Talking with Familiar Strangers: An Empirical Study on HTTPS Context Confusion Attacks

Mingming Zhang[1,Φ], Xiaofeng Zheng[1,2,Φ,*], Kaiwen Shen[1], Ziqiao Kong[2], Chaoyi Lu[1], Yu Wang[1], Haixin Duan[1,2,*], Shuang Hao[3], Baojun Liu [4] and Min Yang[5]
[1]Institute for Network Sciences and Cyberspace, Tsinghua University [2]QI-ANXIN Group [3]University of Texas at Dallas
[4]Department of Computer Science and Technology, Tsinghua University [5]Fudan University

ABSTRACT

HTTPS is principally designed for secure end-to-end communication, which adds confidentiality and integrity to sensitive data transmission. While several man-in-the-middle attacks (e.g., SSL Stripping) are available to break the secured connections, state-of-the-art security policies (e.g., HSTS) have significantly increased the cost of successful attacks. However, the TLS certificates shared by multiple domains make HTTPS hijacking attacks possible again.

In this paper, we term the HTTPS MITM attacks based on the shared TLS certificates as *HTTPS Context Confusion Attack (SCC Attack)*. Despite a known threat, it has not yet been studied thoroughly. We aim to fill this gap with an in-depth empirical assessment of SCC Attack. We find the attack can succeed even for servers that have deployed current best practice of security policies. 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 additional attacks including phishing and payment hijacking. Particularly, vulnerable HTTP headers from a third-party server are exploitable for this attack, and it is possible to hijack an already-established secure connection.

Through tests on popular websites, we find vulnerable subdomains under 126 apex domains in Alexa top 500 sites, including large vendors like Alibaba, JD, and Microsoft. Meanwhile, through a large-scale measurement, we find that TLS certificate sharing is prominent, which uncovers the high potential of such attacks, and we summarize the security dependencies among different parties. For responsible disclosure, we have reported the issues to affected vendors and received positive feedback. Our study sheds light on an influential attack surface of the HTTPS ecosystem and calls for proper mitigation against MITM attacks.

CCS CONCEPTS

• **Security and privacy → Security services**; **Network security**.

Φ Both authors contributed equally to this research.
* To whom correspondence may be addressed at zxf19@mails.tsinghua.edu.cn and duanhx@tsinghua.edu.cn.

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417252

KEYWORDS

HTTPS Hijacking Attack; TLS Certificate Sharing

ACM Reference Format:
Mingming Zhang[1,Φ], Xiaofeng Zheng[1,2,Φ,*], Kaiwen Shen[1], Ziqiao Kong[2], Chaoyi Lu[1], Yu Wang[1], Haixin Duan[1,2,*], Shuang Hao[3], Baojun Liu [4] and Min Yang[5] . 2020. 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 (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372297.3417252

1 INTRODUCTION

HTTPS, based on TLS and PKI, is designed for channel-oriented security[56], and provides authentication,confidentiality and integrity for web visits. Recent data has shown that HTTPS has been widely adopted on the Internet [6], and mandated for high-profile websites. For example, Google reports that over 94% of its traffic is now using HTTPS [5].

Given the wide usage of HTTPS, network adversaries are aiming at breaking its end-to-end protection. Specifically, several man-in-the-middle (MITM) attacks have been proposed to intercept secure connections, such as SSL Stripping [51] and SSL Sniffing [49]. To mitigate such threats, websites can deploy security practices, such as HTTP Strict Transport Security (HSTS) [37]. Moreover, web browsers have improved their UI to show secure connections (e.g., a "lock" sign or "insecure" alarm), and report authentication errors for users to notice interception behaviors. These countermeasures have made HTTPS traffic hijacking difficult.

Recent studies have shown that shared TLS certificates (Certificate Sharing) make HTTPS hijacking attacks possible again. Delignat *et al.* studied an origin confusion attack on virtual hosts [31] exploiting the wildcard or multi-domain TLS certificates. Based on the idea, another work [14] further provided more examples where attackers can load malicious content using compromised webpages. However, prior work only exhibited several attack cases, and there has been no systematic empirical research, neither examining the client-side (e.g., browsers' behavior) nor considering the complex network environment.

In the remainder of the paper, we call the HTTPS MITM attacks leveraging shared TLS certificates as *HTTPS Context Confusion Attack (SCC Attack)* based on their consequences. The research questions we seek to answer include: (1) Under what circumstances do the shared TLS certificates bring threats? (2) How to discover potential websites at a scale that are vulnerable to SCC attacks? (3) How severe is the impact in the real world, and how many popular sites are SCC-attack-vulnerable? To answer these questions, we

conduct a large-scale empirical study on SCC attacks. We demonstrate that TLS certificate sharing imposes a risk of intercepting HTTPS traffic, even when strict security practices are deployed by websites.

In more detail, we delve into the real-world influences of SCC attacks. First, we show attackers can launch such attacks to bypass state-of-the-art HTTPS security policies, such as HSTS, by using the inconsistency of HTTP headers (Section 4.1). Second, we show how attackers can disrupt HTTPS-protected user actions, such as online payments and file downloads, in complicated connection status. We find that even with a secure connection established between a client and server, HTTPS hijacking is still possible (Section 4.2). Third, we test on browsers notifications and behaviors during SCC attacks (Section 4.3). Then, to evaluate the real-world impact of the attack, we propose methods on discovering SCC-attack-vulnerable servers in the wild based on active scanning (Section 5.1). Last, we perform a measurement study on popular web servers (Section 5.2) to show the threat scale.

Our results show that certificate sharing is prevalent (over 86.82% of certificates we probed from Internet are multi-domain or wildcard ones), and that 126 apex domains (e.g., live.com, alipay.com, jd.com) of Alexa Top 500 have vulnerable subdomains. Among these, the apex domains under Alexa Top 100 have the most severe impact due to a large number of shared certificates (52% of all fetched multi-domain certificates). Moreover, we have analyzed the security dependencies among multiple parties and found the flawed implementations on one server can affect the security of a great many parties.

For mitigation, we provide feasible recommendations for modern browsers, such as warning users of the changes in the secure context. We have been reporting the issue to affected vendors and CNNVD[1], and have received responses from five of them: CNNVD, Alibaba, and JD have confirmed our issues, and we are still discussing with Microsoft and Netease for more technical details.

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

- We conduct an empirical analysis of SCC attacks that leverage shared TLS certificates. Through this attack, we find an adversary can downgrade established secure connections to plaintext, bypass HTTPS security policies, and disrupt HTTPS-protected services. The typical attack scenarios include payment hijacking, downloading hijacking, and website phishing.
- We evaluate SCC attacks from the client side, and analyze popular browsers' behaviors to attacks in different scenarios. We show the weakness of tested browsers against the SCC attack and propose enhancements to mitigate the issues.
- We propose methods to discover SCC-attack-vulnerable websites at scale, and perform a large-scale measurement study on its real-world impact. Our results show that 25.2% subdomains under Alexa Top 500 sites are vulnerable to the SCC attack, which span 126 apex domains. Moreover, we uncover the security dependencies among business parties.

[1]China National Vulnerability Database of Information Security

```
Certificate:
   Data:
       Version: 3 (0x2)
       Issuer: C=US, O=DigiCert Inc, CN=DigiCert SHA2 Secure Server CA
       Validity
           Not Before: Nov 28 00:00:00 2018 GMT
           Not After : Dec  2 12:00:00 2020 GMT
       Subject: CN=*.example.com, …
       …
       X509v3 extensions:
           …
           X509v3 Subject Alternative Name:
               DNS:a.example.com, DNS:b.example.com, DNS:c.example.com
           …
```

Figure 1: Example of the Shared TLS Certificate

2 BACKGROUND

2.1 HTTPS and Connection Security

On top of TLS, HTTPS provides communication security for web visits, which has been widely adopted by high-profile websites [56]. Besides encryption and integrity, it also provides authentication to protect against man-in-the-middle attacks, which rely on TLS certificate validation. For the convenience of issuing, managing, and revoking, TLS certificates can be shared by multiple subjects [29, 58]. As shown in Figure 1, a shared TLS certificate uses the Subject Alternative Name (SAN) extension to include multiple names or addresses. If none of the names are matched, or other invalid reasons(e.g., expired, self-signed, or malformed) occur, user agents will show authentication warnings to users. In contrast, once a certificate is validated, a trusted relationship can be established between the client and the server. Then all subsequent data are exchanged using the encrypted connection.

For enhancing HTTPS security, several mechanisms are adopted by modern browsers and popular websites and implemented through HTTP security headers [11, 52]. The headers are typically the instructions declared by servers to enforce the security policies in browsers [11, 37]. For instance, Strict-Transport-Security header forces browsers to interact with the server using only HTTPS connections during a period (max-age), which is a core of the HSTS policy. However, these headers can not fully guarantee security due to the implementation issues both on the client-side and the server-side.

2.2 SSL-Stripping-Based Attacks

Over the past few years, MITM attackers have become the main adversaries to SSL/TLS [39]. One of the most representative attacks is *SSL Stripping Attack*, which is introduced by Moxie Marlinspike, and attempts to bypass SSL/TLS [51]. In this attack, the adversary needs to intercept the initial HTTP connection when the user accesses the website for the first time, then delivers the request to the remote server. When he receives a 301/302 redirect to an HTTPS URL, he can replace the secure links returned by the server with plaintext ones, and keep the mapping of the changes. In this way, he can control the specific secure page and downgrade HTTPS to HTTP. As a consequence, the attacker can observe the sensitive data of users from the middle [57].

Further, Leonardo developed a tool combining SSL Stripping and a malicious DNS tool [33], and the work is called *SSLStrip+* [15, 34]. By doing this, attackers can partially bypass the HSTS policy. After measuring on the deployment of HSTS, Li *et al.* introduced an *Enhanced SSL Stripping Attack* while clicking or submitting [46].

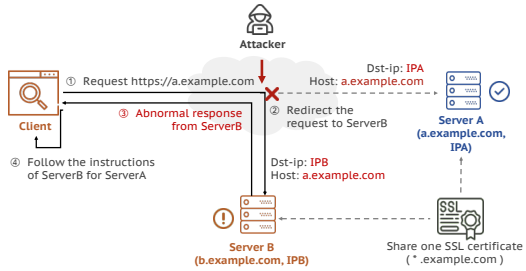

Figure 2: Threat Model

The basic idea is to use the XSS script at the front-end and replace HTTPS with HTTP. Similarly, Chen *et al.* demonstrated the possibility of stripping HTTPS by malicious proxies without breaking any cryptographic scheme [27]. They described that proxy could redirect script requests to insecure sites.

In such SSL stripping attacks, there needs an insecure initial request to be tampered. Meanwhile, there are no trusted TLS certificates provided by the remote server to protect the traffic. Hence, careful users can notice the abnormal validation status of certificates in the browser. Besides, an attacker can hardly launch an SSL Stripping attack after an updated security policy, HTTP Strict Transport Security (HSTS), is proposed.

3 THREAT ANALYSIS

The HTTPS MITM attacks we discuss rely on the design that multiple domains can share TLS certificates (see Figure 1). Using the shared certificates, adversaries can load pages to one origin from others due to the origin confusion issues [31]. However, the domains in the shared certificates do not always enforce the same security practices, some of which are misconfigured, especially in HTTP security headers. By rerouting HTTPS requests to the flawed servers, adversaries can invite their weak policies to the secure origins, and bypass the security policies of the secure servers.

In this paper, we specifically summarize and term the HTTPS MITM attacks using the shared TLS certificates as *HTTPS Context Confusion Attack (SCC Attack)*. Except for the origin confusion from the server-side, there is also secure browsing context confusion for programs and users from the client-side. It is because browsers treat the instructions as returned by the accessed origins and take insecure actions in the secure contexts.

SCC Attack Model. SCC attacks do not exploit vulnerabilities of the TLS protocol, and we assume the certificates are valid and issued by trusted CAs. In our threat model, we assume the attacker locates in the same LAN as the victim users, being able to *reroute the encrypted traffic* and *tamper with the plaintext data*. Studies have discovered numerous home routers with weak credentials [43]. As such, we assume the typical adversaries sniffer in local Wi-Fi or ethernet who share the same media. Similarly, attackers can also locate in the open Wi-Fi network (e.g., at coffee shops) without strong security protection (e.g., WPA2). Besides, attacks can be launched by malicious middleboxes (e.g., gateway, proxies) or even from ISPs and governments level as well.

SCC Attack Overview. Figure 2 illustrates our threat model that consists of four major components: (1) A victim user who browses a webpage; (2) A man-in-the-middle attacker who can maliciously reroute HTTPS traffic; (3) A webserver (ServerA), enforcing strict security practices (e.g., HSTS), which the user attempts to visit; And (4) another webserver (ServerB) with flawed security policies (e.g., misconfigured HTTP security headers). In particular, domains of ServerA and ServerB share one valid TLS certificate.

An adversary first terminates a TLS connection (e.g., by connection reset) between the client and ServerA, and maliciously reroutes the HTTPS request (①), which is originally from the victim to ServerA, to ServerB to re-establish a TLS connection (②). At this time, the browser still regards the context as under the connection with ServerA. The authentication of ServerB can be passed since the servers share one valid certificate. So, after receiving the flawed security configurations from ServerB (③), the browser will enforce the weak policies for ServerA (④). In short, the client initially visits ServerA but establishes a TLS connection with ServerB without authentication errors.

Differences with SSL-Stripping-based Attacks. Compared with previous SSL-Stripping-based attacks, there are several features special for SCC attacks because of certificate sharing. First, an SCC attack can succeed even when strict security policies have been deployed on the accessed websites. It is the third-party servers with shared certificates that communicate with clients. Second, it does not exploit initial plaintext requests. Hence, none of the strict HTTPS policies, such as the directives in HSTS mechanism or `upgrade-insecure-requests` in CSP, can prevent users from SCC attacks. In particular, SCC Attack applies to *established secure connections*. Third, it does not require installing root certificates to clients, since the connection is protected by a trusted and valid certificate. As a result, SCC attacks are undetectable to web browsers or applications. Last, web browsers do not display authentication errors during the traffic rerouting, so victim users can hardly notice SCC attacks.

4 SCC ATTACK IN THE REAL WORLD

In this section, we introduce the scenarios of bypassing HTTPS Security Policies in SCC attacks (4.1) and demonstrate the technical methods for attacking in the complex network environment (4.2). For each attack , the attacker needs to maliciously reroute the target HTTPS request to the certificate-sharing server, as mentioned in our model, which is possible through DNS spoofing [64, 65], IP/Port redirection[2] and ARP spoofing[36].

4.1 Bypassing HTTPS Security Policies

Based on the threat model, SCC attackers can exploit the header inconsistencies among servers that share TLS certificates. After systematic analysis, we find the following HTTP response headers are exploitable and can bring disturbing threats, which can directly downgrade secure connections and expose users to risk.

Scenario 1: Downgrading HTTPS to HTTP by Insecure Location Headers. In practice, web servers can upgrade HTTP connections via a 3xx redirect to an HTTPS URL by default to achieve maximum security. However, 3xx redirects can expose the communication to threats if the `Location` field is set with an insecure value (e.g., HTTP URL). Hence, adversaries can use the insecure 3xx redirects from servers to downgrade HTTPS traffic to plaintext, which we term as *HTTPS downgrading attacks*. This idea applies to SCC attacks in the certificate sharing scenarios.

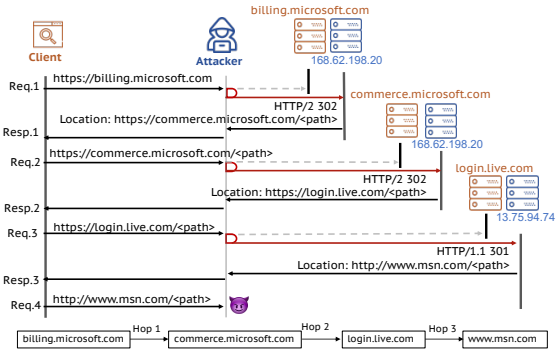

Figure 3: Multi-Hops Downgrading Attack. For each pair of servers, the TLS certificate provided by the right server include the domain name of the left one.

(1) *One-shot Downgrading Attack (Down-1).* Attackers can downgrade an HTTPS request by only rerouting the traffic once. Refer to Figure 2, if the ServerB with shared certificates returns a 3xx redirect to an HTTP URL, the attacker can launch a one-shot downgrading attack. The workflow goes as below.

First, when the client requests https://a.example.com, the attacker should reroute the request to the flawed ServerB (IPB) by means like DNS spoofing or TCP redirection. ServerB will not reject the request: if it has `Host` checking issues (e.g., no or incorrect Host checks) [24, 26] and handles the request of which the `Host` is unmatched by a default server [31]; or if it implements the domain aliasing and redirects the request to a valid domain, to avoid the frequent typos when users type into the address bar [9].

Then, the HTTPS connection is directly downgraded if ServerB returns an insecure 3xx redirect via the default handler. We have conducted traffic rerouting measurement by changing the destination IP addresses (Section 5). We find 28.42% of the responses are successful with 3xx redirects even if `Host` is unmatched, of which 17.17% are insecure redirects. For example, when we send the request https://login.live.com to an IP (13.75.94.74) of MSN, the server replies a 301 redirect with the `Location` set to http://www.msn.com/?<parameters>.

Last, if the user agent follows the insecure redirects, adversaries can tamper with the plaintext contents for further attacks, such as phishing, resource replacing.

(2) *Multi-hops Downgrading Attacks (Down-2).* Compared with the one-shot downgrading attack, multi-hops downgrading attacks mean that an HTTPS request can be downgraded by linking multiple 3xx redirects. Assume we want to intercept with the HTTPS traffic between the client and the target server. However, no server sharing TLS certificates with that domain returns an insecure 3xx redirect that can be used for one-shot downgrading. We find this domain still vulnerable to SCC attacks, since we can link up a series of 3xx redirects in the same way, and finally lead the request to the insecure URL.

Figure 3 depicts an example of the whole process. In this case, the client receives a redirect to https://commerce.microsoft.com/<path>, after the attacker maliciously reroutes the request, originally toward https://billing.microsoft.com, to 168.62.198.20 (Req.1-Resp.1). Then the attacker repeats the same rerouting action on https://commerce.microsoft.com/<path> and https://login.live.com/<path> (Req.2-Resp.3). When the client follows the third redirect, it sends a plaintext request (Req.4), which can be intercepted by the attacker.

Scenario 2: Bypassing HSTS Policy by Flawed Strict-Transport-Security Headers. Web servers may declare HSTS policy via the HTTP response header, `Strict-Transport-Security` (STS), forcing browsers to access them in HTTPS only. The deployment of HSTS policy grows in recent years [16, 54]. However, HSTS still has security issues due to misconfigurations [42, 46, 60] and partial adoptions [21, 61]. Adversaries can use these weak practices to bypass the protection of HSTS policy [59], which also applies to SCC attacks.

Similar to the attacks in Scenario 1, we assume ServerA has the full adoption of HSTS policy, but ServerB is not well-configured. An SCC attacker can bypass the HSTS-protection of ServerA by the flawed STS headers on ServerB. In total, attackers can launch the following three kinds of attacks.

(1) *Clear HSTS Policy for ServerA (HSTS-1).* In this attack, adversaries firstly need to find a ServerB that shares TLS certificates with ServerA and returns an STS header with the `max-age` directive set to zero. When the victim user visits ServerA, the attacker maliciously reroutes the HTTPS request to ServerB, which responses with `STS:max-age=0`. Subsequently, the browser disables the HSTS policy for ServerA, since it treats the flawed header as returned by ServerA. After that, the next time when the user accesses ServerA by typing the domain into the address bar without specifying the protocol, the browser will initially send out an HTTP request, which can be intercepted by the attacker.

(2) *Cancel HSTS Policy for ServerA's Subdomains (HSTS-2).* Recall that if the `includeSubDomains` directive is absent in a domain's STS header, browsers will not protect its subdomains through HSTS policy by default unless they are separately set. As such, a man in the middle can attack the subdomains of ServerA (a.example.com) using ServerB's flawed STS header in our threat model. Assume that ServerA is well-configured with HSTS policy, while ServerB's STS header does not include `includeSubDomains`. After rerouting the request to ServerB, the browser will update the HSTS policy received from it for ServerA, and will not protect ServerA's subdomains with HSTS policy by default. So the next time the user accesses one of ServerA's subdomains by directly typing in the domain, the insecure HTTP request that can be tampered will be sent out.

Besides, if both `max-age=0` and `includeSubdomains` appear in ServerB's STS header, the latter is ignored by browsers [37]. In this case, the browser will clear HSTS policy for ServerA, and will not set HSTS for its subdomains.

(3) *Decrease HSTS Validity Period for ServerA (HSTS-3).* As a less harmful scenario, attackers can shorten the HSTS cache time for ServerA in browsers, by rerouting the request to ServerB, of which the STS `max-age` is smaller than that of ServerA. For example, the HSTS validity period for ServerA is set to 31,536,000 seconds (one year) as standard [37], while the value of ServerB is only one day or less. As such, if users have not accessed ServerA during the short period (e.g., one day), the user agent will send out an HTTP request the next time (e.g., after one day) a user visits ServerA.

Discussion. The above issues can be mitigated if all domain names in the shared TLS certificate are added to the preload list [4,

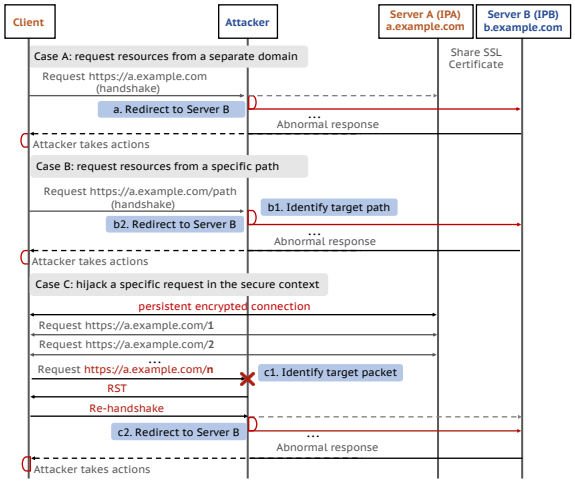

Figure 4: Timing for Hijacking the HTTPS Request

41]. However, HSTS preload is an opt-in project, and not all domains have been added in it yet.

4.2 Attacking in Different Connection Status

In the above scenarios, we prove that adversaries can launch SCC attacks at any proper time, unlike SSL Stripping attacks that can only succeed at the initial request. They can intercept HTTPS requests either in the process of the TLS handshake or from the middle of an established connection, making SCC attacks more general.

Driven from the attack model, adversaries meet the following technical challenges for successful attacks. First, they need to precisely identify the HTTPS request to be hijacked and find the timing for intercepting it, since the request can be delivered either over a new connection or in a persistent one shared with other requests. Second, they need to terminate and restart the connection secretly if it is keep-alive. Third, the interception must be unnoticeable to users so that the attackers can disrupt the user actions. Below we elaborate on the technical details to overcome the above challenges.

Timing of Attacks. Before attacking, adversaries need to force clients to establish a TLS connection with a third-party server, like the ServerB in Figure 2. Thus, they should find the proper time to terminate the current connection and reroute the HTTPS request.

In Figure 4, the first two cases present the scenario of one request per connection. We distinguish the interception for "a request to a separate domain" from "a request to a specific path". If a dedicated domain only serves the single resource to be hijacked (*case a*), attackers can reroute the request to a ServerB by poisoning the DNS cache. As such, the interception will not influence the requests for other resources served by different domains.

If multiple resources are served by one domain via different paths (*case b*), more steps are needed to identify the request first. The attacker can not reroute all traffic under that domain to ServerB; otherwise, users will notice the abnormal functionality of web pages. Instead, he should identify the particular HTTPS request toward the target path over the encrypted flow (*HTTPS Path Leaks*).

However, attackers can only intercept the traffic during the handshake process, for they need to let the client connect to ServerB by rerouting the request. Thus if a secure persistent connection is already established (*case c*), the attacker should trigger a legitimate restart of the TLS handshake (*TLS re-handshake*) without security warnings shown in the browser.

HTTPS Path Leaks. Previous work demonstrates that the cookies injected in an HTTP session will be attached to subsequent HTTPS connections [40, 66]. Although an acknowledged threat, this has still been troubling the public over the years. In 2018, Chen *et al.* proposed a new method to leak HTTPS paths via a side-channel [23], which has not been mitigated yet. However, they did not introduce the attack scenarios by leaking HTTPS paths. We can apply this method to our threat model. Assume an attacker attempts to intercept the request of https://a.example.com/n in Case C of Figure 4. Before that, he can first inject a long cookie to that path via an HTTP request to a.example.com, so that the TLS record size of the target request packet can be large. Then, he can sniff at the TLS layer, identifying that "large packet", which he is going to hijack.

TLS Re-handshake. Assume that a client has already established a TLS connection with ServerA. We term the process, in which the client initiates a handshake for a new connection after the connection is reset, as *TLS re-handshake*, which can also be considered as a new handshake. It is a new state of TLS connection, which is different from the TLS renegotiation [56]. From our test in Section 4.3, we find some browsers attempt to restart a handshake for the remaining requests when `Timeout` occurs, or an in-path entity (e.g., middlebox, MITM attacker) sends a `TCP RST` in the middle of a persistent connection, as shown in Figure 4 (case c). As such, the attacker can intercept the HTTPS request while the browser starts a TLS re-handshake toward ServerA, and reroute it to an exploitable ServerB. Then he can go on for further steps like the regular SCC attacks.

4.3 Browser Behaviors to SCC Attacks

While performing SCC attacks mentioned above, we wonder whether the browsers have policies to defend these issues or alert the users from being tricked. In this section, we test the browser behaviors when an SCC attack occurs. Because bypassing HSTS policy attacks are not directly reflected in browser behaviors except for updating the HSTS policy, we will not discuss them here.

Browser Security Notifications in an HTTPS Downgrading Attack. To inform users about the connection security, mainstream browsers display indicators (e.g., a lock, a shield) in the address bar. Meanwhile, they also warn the users when the authenticity of the remote server is failed. In our HTTPS downgrading attacks, browsers present no authentication warnings and show the certificates as "valid" in the address bar on account of the shared certificates that are trusted by the accessed domains. It has the users believe that they have been browsing in an encrypted and secure context.

As for the indicators in the address bar, there are subtle differences among scenarios. We divide all downgrading attacks into the following three kinds and test the browser behaviors separately.

First, downgrading the requests via the address bar or the hyperlink. In requests like these, the browser will navigate to documents or resources under the "new" context. So the address bar will directly turn to an HTTP URL after the downgrading attacks. After that, the browser's appearance depends on what attackers do to the plaintext request, which is similar to known website forgeries.

Table 1: The behavior of popular browsers in different OS when trying to trigger a TLS re-handshake

		Windows	MacOS	Linux
RST	Chrome	✓ **Re-handshake without warnings**	✓ **Re-handshake without warnings**	✓ **Re-handshake without warnings**
	Firefox	✓ Re-handshake without warnings. ImageB is loaded, but the request is pending all the time.	✓ Re-handshake without warnings. ImageB is loaded, but the request is pending all the time.	✓ Re-handshake without warnings. ImageB is loaded, but the second request is pending all the time.
	Edge	✓ **Re-handshake without warnings**	-	-
	Safari	-	✓ **Re-handshake without warnings**	-
Timeout	Chrome	✓ **Re-handshake without warnings**	No Re-handshake and second request, Network failure: ERR_TIMED_OUT	Re-handshake and ImageB is loaded. Both the server and client send Alert.
	Firefox	✓ Re-handshake without warnings, ImageB is loaded, but the request is pending all the time.	No Re-handshake. ImageB is not loaded, and the request is pending all the time.	Re-handshake and ImageB is loaded. Both the server and client send Alert.
	Edge	No Re-handshake. ImageB is not loaded, and the request is pending all the time.	-	-
	Safari	-	No Re-handshake and second request. Network failure: ERR_TIMED_OUT	-

[1] The cases with ✓can be exploited by attackers to trigger re-handshakes successfully.
[2] The light gray squares indicate the successful cases without any warnings.

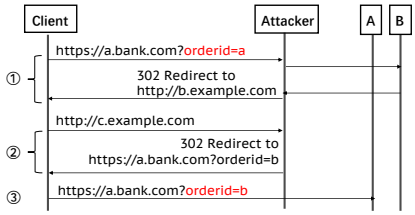

Figure 5: Recover the Context Back to HTTPS after the Downgrading Attack. A and B serve for a.bank.com and b.bank.com (only for example), which share TLS certificates.

Second, downgrading the requests and then upgrading the connection back to HTTPS after attacks. As presented in Figure 5, attackers can first downgrade the HTTPS request of interest using the shared certificates (①). Then they force the browsers to follow one or more 3xx redirects returned by themselves (②), and finally to request an HTTPS URL which recovers the context back to HTTPS (③). During the process of ②, attackers can forge 302 redirects and take malicious behaviors for tricking on the clients, such as injecting cookies or replacing parameters.

We have tested how browsers react to this kind of downgrading attacks. During the intermediate redirects (②), there are no changes in the appearance of the browsers, including Chrome, Firefox, Safari, and IE. The webpage and the address bar always show the contents and the address of the first HTTPS request, say, https://a.bank.com?orderid=a in Figure 5, because the clients do not receive the response bodies to be rendered. Only after the request of https://a.bank.com?orderid=b finishes, the URL and the page turn to this HTTPS content. It is difficult for users to notice the attack because the webpage jumping time is short, and the browsers show the secure context all the time.

Moreover, assume that the first request in ① is going to a non-rendering resource (e.g., installation packages, email attachments). After downgrading, in ②, the attacker forces the browser to request a malicious executable, which is fetched in an HTTPS context. During the download process, we find the browser has no changes, and the address bar stays at the original HTTPS request without any jumps. Thus, this kind of attack is transparent and undiscoverable to users.

Third, downgrading the requests for passive contents of the webpage. If the requested passive contents (e.g., image, video) are going to be rendered on the page, we find the Firefox in iOS still shows the "lock" icon after downgrading the request and replace the passive content, while the Chrome turns the "lock" to an "exclamatory mark", and Safari, as well as IE, shows no icon except for the URL.

Browser Behaviors to the Trigger of TLS Re-handshake. We have also tested how modern browsers handle the TLS re-handshake in different operating systems. We set up our test environment on two HTTPS servers (CentOS 6.9), ServerA and ServerB, which share the TLS certficate. Both of the servers serve two files, ImageA and ImageB. On ServerB, we accept all requests without checking the Host header. Then, we request these files from ServerA through JavaScript over a persistent connection. We send a TCP RST or trigger the Timeout in between two requests, to check whether the browser will start a new handshake and finish the second one. If it happens, we will reroute the second request to ServerB.

In the cases with ✓shown in Table 1, the browser will immediately start a TLS re-handshake without showing warnings, after the connection reset or timeout. Among these cases, we see that Chrome can start a new handshake and transparently load the resources from ServerB, after receiving a `RST` packet on different operating systems, as well as `Timeout` on Windows. Besides, in Firefox on all operating systems, the users can notice the attacks. Though some browsers can not load ImageB from ServerB after re-handshaking, it is still vulnerable if ServerB returns a 301/302 redirect toward an HTTP URL.

5 DISCOVERING VULNERABLE SERVERS IN THE WILD

5.1 Methodology

In our threat model, adversaries aim to attack the servers that may adopt full security settings, and they use the flaws of other servers that have security dependencies with the targets. As such, given a list of domain names to find out the vulnerable ones, we first need to collect the *related domains* that share TLS certificates with them. Then, we should send HTTPS requests to the tested domain names and crossly reroute the requests to the related domains that can be exploited (as shown in Figure 7). In this section, we introduce how to discover the exploitable servers in the wild.

Getting Related Domain Names. As aforementioned, one primary security dependency of domain names comes from the shared

TLS certificates, since each certificate can be trusted by all hostnames shown in CN or SAN fields, especially when there are wildcard domains. We assume the hostnames in one certificate may affect each other and group them together, no matter whether they have a real relationship in business.

Thus, we can parse the related domains from the TLS certificate datasets. For a comprehensive analysis, there are several options for certificate dataset: (1) *Active Scan.* We can conduct an active scan over IPv4 address space to collect the TLS certificates. However, only the default certificates in most servers can be fetched due to the lack of valid SNIs. (2) *Passive Dataset.* Another option is to parse the certificates from passive traffic. It shows the in-use TLS certificates from the real network environment, but it needs representative vantage points to sniffer for a long time, which is high cost and inefficient. (3) *Public Dataset.* A feasible option is to search in the public TLS certificate datasets, including Certificate Transparency (CT) logs, Censys, and Rapid7. However, they include massive expired certificates, in which there can be a large quantity of redundant data.

Rerouting HTTPS Requests. After mapping the domain names, we start to send HTTPS requests to them, while rerouting each request to the *related domains* in the same group as the example in Figure 7. The traffic rerouting process is done at the TCP and IP layers so that we can consider the TCP 4-tuple.

(1) *Switch the Server IP Address.* Refer to the threat model (Fig. 2), without regard to load balancing, each related domain (b.example.com) can be mapped to another single IP address (IPB). We can reroute the request of https://a.example.com to IPB, to let the client establish the connection with ServerB that might return vulnerable responses.

Here, we can choose two kinds of destination IP addresses to reroute the traffic. If the domain name of ServerA maps to multiple addresses via DNS (e.g., for load balancing), each of the addresses can be chosen because different servers may have inconsistencies in configurations. Alternatively, the resolved IP addresses of the *related domains* that share certificates are also optional. Because the certificates are shared and valid, browsers report no authentication errors after the rerouting.

(2) *Switch the Server Port Number.* Recall that the HTTPS service runs on the port 443 by default [56]. Alternatively, the TLS server can also listen on other TCP ports (e.g., 8443, 8843) if being specified [63]. Hence, there can be multiple servers, serving on different ports of the same IP address, that may share TLS certificates and have inconsistencies. Same as IP addresses, we can also reroute the request by switching the destination port number from the TCP layer.

(3) *Switch the Client IP Address.* We find one server's HTTP responses may vary from the client IP addresses located in different regions, which can still happen regardless of the DNS resolution. For example, a CDN edge server on one IP can serve different content for users from multiple countries. So the client IP address plays a decisive role during website access, which also applies to SCC attacks.

In this scenario, we give an attack model in Figure 6. First, the request of https://a.example.com is delivered to the attacker's server (the 1st-phase) that acts as a malicious client-side proxy and locates in another region, by means like spoofing WPAD or tricking the user

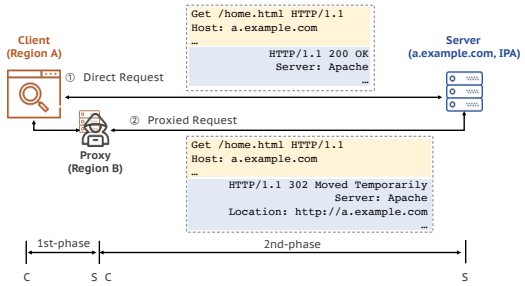

Figure 6: Client-IP Switching Model. The attacker and the client locate in different regions. And the server response them with different policies.

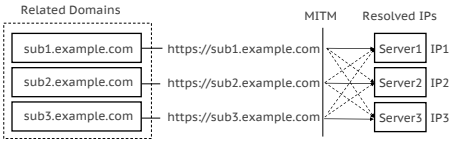

Figure 7: An Example of Crossing Requests

into using a malicious proxy. Then, the proxy server establishes the TLS connection with the remote server and sends the request to it (the 2nd-phase). In turn, the target server replies to the proxy server according to its geo-location, and the proxy forwards the response back to the client. If the response in the 2nd-phase has vulnerable security headers, it is possible to enforce the weak security policy for `a.example.com` in the victim's browser.

(4) *Take CDN into Consideration.* As a middle entity, Content Delivery Network (CDN) is widely deployed to improve the performance and the security of websites. It is geographically distributed and serves each customer's website with multiple nodes. However, we find configurations inconsistencies among different CDN nodes, which can be exploited to launch potential SCC attacks. Like the Barrel Principle, the node with weak configuration can affect the security of other nodes and customer's websites, since adversaries can reroute the traffic to that flawed node and exposed the connections to threat.

5.2 Real-world Impact: Measurement Analysis

To better understand the impact of the preceding issues, we conduct a systematical measurement on popular websites to discover exploitable servers using the above methodology.

5.2.1 Iterative Scan. Given a list of domain names to be tested, first, we parse their *related domains* from the TLS certificates on 8 CT log servers of Google and Symantec as much as possible. Because there are massive wildcard domains in the certificates, we also collaborate with our industrial partner and extract the concrete subdomains of the tested domains from their passive DNS database as a supplement.

Next, we query the domain names and get their IP addresses after filtering out the invalid domain names, including the non-existent and the expired ones. For each group of related domain names and their resolved IP addresses, we try to crossly send HTTPS requests to each IP with the Host and SNI set to each domain name, acting as rerouting the requests to different server IPs as shown in Figure 7. Here, the IP addresses represent the servers that provide the shared TLS certificates with the tested domains.

Table 2: Overview of Dataset and Scale of Affected Domains for All Attack Scenarios within Alexa Top 500 Apex Domains.

Alexa Rank			1-100	101-200	201-300	301-400	401-500	Total
Multi-domain Certificates			4,630	1,400	1,017	1,120	725	8,892
All extended FQDNs			83,367	67,262	41,296	60,325	39,977	292,227
FQDNs with HTTPS			12,453	5,695	5,113	5,904	5,152	34,317
# Affected Apex Domains	HTTPS downgrade (C1.1)	Down-1	36	19	19	20	20	114 (22.8%)
		Down-2	11	4	4	4	4	27 (5.4%)
	HTTPS downgrade (C1.2) (Filter out HSTS)	Down-1	32	17	16	18	15	98 (19.6%)
		Down-2	7	4	1	3	4	19 (3.8%)
	HSTS Bypass (C2)	HSTS-1	3	2	0	0	0	5 (1%)
		HSTS-2	7	6	5	2	1	21 (4.2%)
		HSTS-3	7	7	7	7	3	31 (6.2%)
	All	C1.1+C2	37	21	21	24	23	126 (25.2%)
		C1.2+C2	34	19	19	24	18	114 (22.8%)
# Vulnerable FQDNs	HTTPS downgrade (C1.1)	Down-1	826	434	352	476	354	2,442 (7.12%)
		Down-2	266	48	151	98	24	587 (1.71%)
	HTTPS downgrade (C1.2) (Filter out HSTS)	Down-1	590	391	268	174	328	1,751 (5.10%)
		Down-2	119	48	5	95	24	291 (0.85%)
	HSTS Bypass (C2)	HSTS-1	23	19	0	0	0	42 (0.12%)
		HSTS-2	37	24	14	19	1	95 (0.28%)
		HSTS-3	54	24	28	43	12	161 (0.47%)
	All	C1.1+C2	1,087	497	391	572	371	2,918 (8.50%)
		C1.2+C2	725	458	297	304	356	2,140 (6.24%)

While performing the HTTPS requests, we also actively and dynamically fetch the TLS certificates and add the newly parsed domain names into our testing dataset. We repeat the above scanning process until no new domain names can be found out. In this way, we can fully get the domain names that have security dependencies caused by the shared certificates. We determine a domain as "SCC-vulnerable", if any server (IP) that provides the shared TLS certificate replies with the exploitable headers as we mentioned in Section 4.

Limitation. In this iterative scanning process, we can fully discover all related domain names that might be exploitable in the certificate sharing scenario. However, we need to send requests of each related domain to each IP crossly, so the required HTTPS requests shall increase exponentially.

5.2.2 Dataset. Considering the performance, we only use Alexa Top 500 apex domains as the seed to be tested.

Domain Names. By parsing subdomains from the CT logs and the passive DNS traffic, we get 283,311 subdomains under the 500 apex domains. After the iterative scan, we extend the dataset to 333,640 subdomains spanning 5,780 apex domains, while 292,227 of them are under Top 500 apex domains (Table 2). The added domain names are all parsed from the newly fetched TLS certificates.

Based on the domain set, we collect 6,765,333 domain-ip pairs for *HTTPS requests rerouting* test, and 4,503,824 (66.57%) pairs have passed the TLS certificate validation while connecting. Here, each domain-ip pair represents an HTTPS request. The HTTPS requests with the successful or the redirection status code (i.e., 2xx, 3xx) cover 59,713 concrete subdomains spanning 4,043 apexes, and 34,317 of these subdomains are under Top 500 apex domains.

The 34,317 subdomains here include all related domains of the top 500 apex domains we can get. In Table 2, we have shown the number of subdomains in each partition by Alexa Rank. In our analysis, we mainly focus on these domain names to discover the SCC-vulnerable sites and measure the threat scale.

TLS Certificates. During the iterative scanning, we collect 12,734 valid certificates shared by the subdomains under Alexa Top 500. Over 86% are multi-domain certificates, indicating that certificate sharing is common in reality. Besides, from all partitions in Table 2, we find the number of multi-domain certificates has a strong correlation with FQDNs' amount. As such, under a certain of apex domains, the more of subdomains, the more of the shared certificates.

5.2.3 Analysis and Findings. From the measurement results, we find that the shared TLS certificates can expose popular domains to threats. Besides, we demonstrate the security dependencies among multiple subjects or organizations.

Overview of Threats. By analyzing the responses of HTTPS requests, we find 2,918 (8.50%) subdomains under 126 (25.2%) Alexa Top 500 apex domains are vulnerable to SCC attacks. In Table 2, we give an overview of the vulnerable subdomains for all attack types.

Downgrading HTTPS to HTTP. According to the statistics, HTTPS downgrading issues account for a large proportion of all exploitable cases (Table 2, C1.1). By rerouting HTTPS requests, we find the secure connections toward 2,442 subdomains under 114 apex domains can be attacked by a one-shot downgrade. The affected apex domains include the famous ones like baidu.com, amazon.com, and tmall.com. For example, when rerouting the requests of the subdomains of tmall.com to 47.88.135.224 (AS45102, Alibaba), we can always receive a 302 redirect to http://err.taobao.com/error1.html. The connection is directly downgraded to a plaintext one via this single roundtrip. Additionally, 587 subdomains under 27 apex domains (e.g., office.com, linkedin.com, and microsoft.com) can be downgraded through multi-hops downgrading attacks.

For both one-shot and multi-hops downgrading attacks, there are the security dependencies among different domains caused by 3xx redirects returned by the servers that share certificates. Like the example we show in Figure 3, the security of billing.microsoft.com can be influenced by that of www.msn.com after three hops. Hence, we further delve into all targets of the 3xx redirects. We compare the domain in the Location field with the original hostname and show the results in Table 3. In total, we find 16.56% requests are redirected to the same domain name via the Location field, while 49.12% to other FQDNs under the same apex domain. Besides, there are also 34.32% cases go to the third-party domains under different

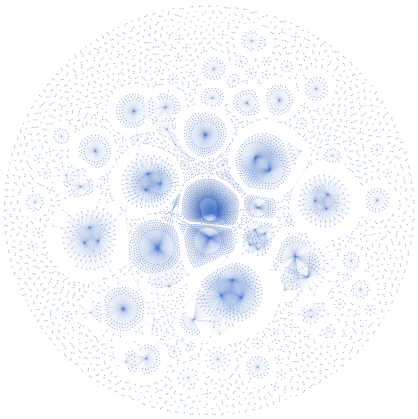

Figure 8: The Security Dependencies among the Tested FQDNs. Each node represents one FQDN, and each link shows the security dependency between two domain names.

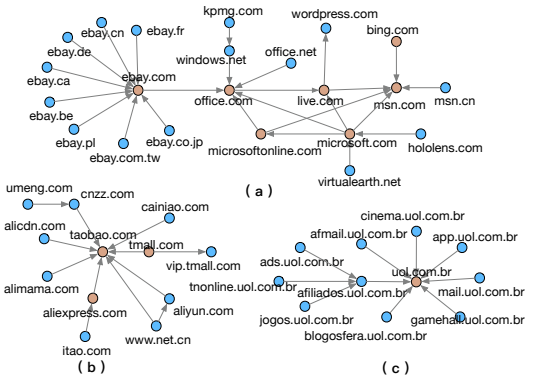

Figure 9: The Security Dependencies among the Apex Domains. Given a pair of domain: A->B, we mean the security of A can be influenced by that of B.

apexes. For instance, maps.live.com can be finally redirected to www.msn.com.

However, there is an exception for HTTPS downgrading attacks. Referring to Figure 2, the plaintext request will not be sent out, if ServerB returns a 302 redirect to http://c.example.com, and the browser has already enforced HSTS for c.example.com. It means the TLS connection can not be downgraded, though ServerB returns an insecure redirect. So we filter out the downgrading cases of which the 3xx redirect target domains are configured with HSTS. After that, there are still 1,751 subdomains that can be one-shot downgraded by traffic rerouting (Table 2, C1.2). Specially, we find 22 subdomains that can not be one-shot downgraded anymore due to the 3xx targets' HSTS policy, but they can still be downgraded after over two hops attacks. It shows the multi-hops attacks can increase the possibility of HTTPS downgrade attacks.

Bypassing HTTP Strict Transport Security. For bypassing HSTS policy, we list the measurement results in Table 2 (C2) based on the attack types. Within Alexa Top 500 websites, we find a total of 271 FQDNs under 43 (8.6%) apex domains that can be influenced by the STS header of other domains that share TLS certificates. Notice that there are overlaps of the apex domains among three types of attacks, since the vulnerable subdomains for each attack type may be under the same apex domain. In more detail, we can directly clear the HSTS policy for 42 subdomains under five apex domains (C2, HSTS-1), such as gaode.com, baidu.com, and aliexpress.com, by rerouting the HTTPS requests to the flawed servers.

Summary. From Table 2, we see that the total number of domain names for all types of attacks has a strong correlation with that of the multi-domain certificates. The reason is that SCC attacks are caused by the flawed servers that provide the shared TLS certificates, namely, the multi-domain or the wildcard certificates. In terms of the domain ranking, we find that Alexa Top 100 domain names have the most problematic subdomains, accounting for 37.25% of all vulnerable FQDNs and 36% of all affected apex domains we found. We infer it is caused by a large number of shared certificates.

Security Dependencies among Multiple Parties. In our threat model, the servers from different corporations or paries have the security dependencies caused by the shared TLS certificates. For example, in the HTTPS downgrading scenario, the security of the accessed domain name can be influenced by both the domains that share TLS certificates and the target domains of 3xx redirects. If a domain name has not deployed security policies well, all domains that depend on it may be exposed to threats.

In order to show the security dependencies in the certificate sharing scenario, we extract all domain pairs, including the requested domain (Host) and the related domain, as well as the Host and the 3xx target domain. As a result, we find converging clusters among FQDNs, as shown in Figure 8. The in-degree of each node represents the number of domains that depend on it. We find the center node with the maximum in-degree is pages.ebay.com, which is relied on by over 900 concrete domain names. If the domains at the convergent nodes are vulnerable, there will be potential security threats for those around them.

Zooming into the apex level, we demonstrate the dependency link samples among the affected apex domains in our measurement (Figure 9). Overall, we can summarize the following relationships among parties that have security dependencies: (1) *The sub-domain and the apex-domain of the same corporation.* As Figure 9 (c) shows, the subdomains of uol.com.br can be downgraded by the server of uol.com.br. (2)*The trans-regional services of the same corporation.* For instance, eBay registers multiple apex domain names (e.g., ebay.cn, ebay.de, ebay.jp) for the service of different regions. We find these domain names share TLS certificates with ebay.com. Then we try to reroute the HTTPS request, originally toward their regional subdomains, to an eBay's IP address like 66.135.201.205, and we receive a 302 redirect to http://pages.ebay.com/messages/CN_page_not_found.html. Therefore, the traffic can always be downgraded to plaintext. (3) *The subsidiary and the holding company.* In (b) and (c), we see that domains of Office, MSN, and Bing rely on the security of Microsoft's domain, while Xiami, Tmall, and AliExpress rely on Alibaba Group. (4) *Business partnership or investment relationship*, such as WordPress/KPMG and Microsoft, Merrill Lynch (ml.com) and Bank of America, Umeng and Alibaba Group. (5) *Other relationship, like the service providers and their customers.* For example, we find that a domain of NIH, www.myitsm.nih.gov, can be redirected to www.servicenow.com while we reroute the request to 137.187.0.26 (belongs to NIH). The latter is involved in the development of the former.

Table 3: Statistics on the Targets of 3xx Redirects. Here, the targets are domains shown in the `Location` field.

3xx Redirected to	Same FQDN	Same apex-domain	Other apex-domain
All redirects	16.56%	49.12%	34.32%
Filter out the targets that enforce HSTS	17.22%	46.73%	36.05%

Besides, there are also dependency links across multiple parties, such as ebay.com->office.com->live.com. In summary, TLS certificates are widely shared by the parties that have the above relationship, which is necessary for business. However, the threat can be on a large scale if some of the domain names have security flaws as we introduced in Section 4.

5.3 Case Study

Through our measurement work, we have found thousands of vulnerable domain names. In this section, we give the case studies and show that SCC attacks may occur in scenarios that are closely related to our daily lives.

5.3.1 HTTPS downgrading attacks by switching the destination IP. These attacks depend on the insecure 3xx redirects returned by the certificate-sharing servers. Below, we give a brief description of each case to show how to attack the HTTPS request of interest without being noticed.

Online Payment Hijacking. E-commerce payment, through traditional bank accounts or online payment services (e.g., PayPal, Alipay), brings convenience for online purchases. However, we find that adversaries can attack the payment process using shared TLS certificates. Here is a case of meddling with the payment process while shopping on JD.com, a popular E-commerce website in China, from the middle of the secure context.

Case 1: Hijacking the Process of Bank Payment. In the payment process of online shopping, the website requires buyers to choose which bank they wish to pay through. After that, JD will send a POST request with the payment information to a single domain, channel.jdpay.com, which then delivers the attributes to the server (ibsbjstar.ccb.com.cn) of China Construct Bank (CCB). However, the HTTPS request, from the client to channel.jdpay.com, can be downgraded if an attacker reroutes it to the IP of www.yhd.com. It is because www.yhd.com shares a TLS certificate with *.jdpay.com, and it returns a 302 redirect toward http://www.yhd.com?l=1&err=53 if we reroute the request of https://channel.jdpay.com to it.

When the browser follows the insecure redirect, the attacker can hijack the HTTP request and forge another 302 response to the client. In this response, he sets the `Location` field as a payment URL of the attacker's order, https://ibsbjstar.ccb.com.cn/CCBIS/B2CMainPlat_03_EPAY?<parameters-of-attacker's-order>. Hence, the victim user may pay for the attacker's bill after the browser renders this replaced payment webpage. During this attack, the browser shows no alerts to the users, since the "lock" icon shows the context is secure and protected by a valid certificate.

Download Hijacking. Another hijacking scenario occurs when we download the non-rendering resources (e.g., email attachment, installation packages) from a website. Attackers can downgrade the HTTPS requests and replace the resource with another one, such as a malicious executable. This kind of attack is more concealed since the address bar remains unchanged if the response content of a subsequent request is not rendered on the webpage.

Case 2: Replacing the Installation Package of Xiami Music. Xiami Music is a popular music streaming service owned by Alibaba Group. To download an installation package for Windows version from its homepage, an HTTPS request is sent to a URL[2] under the domain name, files.xiami.com. However, we find the server of taobao.com shares a TLS certificate with *.xiami.com, which is exploitable for SCC attacks. Thus, an attacker can reroute the downloading request to the server of taobao.com, which returns a 302 redirect to http://err.taobao.com/error1.html. After that, he intercepts the HTTP traffic and returns a malicious executable to the innocent user instead, without any warnings shown in the browser. Throughout the process, the address bar always stays at the URL, https://www.xiami.com, and shows a secure lock icon.

Website Phishing. Adversaries can also forge the whole webpage and launch a phishing attack, to meddle with users' actions or leak their private data. It occurs when users click a hyperlink to open a new webpage or access a website through the address bar.

Case 3: Phishing on Microsoft or Netease Websites. The login page of Microsoft is hosted on https://login.live.com/login.srf?<parameters>, while www.msn.com shares the TLS certificate with *.live.com. The attacker can reroute that HTTPS request to the server of www.msn.com, which then returns a 301 redirect to an HTTP URL[3]. It is because the server of MSN does not reject the request when the Host header is unmatched but hands it over to the default virtual host. When the browser follows the 301 redirect, the network attacker can hijack the plaintext traffic and replace the page. For instance, he can forge a login webpage and trick users into handing over sensitive information like their accounts. According to the relation group shown in Figure 9, we find the login processes of MSN and Bing are all SCC-vulnerable since Microsoft authorizes all of them via login.live.com.

Similarly, the request for Netease Enterprise Mailbox, qiye.163.com, can also be downgraded if we reroute it to www.163.com. In this site, attackers can launch a phishing attack as well.

5.3.2 HTTPS downgrading attacks by switching the client IP. Different from the cases in Section 5.3.1, here we request one server from different regions.

Case 4: Intercepting the HTTPS Traffic toward Sina Weibo. Sina Weibo is the most famous microblogging website in China, of which some services are hosted on the Akamai CDN. We separately send the request, https://www.weibo.com/, to a CDN IP address, 180.149.134.141, from the client IPs located in multiple regions. Then, we get different response headers from that server, as shown in Table 4 in the appendix. The IP address in China receives a `200 OK`, while those in Japan and the US get 302 redirects to an HTTP URL. It is because the CDN edge server provides the trans-regional

[2]The full downloading URL in Xiami is https://files.xiami.com/webh5/files/xiamiWin/068ba49b0603140cabe09fc34771cd70.xiami7.2.7.exe

[3]The full URL is http://www.msn.com/?redirfallthru=https%3a%2f%2fsignup.live.com%2f%3f

services, and it redirects the request to the specific URLs based on the user's location.

In this case, when Chinese users browse https://www.weibo.com/, network attackers can force the requests to go through the controlled proxy servers in Japan or the US, and finally reroute them to 180.149.134.141. The proxy servers then forward the insecure responses back to the users. If the client follows the 302 Redirect and sends out the plaintext request, the attacker can intercept it an go on for phishing attacks or injecting the advertisement.

5.3.3 HSTS Bypassing Attack. As aforementioned in Section 4.1, clearing HSTS policy by setting max-age to zero is more dangerous than the other two scenarios. Here, we show a case of this kind.

Case 5: Clearing the HSTS Policy for AutoNavi. AutoNavi Software is a web mapping, navigation, and location-based services provider, which is known as Gaode in China. When requesting https://gaode.com, we find the server returns a response with the HSTS max-age set to 31,536,000 seconds. However, the HSTS validity time of Gaode is updated to zero, if a network attacker reroutes the request to another IP address, 106.11.223.100 (owned by Alibaba Group). It is because the server hosted by this IP address returns `strict-transport-security:max-age=0` in its headers. This header can clear Gaode's HSTS cache at the client-side, since the browser treats it as returned by `gaode.com`. Therefore, the next time users access Gaode's website by typing the domain directly without the protocol, the browser will initiate an HTTP request first, which is considered dangerous.

6 DISCUSSION

6.1 Root Causes

Key Reason 1: Certificate Sharing. Consider convenience and cost, the multi-domain and the wildcard certificates are widely used by modern servers (one certificate is valid for multiple domains). Besides, sharing the same certificate is also widespread due to CDNs, virtual hosts, associated services, and commercial cooperation (multiple servers provide one certificate). In short, certificate sharing plays a vital role for certificate management and grows at a scale. However, the shared TLS certificates can bring security dependencies to different servers. The subjects in these shared certificates are linked together like familiar strangers. The flawed implementation on one server may expose other domains to threats, just like the barrel effect. As an essential cause for SCC attacks, the shared TLS certificates need more public attention and should be well deployed.

Key Reason 2: Problematic Implementations of Security Policies among Different Parties. Due to the intricate details in the protocols and policies of HTTPS, misconfiguration widely occurs in today's implementation, and HTTP header inconsistency is common among multi-servers as well. First, servers may not implement strict Host checks [26, 31]. For example, servers may not reject the request when the `Host` header is not matched, but return a `301/302 Redirect` or even `200 Accept` via the fallback server instead. In our threat model, such a ServerB with shared certificates can give attackers the chance to attack the benign ServerA. Second, some servers have not enforced the best practice of security policies, such as HSTS and CSP. It is the misconfigured HTTP headers and the vulnerable response contents that are used for SCC attacks on secure traffic. Besides, the pervasiveness of inconsistencies among servers still affect some high profile websites [25, 48, 52]. Therefore, developers should pay more attention to the implementation details for security.

Key Reason 3: Absent Policies for Maintaining the Secure Context. An SCC attack can succeed because (1) the traffic is rerouted to a third-party server, and (2) that server has security flaws. However, applications from upper layers do not know the change of the peer host from transport and network layers. User agents can only authenticate servers and check the hostnames by TLS certificates, which are unreliable in the certificate sharing scenario. As such, a policy is needed to let clients know the server's hostnames they are talking with and take actions to maintain the secure context. Though HSTS can help to mitigate the HTTPS downgrading attacks, it is hard to enforce full HSTS policy for all sites, even Google' servers [4]. Besides, the HSTS policy can be bypassed as well.

6.2 Mitigation

Browsers should enhance the following policies to protect the browsing context. *First, add a notification for the insecure changes of context.* As we discussed in Section 4.3, attackers can have the browsing context recovered to the secure one after launching the HTTPS downgrading attacks (Figure 5). However, modern browsers will not warn the users of the downgrading process in the middle of the attack, and they present "secure" icons which expose the users to a threat. As such, we recommend browsers to show security indicators about the changing of the browsing context. In more detail, the HTTPS connection that is redirected from an HTTP one should not be shown as "secure", though it is protected by a trusted TLS certificate. For example, the request for https://a.bank.com?orderid=b in the third process in Figure 5 should be marked as "not absolutely safe" or other warnings like that.

Besides, there is a trend that Google Chrome may stop HTTP file downloads from the HTTPS webpage [3]. Based on this, we further recommend that browsers should also block the secure download, the context of which has been changed like "HTTPS->HTTP->HTTPS". It is because the resources downloaded through the final HTTPS request of the link can be a malicious one replaced by the attacker.

Second, block all mixed contents. In SCC attacks, the attacker may hijack the request for a subresource of a webpage, so that there will be mixed contents in the browser. While mainstream browsers block the mixed active contents [8, 12, 17, 18], the passive contents have been considered as less disturbing since they may not be relevant to security or privacy. However, we find the QR codes, mostly used for the login or the payment process in China, are presented as images that are the passive contents, and SCC attacks can downgrade the requests for them.

Up until now, mixed passive contents (e.g., image, video) have not been blocked by all browsers yet. Only Chrome that releases an update to block all mixed content (active and passive), ensuring that HTTPS pages can no more load any HTTP resources[10]. Considering the security, we recommend all browsers to block any insecure requests from the HTTPS context.

6.3 Responsible Disclosure

We have submitted the issues caused by the shared TLS certificates and a demo case to the China National Vulnerability Database of Information Security (CNNVD). It rates the issues as the medium risk. Meanwhile, we have been in the process of contacting all of the affected vendors (nearly one hundred). Up until now, we have contacted the individual vendors of which we showcase in the paper, and received the following responses. **(1) JD**: We report the issues and the payment hijacking attack to JD's SRC, which has confirmed the exploitation as a threat and adopted updates to their website. **(2) Alibaba**: Alibaba has confirmed the attacks that we report as well, and it is still in the process of mitigating the issues. **(3) Microsoft and Netease**: They concern about the reported issues and require the PoC of the attacks. Since we have not finished the demo videos, we still need to communicate with them continually. **(4) Sohu**: We also report the certificate sharing issues to Sohu SRC, but it has not replied yet. Moreover, we have been discussing the issues and mitigations with Chrome. They confirm the threat model and reply that their new plan, applying mixed content checks to all redirects when loading a resource, will help to mitigate these issues. However, this update can not cover all attack scenarios, as we mentioned in Mitigation 1. As such, we will further contact other affected vendors to fix the issues and discuss the mitigation methods with the browser vendors.

7 RELATED WORK

HTTPS Security and Stripping Attacks. HTTPS security has been discussed for years. More and more researchers focus on the adoption and deployment of HTTPS [35, 38, 45, 55]. Previous works have introduce a range of attacks on HTTPS, such as the attacks on the cipher weakness [1, 19]. Also, the attackers can rollback the TLS version, conducting the downgrade attacks [7, 20]. Though policies have been proposed to enhance HTTPS security, users have still been exposed to potential MITM attacks, especially when there are intermediate entities like interception software [30, 32], malicious proxies [28, 30, 32, 62], and content delivery networks (CDNs) [47] in the path. Regardless of protocol defects, a man in the middle can simply strip HTTPS protection, by SSL Stripping attacks [50, 51, 53] or PBP attacks [27]. For defending such stripping attacks, optional mechanisms including HSTS are presented. By measuring and exploring the adoption, works show that HSTS policy is still messed in configuration [42, 46, 54, 60] and can be bypassed [59]. Some variants of stripping attacks appear, when HSTS is partially deployed [22, 42, 61], or in the help of malicious DNS servers [13, 33]. However, these stripping attacks are noticeable to users now, because of browser security indicators, which show connection security and the authenticity of the remote web server.

For further attacks without being noticed, [31] and [14] uncover a new possibility of stripping attacks under the shared security environment, which can confuse the origins. As a common phenomenon, the multi-domain certificates are widely used in the wild for the convenience of certificate management. Thus, we begin to make an empirical study on the HTTPS hijacking or stripping attacks based on shared certificates (SCC attacks) to explore the threat of such attacks in the real-world. Compared with previous stripping attacks, SCC attacks can apply to the established secure connection. Moreover, attackers can bypass well-deployed policies by a flawed third-party server under the protection of a legitimate TLS certificate.

HTTP Misconfigurations and Inconsistencies. HTTP headers' inconsistencies are proved disastrous in an adversarial context [44]. However, due to the misconfigurations by developers, implementations vary among different entities. The inconsistencies exist between two web servers, between middleboxes (e.g., proxies, firewalls, CDN nodes) and back-end servers, or even between different user agents [52]. In fact, ambiguities among entities can appear in any security policies enforced by HTTP headers, including HSTS and CORS [25, 42]. Mendoza *et al.* explored the impact of subtle inconsistencies of HTTP security headers for websites on different platforms (e.g., desktop, mobile), which expose users to attack [52].

Notably, Chen *et al.* demonstrated the implications of inconsistent implementations while processing `Host`, a critical HTTP header indicating the origin while enforcing security policies [26]. The defects of interpreting `Host` may break the isolation of different origins, especially in HTTPS environments [31]. Under the premise of origin confusion, we find the well-known inconsistencies of security policies may influence the secure context and will lead to more attack scenarios with the help of shared certificates.

8 CONCLUSION

Though the specially designed policies can mitigate well-known attacks like SSL stripping, it is still vulnerable in the origin confusion scenarios with shared TLS certificates. We systematically evaluate the implications of SCC attacks. We show the adversaries can hijack the secure traffic between clients and well-configured servers, by using the misconfigured response headers from other servers. The attacks apply to established secure connections, which are unnoticeable for users and applications because the accessed website's certificate is valid for the flawed server. We find several attack scenarios like payment hijacking, download hijacking from major sites, like Microsoft, Alibaba, and JD.com. Meanwhile, we propose a systematical methodology to discover the exploitable web servers in the wild. Through a measurement study, we find 25.2% subdomains of Alexa Top 500 websites are affected by these issues. From the implementations, we find diversity among the websites on dealing with security policies, especially HTTP headers. Besides, the widely shared TLS certificates should take primary responsibility. As such, we expect the community and the developers should raise more attention to the security status of certificate management and policy implementation.

ACKNOWLEDGMENTS

We sincerely thank all anonymous reviewers for their valuable reviews and comments to improve the paper. We also thank Junlin Wei, Minglei Guo and Xiarun Chen for their help on the paper.

This work is supported in part by the National Natural Science Foundation of China (Grant No. U1836213 and U1636204), the BNRist Network and Software Security Research Program (Grant No. BNR2019TD01004), Beijing Nova Program of Science and Technology (Grant No. Z191100001119131), and the Joint Funds of the National Natural Science Foundation of China (Grant No.U1836113).

REFERENCES

[1] [n.d.]. CVE-2011-3389: BEAST Attack. https://nvd.nist.gov/vuln/detail/CVE-2011-3389.

[2] [n.d.]. Danami: Port/IP Redirection. https://docs.danami.com/juggernaut/user-guide/port-ip-redirection. Accessed: Nov 3, 2019.

[3] [n.d.]. Google Chrome engineers want to block some HTTP file downloads. https://www.zdnet.com/article/google-chrome-engineers-want-to-block-some-http-file-downloads/. April 10, 2019.

[4] [n.d.]. Google: HSTS Preload List. https://opensource.google.com/projects/hstspreload.

[5] [n.d.]. Google Transparency Report: HTTPS encryption on the web. https://transparencyreport.google.com/https/overview.

[6] [n.d.]. HTTPS usage statistics on top 1M websites. https://statoperator.com/research/https-usage-statistics-on-top-websites/. Accessed: Dec 14, 2019.

[7] [n.d.]. Man-in-the-Middle TLS Protocol Downgrade Attack. https://www.praetorian.com/blog/man-in-the-middle-tls-ssl-protocol-downgrade-attack. Accessed: Auguset 23, 2019.

[8] [n.d.]. MDN Web Docs: Mixed content. https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content.

[9] [n.d.]. MDN Web Docs: Redirections in HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections.

[10] [n.d.]. No More Mixed Messages About HTTPS. https://blog.chromium.org/2019/10/no-more-mixed-messages-about-https.html. October 3, 2019.

[11] [n.d.]. OWASP Secure Headers Project. https://www.owasp.org/index.php/OWASP_Secure_Headers_Project. Accessed: Dec 2, 2019.

[12] [n.d.]. Preventing Mixed Content. https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content.

[13] [n.d.]. SSLStrip2. https://github.com/LeonardoNve/sslstrip2.

[14] [n.d.]. TLS Redirection (and Virtual Host Confusion). https://github.com/GrrrDog/TLS-Redirection#intro.

[15] [n.d.]. Trying to take the dum-dum out of Security. https://web.archive.org/web/20150921195009/http://sign0f4.blogspot.com/2014/10/mitmf-v07-released-sslstrip-integration.html.

[16] [n.d.]. Usage statistics of HTTP Strict Transport Security for websites. https://w3techs.com/technologies/details/ce-hsts/all/all. Accessed: Dec 13, 2019.

[17] 2016. Mixed Content (W3C Candidate Recommendation). https://www.w3.org/TR/2016/CR-mixed-content-20160802/.

[18] 2019. Mixed Content (W3C Editor's draft). https://w3c.github.io/webappsec-mixed-content/.

[19] Nadhem J Al Fardan and Kenneth G Paterson. 2013. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 526–540.

[20] Eman Salem Alashwali and Kasper Rasmussen. 2018. What's in a downgrade? A taxonomy of downgrade attacks in the TLS protocol and application protocols using TLS. In *International Conference on Security and Privacy in Communication Systems*. Springer, 468–487.

[21] Stefano Calzavara, Alvise Rabitti, Alessio Ragazzo, and Michele Bugliesi. 2019. Testing for integrity flaws in web sessions. In *European Symposium on Research in Computer Security*. Springer, 606–624.

[22] Stefano Calzavara, Alvise Rabitti, Alessio Ragazzo, and Michele Bugliesi. 2019. Testing for Integrity Flaws in Web Sessions.

[23] Fuqing Chen, Haixin Duan, Xiaofeng Zheng, Jian Jiang, and Jianjun Chen. 2018. Path Leaks of HTTPS Side-Channel by Cookie Injection. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 189–203.

[24] Jianjun Chen. [n.d.]. *Host of Troubles Vulnerabilities*. https://hostoftroubles.com/.

[25] Jianjun Chen, Jian Jiang, Haixin Duan, Tao Wan, Shuo Chen, Vern Paxson, and Min Yang. 2018. We Still Don't Have Secure Cross-Domain Requests: an Empirical Study of {CORS}. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1079–1093.

[26] Jianjun Chen, Jian Jiang, Haixin Duan, Nicholas Weaver, Tao Wan, and Vern Paxson. 2016. Host of troubles: Multiple host ambiguities in http implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1516–1527.

[27] Shuo Chen, Ziqing Mao, Yi-Min Wang, and Ming Zhang. 2009. Pretty-bad-proxy: An overlooked adversary in browsers' https deployments. In *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 347–359.

[28] Taejoong Chung, David Choffnes, and Alan Mislove. 2016. Tunneling for transparency: A large-scale analysis of end-to-end violations in the internet. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 199–213.

[29] David Cooper, Stefan Santesson, S Farrell, Sharon Boeyen, Rusell Housley, and W Polk. 2008. RFC 5280: Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile. *IETF, May* (2008).

[30] X de Carné de Carnavalet and Mohammad Mannan. 2016. Killed by proxy: Analyzing client-end TLS interception software. In *Network and Distributed System Security Symposium*.

[31] Antoine Delignat-Lavaud and Karthikeyan Bhargavan. 2015. Network-based origin confusion attacks against HTTPS virtual hosting. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 227–237.

[32] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception.. In *NDSS*.

[33] Leonardo Nve Egea. [n.d.]. dns2proxy. https://github.com/LeonardoNve/dns2proxy.

[34] Leonardo Nve Egea. 2015. sslstrip+. https://github.com/LeonardoNve/sslstrip2.

[35] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS Adoption on the Web. In *26th USENIX Security Symposium ({USENIX} Security 17)*. 1323–1338.

[36] Steve Gibson. Dec 11, 2005. ARP Cache Poisoning: How one bad machine on your Ethernet Local Area Network (LAN) can ruin your whole day. https://www.grc.com/nat/arp.htm.

[37] Jeff Hodges, Collin Jackson, and Adam Barth. 2012. RFC 6797: Http strict transport security (hsts). *URL: http://tools.ietf. org/html/draft-ietf-websec-strict-transport-sec-04* (2012).

[38] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. 2011. The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 427–444.

[39] Ralph Holz, Yaron Sheffer, and Peter Saint-Andre. 2015. RFC 7457: Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). (2015).

[40] P Johnston and R Moore. [n.d.]. Multiple browser cookie injection vulnerabilities (2004).

[41] David Keeler. 2012. Preloading HSTS. *Mozilla Security Blog* (2012).

[42] Michael Kranch and Joseph Bonneau. 2015. Upgrading HTTPS in Mid-Air: An Empirical Study of Strict Transport Security and Key Pinning. NDSS.

[43] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. 2019. All things considered: an analysis of IoT devices on home networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1169–1185.

[44] Arturs Lavrenovs and F Jesús Rubio Melón. 2018. Http security headers analysis of top one million websites. In *2018 10th International Conference on Cyber Conflict (CyCon)*. IEEE, 345–370.

[45] Olivier Levillain. 2016. *A study of the TLS ecosystem*. Ph.D. Dissertation. Institut National des Télécommunications.

[46] Xurong Li, Chunming Wu, Shouling Ji, Qinchen Gu, and Raheem Beyah. 2017. HSTS Measurement and an Enhanced Stripping Attack Against HTTPS. In *International Conference on Security and Privacy in Communication Systems*. Springer, 489–509.

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

[48] Meng Luo, Pierre Laperdrix, Nima Honarmand, and Nick Nikiforakis. 2019. Time Does Not Heal All Wounds: A Longitudinal Analysis of Security-Mechanism Support in Mobile Browsers.. In *NDSS*.

[49] Moxie Marlinspike. 2002. SSLSniff Attack. https://moxie.org/software/sslsniff/. Accessed: Nov 4, 2019.

[50] Moxie Marlinspike. 2009. More tricks for defeating SSL in practice. *Black Hat USA* (2009).

[51] Moxie Marlinspike. 2009. SSLStrip Attack. https://moxie.org/software/sslstrip/. Accessed: Nov 4, 2019.

[52] Abner Mendoza, Phakpoom Chinprutthiwong, and Guofei Gu. 2018. Uncovering http header inconsistencies and the impact on desktop/mobile websites. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 247–256.

[53] Marlingspike Moixe. 2009. New tricks for defeating ssl in practice. In *BlackHat Conference, USA*.

[54] Ivan Petrov, Denis Peskov, Gregory Coard, Taejoong Chung, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. [n.d.]. Measuring the Rapid Growth of HSTS and HPKP Deployments. ([n. d.]).

[55] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS usage in Android apps. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 350–362.

[56] Eric Rescorla. 2000. RFC 2818: Http over tls. (2000).

[57] Ivan Ristic. 2013. *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck.

[58] Peter Saint-Andre and Jeff Hodges. 2011. RFC 6125: Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X. 509 (PKIX) Certificates in the Context of Transport Layer Security (TLS). *Internet Engineering Task Force (IETF), RFC* (2011).

[59] Jose Selvi. [n.d.]. Bypassing HTTP strict transport security. ([n. d.]).

[60] Suphannee Sivakorn, Angelos D Keromytis, and Jason Polakis. 2016. That's the Way the Cookie Crumbles: Evaluating HTTPS Enforcing Mechanisms. In

Table 4: The Response Headers of the Same HTTPS Requests Sent from Different Regions (for Case 4)

Request URL	https://www.weibo.com/	
Dst IP	180.149.134.141	
Src IPs	**Location**	**Response Headers**
111.25.158.225	Jilin, China	HTTP/1.1 200 OK Server: WeiBo/LB Transfer-Encoding: chunked pramga: no-cache Content-Encoding: gzip LB_HEADER: venus244 ...
45.32.47.58	Tokyo, Japan	HTTP/1.1 302 Moved Temporarily server: nginx status: 302 pramga: no-cache location: http://weibo.com/jp x-via-cdn: f=Akamai, s=23.207.172.156, c=45.32.47.58 ...
144.202.112.190	Los Angeles, US	HTTP/1.1 302 Moved Temporarily pramga: no-cache server: nginx status: 302 location: http://weibo.com/us x-via-cdn: f=Akamai, s=23.208.64.165, c=144.202.112.190 ...

Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society. ACM, 71–81.

[61] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. 2016. The cracked cookie jar: HTTP cookie hijacking and the exposure of private information. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 724–742.

[62] Christopher Soghoian and Sid Stamm. 2011. Certified lies: Detecting and defeating government interception attacks against SSL (short paper). In *International Conference on Financial Cryptography and Data Security*. Springer, 250–259.

[63] speedguide. [n.d.]. Port 8443 Details. howpub=https://www.speedguide.net/port.php?port=8443.

[64] Joe Stewart. 2003. DNS cache poisoning–the next generation.

[65] Paul Vixie. 1995. DNS and BIND Security Issues.. In *Usenix Security Symposium*.

[66] Xiaofeng Zheng, Jian Jiang, Jinjin Liang, Haixin Duan, Shuo Chen, Tao Wan, and Nicholas Weaver. 2015. Cookies lack integrity: Real-world implications. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 707–721.

APPENDIX

A THE RESPONSE HEADERS OF THE SWITCHING CLIENT IP CASE

In Section 5.3.2, we show a Case 4 of downgrading HTTPS traffic by switching the client IP address. Here, in Table 4, we present the response headers we receive at the locations, including Jilin (China), Tokyo (Japan), and Los Angeles (US). From the `Location` headers, we see that WeiBo serves for users in Japan via the path of `/jp`, and users in the US via `/us`.