigate CrossPUSH. To mitigate CrossPUSH attacks, browsers should detect and reject server pushes where the authority of the push stream does not match the origin of the current connection. This includes preventing such pushes from being cached. Additionally, we suggest browsers adhere to the recommendations of RFC 9110 [2], i.e., perform a secondary DNS query to check that the origin's host contains the same server IP address as the established connection in HTTP/2 and HTTP/3.

Enforcing single-domain certificates to mitigate CrossSXG. In terms of CrossSXG, we suggest maintaining its cross-origin delivering ability but deleting its cross-origin signing ability, i.e., an SXG provider should not use a multiple-domain shared certificate to generate SXG. Since verifying the attribution of SXG resources only needs single-domain certificates, browsers can just reject to decode the SXG that is signed by a shared certificate.

Inspecting certificate status in domain registration. To mitigate the risk of attackers reselling domains while retaining the certificates, domain buyers should inspect Certificate Transparency (CT) logs when registering new domains. Registrars should also assist by reminding customers to check the certificate status of domains being transferred, and potentially even notifying CAs directly.

Revoking shared certificate upon requests from domain owners in SAN lists. Given the potential for attackers to use tactics (as described in Section IV-D) that prevent certificate revocation, CAs should facilitate the removal of domains from shared certificates at the request of domain owners listed in the SAN. Since revoking the shared certificate may affect the current use, CAs should also notify the current holders of the shared certificate and assist them in renewing a certificate without the removed domains.

B. Ethical Consideration and Responsible Disclosure

Ethical consideration. We take the utmost care of potential ethical issues. (1) *In terms of client data collection*, our client-side measurement is conducted in collaboration with an industrial partner who deployed a measurement script on their website. First, the script is designed solely to determine whether a client supports server push and SXG, and to log the browser's `User-Agent` string. No other potentially sensitive client attributes are sent or stored. Second, the partner company includes a consent mechanism at login and sign-up, informing users that their browser features would be measured. Third, the measurement script and collected data are also rigorously reviewed by our partner company to ensure it cannot track users or affect user privacy or experience. Notably, all websites and servers are under the company's control, and we do not participate in deployments. At last, the client-side attack tests are locally conducted on our own websites with web application firewalls to strictly block other

traffic, which does not affect any other users or websites; (2) *In terms of server-side measurement*, we limit the scanning rate to minimize the impact on real-world servers; (3) *In terms of disclosure process*, companies encourage security tests through bug bounty programs, and we carefully follow disclosure guidelines when disclosing issues to vendors and domain owners.

Responsible disclosure. We have reported these security problems to vulnerable browsers and affected websites. Up to now:

- **Huawei** acknowledged our report and has confirmed the vulnerability. They have updated their browser application to 14.0.4 to prevent our attacks.
- **Baidu** was interested in the attacks and had an in-depth discussion with us about the specifics. They have confirmed the vulnerability and will arrange a fix for this problem in the next version.
- **Xunlei browser** acknowledged our report and has confirmed the vulnerability and will fix this problem in the next version.
- **Google** acknowledged the vulnerability and is still in discussion on how to address this issue.
- **Microsoft** acknowledged our report and replied that they would remove the dangling DNS record of the affected websites later.

C. CDN Analysis

Content Delivery Networks (CDNs) might exacerbate our CrossPUSH attack, making it more severe and widespread. With CDNs, end-to-end HTTPS connection is split into two parts: the front-end communication between client and CDN surrogate servers, and the back-end communication between CDN surrogate servers and original web servers. Since surrogate servers always use shared certificate [35] and some CDNs even merge different entities' domains into one certificate [36], these naturally shared certificates of CDNs in front-end connections potentially create an avenue for attackers to launch CrossPUSH attacks.

For instance, Fastly CDN uses a wildcard shared certificate to support HTTP/2 in TLS services[16], as long as the surrogate servers support forwarding HTTP/2 push stream to clients, an attacker can push scripts to any other websites in **.freetls.fastly.net* as all of them are authorized by the shared certificate on Fastly surrogate server. To evaluate this phenomenon, we investigate nine prominent CDNs and find that HTTP/2 features have not been well-supported. Specifically, certain CDNs, such as Amazon Cloudfront and CDN77, do not provide HTTP/2-to-origin support. Some CDNs, like Cloudflare and Fastly, do not forward HTTP/2 push streams. Moreover, CDNs like Tata Communications have not implemented any form of HTTP/2 support at all. However, there is a growing trend within the community towards embracing those advanced features, e.g., Go programming language has introduced client-side support for HTTP/2 server push recently[17]. This trend reveals that the landscape of CDN support for these features may evolve, possibly influenced by vulnerabilities highlighted in our study. Furthermore, we suggest CDN vendors consider our attacks when integrating HTTP/2 features in the future, and the recommendations outlined in Section VII-A can assist CDNs in preventing these potential issues.

[16]https://docs.fastly.com/en/guides/setting-up-tls-on-a-shared-fastly-domain#support-for-http2-ipv6-and-tls-12

VIII. Related Work

A. CrossPUSH and CrossSXG

Jake's blog [37] first described that HTTP/2 server push can push cross-origin resources to other domains in a certificate. However, he only briefly mentioned it and did not conduct an in-depth study. The real-world security threats, exploitation methods, and attack conditions of this feature are not yet known. As for SXG, Andrew Ayer [38] exposed in the draft discussion stage that this feature will enable the attacker who has obtained the certificate to launch an off-path attack on the victim's website and gave several examples of obtaining illegitimate certificates in the past few years. Nonetheless, this security issue has not been systematically studied. In contrast, we provide a detailed and in-depth study of CrossPUSH and CrossSXG attacks, along with their exploitations and practicality evaluations in the real world.

B. Other Attacks of Shared Certificate

Previous study shows that it is common for administrators to deploy shared certificates across multiple services, possibly without consideration to cross-protocol attacks [39]. For instance, a MitM attacker can redirect the HTTP request to an FTP server that has a certificate compatible with the web server. If the FTP server is error-tolerant, it will reflect the embedded JavaScript to the client in an error message, making the exploitable browser execute it within the context of the original request. Another work leverages the shared TLS certificate to launch the HTTPS context confusion attack [32]. By rerouting encrypted traffic to another flawed server that shares the TLS certificate, attackers can bypass the security practices, hijack the ongoing HTTPS connections, and subsequently launch other attacks, including phishing and payment hijacking. We should note that all of these attacks need the adversary to have the privilege of rerouting the traffic, and the exploitable server should have some flaws, e.g., not adhering to the best HTTPS practices. On the contrary, the attacker with a shared certificate can launch off-path web attacks on other people's websites via CrossPUSH and CrossSXG, even when the target website does not have any vulnerability or is not online.

IX. Conclusion

We propose how an off-path attacker only with the shared certificate can launch cross-origin web attacks via HTTP/2 server push and SXG, along with a comprehensive study of their practicality in the real world. On the client side,

[17]https://go-review.googlesource.com/c/net/+/181497

11/14 top-used browsers, including Chrome and Edge, are vulnerable to at least one of our attacks, and the threat even spreads to numerous apps. On the server side, we show that three exploitable cases, reselling domains, dangling domains, and cert-sharing domains are prevalent. We consider that the essence of this security problem is server push and SXG, two emerging features with the ability to indicate server authority, can leverage the inconsistencies between browser origin and HTTP/2 authority to break the same origin protection, enabling an adversary to attack any domains in his illegitimately acquired shared certificate. We expect our work to raise awareness of this problem and set off further discussion among practitioners and researchers. We also envision our work to motivate the browser vendors, domain registrars, and CAs to collaborate on operational efforts to address the issues in the current practices.

ACKNOWLEDGMENT

We thank the anonymous reviewers and our shepherd for their insightful feedback that helped improve the quality of the paper. We thank Hui Jiang, Biao Tian, Hao Mo, and Hua Lin from Baidu for their constructive comments and generous provision of high-traffic websites for conducting our measurement. We also thank our industry partners from QiANXIN for their support of passive DNS data. We are grateful to Yihang Wang, Keran Mu, and Run Guo for their peer review and helpful suggestions. This work was in part supported by the NSFC #62272265. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the funding agencies.

REFERENCES

[1] A. Barth, "The web origin concept," *RFC*, vol. 6454, pp. 1–20, 2011. [Online]. Available: https://doi.org/10.17487/RFC6454

[2] R. Fielding, M. Nottingham, and J. Reschke, "RFC 9110: HTTP semantics," 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110.html#section-4.3.3

[3] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Measurement and analysis of private key sharing in the https ecosystem," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 628–640.

[4] A. Delignat-Lavaud and K. Bhargavan, "Network-based origin confusion attacks against HTTPS virtual hosting," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 227–237.

[5] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (HTTP/2)," 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7540

[6] M. Bishop, "RFC 9114: HTTP/3," 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9114

[7] T. Martin and B. Cory, "RFC 9113: HTTP/2," 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9113.html#name-server-push

[8] D. M. Katie Hempenius, "Signed exchanges (SXGs)," 2020. [Online]. Available: https://web.dev/articles/signed-exchanges?hl=en

[9] K. Yasuda, "Signed HTTP exchanges," 2018. [Online]. Available: https://developer.chrome.com/blog/signed-exchanges/

[10] L. Fyrd, "Can I use SXG?" 2023. [Online]. Available: https://caniuse.com/?search=SXG

[11] O. Y. Marc Lamik, "Improve site load times and seo with one-click support for signed exchanges on google search," 2021. [Online]. Available: https://blog.cloudflare.com/automatic-signed-exchanges/

[12] CA/Browser Forum, "Ballot SC22 – reduce certificate lifetimes (v2)," 2019. [Online]. Available: https://cabforum.org/2019/09/10/ballot-sc22-reduce-certificate-lifetimes-v2/#Ballot-SC22-Reduce-Certificate-Lifetimes-v2

[13] J. Yasskin, "Draft-yasskin-http-origin-signed-responses-latest," 2023. [Online]. Available: https://wicg.github.io/webpackage/draft-yasskin-http-origin-signed-responses.html#name-status-of-this-memo

[14] M. Sadique, "Abuse of hidden "well-known" directory in https sites," 2019. [Online]. Available: https://www.zscaler.com/blogs/security-research/abuse-hidden-well-known-directory-https-sites

[15] R. van Elst, "How I got a valid ssl certificate for my ISP's main domain, xs4all.nl," 2015. [Online]. Available: https://raymii.org/s/blog/How_I_got_a_valid_SSL_certificate_for_my_ISPs_main_website.html

[16] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. D. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson, "The matter of heartbleed," in *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*, C. Williamson, A. Akella, and N. Taft, Eds. ACM, 2014, pp. 475–488. [Online]. Available: https://doi.org/10.1145/2663716.2663755

[17] "Node.js v20.10.0," 2023. [Online]. Available: https://nodejs.org/docs/latest-v20.x/api/http2.html#http2streampushstreamheaders-options-callback

[18] C. Bormann and P. Hoffman, "RFC 8949: Concise binary object representation (CBOR)," 2020. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8949.html

[19] K. Du, H. Yang, Z. Li, H. Duan, and K. Zhang, "The ever-changing labyrinth: A large-scale analysis of wildcard DNS powered blackhat SEO," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 245–262. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/du

[20] Q. Xie, S. Tang, X. Zheng, Q. Lin, B. Liu, H. Duan, and F. Li, "Building an open, robust, and stable voting-based domain top list," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 625–642. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/xie

[21] D. Liu, S. Hao, and H. Wang, "All your DNS records point to us: Understanding the security threats of dangling DNS records," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1414–1425. [Online]. Available: https://doi.org/10.1145/2976749.2978387

[22] M. Zhang, X. Li, B. Liu, J. Lu, Y. Zhang, J. Chen, H. Duan, S. Hao, and X. Zheng, "Detecting and measuring security risks of hosting-based dangling domains," in *Abstract Proceedings of the 2023 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2023, Orlando, FL, USA, June 19-23, 2023*, E. Smirni, K. Avrachenkov, P. Gill, and B. Urgaonkar, Eds. ACM, 2023, pp. 87–88. [Online]. Available: https://doi.org/10.1145/3578338.3593534

[23] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and G. Vigna, "Cloud strife: Mitigating the security risks of domain-validated certificates," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_06A-4_Borgolte_paper.pdf

[24] CA/Browser Forum, "Baseline requirements for the issuance and management of publicly-trusted tls server certificates," 2023. [Online]. Available: https://cabforum.org/wp-content/uploads/CA-Browser-Forum-TLS-BR-2.0.2.pdf

[25] The Chromium Project, "Moving forward, together," 2023. [Online]. Available: https://www.chromium.org/Home/chromium-security/root-ca-policy/moving-forward-together/

[26] CA/Browser Forum, "Ballot SC42: 398-day re-use period," 2021. [Online]. Available: https://cabforum.org/2021/04/22/ballot-sc42-398-day-re-use-period/

[27] D. L. Jessica, Jackie Treiber, "Add grace period," 2022. [Online]. Available: https://icannwiki.org/Add_Grace_Period

[28] Hexware, "Domain tasting," 2023. [Online]. Available: https://en.wikipedia.org/wiki/Domain_tasting

[29] Jessica, "Domain kitting," 2021. [Online]. Available: https://icannwiki.org/Domain_Kiting

[30] ICANN, "AGP (add grace period) limits policy," 2008. [Online]. Available: https://www.icann.org/resources/pages/agp-policy-2008-12-17-en
[31] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, "RFC 8555: Automatic certificate management environment (ACME)," 2019. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8555#section-7.6
[32] M. Zhang, X. Zheng, K. Shen, Z. Kong, C. Lu, Y. Wang, H. Duan, S. Hao, B. Liu, and M. Yang, "Talking with familiar strangers: An empirical study on https context confusion attacks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1939–1952.
[33] M. Squarcina, P. Adão, L. Veronese, and M. Maffei, "Cookie crumbles: Breaking and fixing web session integrity," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 5539–5556. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/squarcina
[34] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.
[35] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 67–82.
[36] J. Hoyland, "Exported authenticators: The long road to RFC," 2021. [Online]. Available: https://blog.cloudflare.com/exported-authenticators-the-long-road-to-rfc/
[37] J. Archibald, "HTTP/2 push is tougher than i thought," 2017. [Online]. Available: https://jakearchibald.com/2017/h2-push-tougher-than-i-thought/#you-can-push-items-for-other-origins
[38] K. Ueno and A. Ayer, "Intent to ship: Signed http exchanges (SXG)," 2019. [Online]. Available: https://groups.google.com/a/chromium.org/g/blink-dev/c/gPH_BcOBEtc
[39] M. Brinkmann, C. Dresen, R. Merget, D. Poddebniak, J. Müller, J. Somorovsky, J. Schwenk, and S. Schinzel, "ALPACA: Application layer protocol confusion-analyzing and mitigating cracks in TLS authentication," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 4293–4310.

APPENDIX

A. Attack Requirements

TABLE V: Attack requirements for acquiring shared certificates

Attack Methods	Exploited Resources	Attack Requirements
Domain reselling	Resold domains	No requirements
Domain takeover	Inactive cloud services	`CNAME`/`NS` records of victim.com pointed to discontinued cloud services
	De-provisioned VPS IPs	`A` records of victim.com pointed to de-provisioned VPS IPs
	Expired domain names	`CNAME` records of victim.com pointed to unregistered domain names