HSTS Measurement and an Enhanced Stripping Attack Against HTTPS

Xurong Li[1(✉)], Chunming Wu[1], Shouling Ji[1,2], Qinchen Gu[3], and Raheem Beyah[3]

[1] Zhejiang University, Hangzhou, China
{lixurong,wuchunming,sji}@zju.edu.cn
[2] Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Hangzhou, China
[3] Georgia Institute of Technology, Atlanta, USA
qgu7@gatech.edu, raheem.beyah@ece.gatech.edu

Abstract. HTTPS has played a significant role in the Internet world. HSTS is deployed to ensure the proper running of HTTPS. To get a good understanding of the deployment of HSTS, we conducted an in-depth measurement of the deployment of HSTS among Alexa top 1 million sites, and investigated bookmarks and navigation panels in different browsers. We found five types of threats, including transmission errors, redirection errors, field setting errors, the auto completion mechanism in bookmarks and the embedded addresses in navigation panels. To demonstrate defects we found, we designed an enhanced HTTPS stripping attack, which was upgraded from the original *sslstrip* attack. Finally, we gave three effective suggestions to eliminate these defects. This paper exposed various risks of HTTPS and HSTS, making it possible to deploy HTTPS and HSTS in a more secure way.

Keywords: HSTS · HTTPS · Stripping attack · Security

1 Introduction

Users value security and privacy more than ever. HTTPS [1], which consists of HTTP [2] and SSL/TLS [3,4] protocols, is created to provide confidentiality and integrity of web browsing. Recently, many companies have taken measures to prompt the deployment of HTTPS. Since 2014, Google has improved rankings of the websites which deploy HTTPS [5]. Furthermore, in Chrome, the websites which do not deploy HTTPS can not even make use of geographic location and the application cache. Eventually they will result in an unsafe symbol in the address bar of the Chrome browser. In the past, obtaining and maintaining of the digital certificates would cost a lot. Therefore small companies or big companies with many domain names might not deploy HTTPS for the cause of expense. Fortunately, Let's Encrypt [6], which is a non-profit organization, provides Domain Validation (DV) certificates for free through a fully automated

X. Lin et al. (Eds.): SecureComm 2017, LNICST 238, pp. 489–509, 2018.
https://doi.org/10.1007/978-3-319-78813-5_25

process. Apart from Let's Encrypt, several content distribution networks and cloud service providers, including CloudFlare and Amazon, provide free TLS certificates to their customers.

However, there are still many HTTP connections that exist in the Internet. To handle the mix of HTTP and HTTPS connections seamlessly is difficult for browsers due to the stripping attack. HTTPS stripping attacks have raised widespread concerns since Marlinspike put forward *sslstrip* at the black-hat conference in 2009 [7]. Attackers can intercept the communication between the target website and the client, and change all *https* into *http* in the response packets from the website. Even though this attack violates the rule which states TLS/SSL should ensure end to end security, neither the client nor the server can be aware of the attack for the reason that the packets sent from servers are still encrypted.

To defend against the stripping attack, HTTP Strict Transport Security (HSTS) [8] protocol was presented in 2012. It defines a mechanism enabling websites to declare themselves accessible only via secure connections. In consideration of the complexity of protocol and the diversity of communication platforms, we are concerned about whether the HSTS policy has been understood well. In our work, we conducted a comprehensive measurement about the deployment situation of HSTS on both PC and mobile websites. Subsequently, we investigated the bookmarks and navigation panels in browsers. We found five kinds of risks in the deployment on different platforms, which can be ignored easily by users or developers. These risks are categorized in Table 1. According to the risks we found, there is still a great probability of launching a stripping attack. But after our tests, the old *sslstrip* tool failed to attack the current websites. In order to understand the dangers of these risks well, we enhanced the original stripping attack and implemented a new HTTPS stripping attack through adding an script. Finally, we launched the attack in a simulative environment to test various famous sites, including *mail.qq.com*, *www.amazon.com*, *www.baidu.com*, *taobao.com*. The results of stripping attack were all successful based on the defects we found. The major contributions of this paper are as follows:

- We conduct an in-depth measurement of HSTS deployment on both PC websites and mobile websites, and the results show that many problems exist in the deployment, including incorrect setting methods and field setting errors. Particularly, redirection problems in mobile websites pose a risk to HSTS.
- We perform an investigation about bookmarks and navigation panels in different browsers. Through careful observation, we find that defects of the auto completion mechanism in bookmarks and the embedded addresses in navigation panels may lead to a stripping attack.
- We analyze the old *sslstrip* tool, and find it is not suitable for complicated webpages. Besides, we implement an enhanced HTTPS stripping attack.

Based on the defects in browsers and deployment of HSTS, we launch this attack in several simulative scenarios successfully[1].
- We give three kinds of useful suggestions to handle these security threats above.

Table 1. Five kinds of risks found in the measurement

Incorrect setting method	HSTS is set via HTTP
Field setting errors	Many field settings in HSTS headers do not obey the standard
Redirection problems	HSTS is not deployed correctly during redirections
Bookmark in browsers	The auto completion mechanism in bookmarks only provides HTTP
Navigation panels in browsers	The embedded addresses in navigation panels take HTTPS as HTTP

The rest of this paper is organized as follows. Section 2 provides background information about HTTPS and HSTS. Section 3 details the data collection, and introduces the data source. Section 4 gives an in-depth analysis of deployment of HSTS on both PC websites and mobile websites. Section 5 implements an enhanced HTTPS stripping attack, and demonstrates the attack. Section 6 discusses possible mitigations. Section 7 surveys related work. And finally, Sect. 8 concludes our work.

2 Overview of Web Security

HTTPS [1] was created in 2000. It describes how to use TLS to secure HTTP connections over the Internet. In this section, we will give a short introduction to HTTPS and HSTS, and talk about HSTS security and stripping attack.

2.1 HTTPS and Stripping Attack

A few years ago, HTTPS was deployed only in financial or e-commerce payment pages or login pages. However, the situation has changed over time. More and more sites began to deploy HTTPS. One of the reasons is that many studies show that the site owners should provide HTTPS service on all site pages, including whole resource files and thus encryption of part of the sites is proven unsafe [9,10]. Another reason is the emergence of free certificates and TLS accelerator. The cost to maintain HTTPS service was very expensive, which contained the

[1] We conducted the experiment in local computers and network, which formed an emulated environment.

cost of applying certificates, the cost of updating certificates, and the performance overhead caused by extra encryption or decryption. Fortunately, these problems have been solved in recent years. Many organizations began to provide free TLS/SSL certificates and websites greatly benefited from HTTPS.

Nonetheless, HTTPS stripping attack poses a risk to HTTPS. When users type a domain name without protocol type (HTTP or HTTPS), the default request type is HTTP rather than HTTPS. Usually, if the server provides HTTPS service, the server will give a 302 redirection after receiving an HTTP request. However, the attacker can intercept the traffic through ARP spoofing and replace all *https* with *http* in the response packet. Thus the browser will still request an HTTP website regardless of the 302 redirection. Again, the attacker can replace all *http* with *https* in the request packet. The attack is shown in Fig. 1. The communication between the attacker and the server is encrypted, but the communication between the attacker and the browser is in plaintext. This attack is called HTTPS stripping attack, which can not be detected by browsers or servers as it follows the HTTP communication protocol.

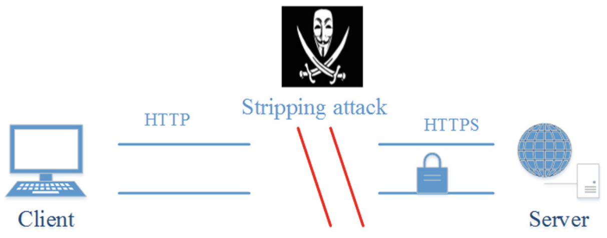

Fig. 1. Stripping attack: the attacker can intercept the traffic, establish an encrypted connection with the server, and communicate with the client via HTTP.

2.2 HSTS Protocol

To avoid the HTTPS stripping attack, HTTP Strict Transport Security (HSTS) policy was created in 2012 [8]. The policy is declared by websites via the Strict-Transport-Security HTTP response header field or by other means, such as user agent configuration. If the server wants to provide HTTPS service all the time, it will send an HSTS header to the browser. According to the information in headers, the browser will remember the domains which want to force to be visited by HTTPS. And when users send an HTTP request next time, the browser automatically converts HTTP to HTTPS in the background. The HSTS policy defines the standard of HSTS headers, and the headers mainly consist of three fields. The first is the *max-age* field, which means the expiration time and it is mandatory. The second is the optional *includeSubdomains* field, which indicates whether the HSTS policy applies to the subdomains of the domain. The last one is the *preload* field and it is also optional. This field indicates whether the domain has been permanently added into the preload list, which is maintained by browser providers. What is essential is that these headers can only be sent by HTTPS requests, hence the attacker can not arbitrarily tamper with the HSTS policy to disable it.

2.3 HSTS Security Consideration

Although HSTS policy can defend against HTTPS stripping attacks to a certain extend, many new security issues still exist. The most common one is the incorrect configuration as many developers do not have a good understanding of the HSTS policy. For instance, if the *max-age* value is set too big or too small, HSTS policy will be reused or invalid. If the *max-age* value is too big, the policy will still work all the time even though the server does not want to provide HTTPS service anymore, which may cause websites unable to be visited. If the *max-age* value is too small, HSTS policy will be invalid in a very short period of time, which can be used to launch MitM attack by attackers. Besides, misuse of *includeSubdomain* and *preload* field will be vulnerable against DoS attacks. If the servers are unaware of being added to preload list and do not provide HTTPS service, the sites will fail to be accessed. In addition, whether the subdomains have properly deployed HSTS, whether each step in the redirection is deployed correctly and whether the web application contains any insecure references to the web application server are all problems concerned. Based on these considerations, we decide to conduct an in-depth measurement about HSTS deployment.

3 Data Collection

We used Python as the programming language in the whole experiment and we rewrote the *urllib2* library so that it could meet our requirements. Not only did we use *urllib2* to send HTTP or HTTPS requests, but also did we record each HSTS information in the event of redirection. First, we surveyed *www* subdomains of top 1 million sites [11] with PC user-agent[2]. The reason we chose this user-agent was that Chrome was the main advocate of HSTS. We sent both HTTP and HTTPS requests for the same domain name and recorded the response packets in each redirection. Then we sent the same requests to 1 million sites, except that mobile user-agent[3] was used instead. We repeated this process for three times to reduce the influence caused by network performance. In total, we sent out 1 million HTTP requests and 1 million HTTPS requests (like *http://www.example*, *https://www.example*[4]). Finally, we successfully visited 937,430 sites with HTTP requests and 631,833 sites with HTTPS requests, respectively, using PC user-agent. For mobile-agent, the number of successfully accessed sites is 936,268 and 635,041, respectively. Because of several sites which

[2] Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/537.36.

[3] Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 Mobile Safari/537.36.

[4] Here *example* refers to a domain name.

failed to respond, the number of responses is less than that of requests. As for HTTPS requests, many servers neither provided HTTPS service, nor provided a 302 redirection after receiving an HTTPS request. Based on these facts, many websites could not be visited when we sent an HTTPS request directly. All experiments were conducted in February 2017 and websites may adopt different policies over time.

4 Current Deployment Measurement

There are two ways to deploy HSTS policy. The first one is preload list, which is inserted into the browsers, and the other is dynamic HSTS, which is deployed by HTTP header. Kranch and Bonneau [12] has studied the preload list carefully. Therefore in this paper, we point out several problems about preload list which was not found in their work. What should be emphasized is that our work is completely different from Kranch's. We explain the reason why the maintenance of preload list is risky. Besides, we first conduct the measurement on mobile sites and analyze the redirections problems in detail. Moreover, we list the specific field setting errors and study the deployment in various browsers.

4.1 Preloaded HSTS

If domain has been added into the preload list, the browser will automatically convert HTTP requests for the domain to HTTPS requests in the background. We discovered a few new problems in preload list. The first one was sites added into the preload list do not send HSTS header. Sites with preload list need to set HSTS header as well since not all browsers support preload list. Users would be hijacked easily when they visit these sites on browsers which do not support preload list.

The maintenance of preload list is a hidden risk as well. Google has provided a website [13], which is used to submit domains for inclusion in Chrome's HTTP Strict Transport Security (HSTS) preload list. However, the requirements of submission are very strict and the sites must satisfy all requirements. If a site has been added into preload list before, but later it does not satisfy all requirements anymore, Chrome will delete it from preload list without notification [14]. As a website owner, one will not be visiting the HSTS preload page every week so the site may just be removed from the preload list without warning and the owner may not even notice it until many months later. Moreover, requirements for preload list are always changing. The website's owner has to pay attention to the state of preload list all the time.

Last but not least, sites in preload list have many setting errors and incorrect deployments [14]. For example, many redirections occur when we visit the HTTPS sites, but HSTS is not properly deployed in each event of the redirections. Not all subdomains support HTTPS and many HSTS headers do not contain *preload* or *includeSubdomain* field.

4.2 Alexa Top Million Websites with Dynamic HSTS

We wanted to know if the redirection was different across different platforms, so we conducted the test on both PC and mobile clients. In the rest of the paper, we use PC sites to mean the visit to the websites with PC user-agent, and use mobile sites to mean the visit to the websites with mobile user-agent.

Table 2. Successful responses of 1 million requests

Request type	HTTPS responses	HTTP responses	Total
PC-HTTP	170,883	766,547	937,430
PC-HTTPS	529,555	102,278	631,833
Mobile-HTTP	171,171	765,097	936,268
Mobile-HTTPS	525,724	109,317	635,041

The Overall Data Distribution. The results of all responses summarized in Table 2. As mentioned in Table 2, we got 937,430 PC responses and 936,268 mobile responses through the HTTP requests, and got 631833 PC responses and 635,041 mobile responses through the HTTPS requests. According to the results, we can know that many sites support both HTTP and HTTPS. Most PC sites (81.8%) and mobile sites (81.7%) still supported HTTP. We think the main reason is that many users are still using outdated browsers or systems, which do not support HTTPS well. Hence, website owners would like to remain compatible with these users' web clients and they responded to both HTTP requests and HTTPS requests. As for HTTPS requests, while more than half of the sites supported direct HTTPS access, a number of sites which did not support HTTPS failed to redirect HTTPS requests to HTTP sites. These websites may be vulnerable to DoS attack if the browsers keep sending HTTPS requests. Consequently, we analyzed the deployment of dynamic HSTS based on the results of responses. We counted the HSTS settings according to the HTTP 200 OK headers in Table 3.

Table 3. HSTS header setting

Request type	Sites with HSTS header
PC-HTTP	36,788 (3.9%)
PC-HTTPS	43,301 (6.9%)
Mobile-HTTP	36,643 (3.9%)
Mobile-HTTPS	43,353 (6.8%)

Two years ago, however, Kranch [12] found just 12,593 sites which attempted to send an HSTS header. This may imply an increasing number of sites realized the significance of HSTS and decided to deploy it. We clearly see the results are different for HTTP and HTTPS requests. Further analysis shows that 8296 PC sites deployed HSTS for HTTPS requests but not for HTTP requests. Particularly, when we visited the 8296 PC sites with HTTP requests, we got 8213 HTTP webpages and 83 HTTPS webpages. This is an interesting phenomenon, since the 8213 PC sites may support both HTTP and HTTPS to stay compatible with more users, but the 83 PC sites redirected HTTP requests to HTTPS requests without HSTS header, which may be a threat. Furthermore, we analyzed the distribution of HSTS deployments with Alexa ranking in Table 4.

Table 4. Alexa ranking and sites with HSTS

Request type	Top10	Top100	Top1W	Top10W	Top100W
PC-HTTP	7	30	814	5,607	36,778
PC-HTTPS	8	33	866	6,266	43,301
Mobile-HTTP	8	31	812	5,589	36,643
Mobile-HTTPS	8	31	855	6,242	43,353

From Table 4, we can learn the top websites attached great importance to the deployment of HSTS. More specifically, among the top 10 websites, *www.qq.com*, which is a news site, only supports HTTP requests, and *www.google.co.in* supports HTTPS but does not deploy HSTS. To our surprise, we found that *www.baidu.com* (PC site) took different strategies according to different IPs. When we sent HTTP request to *www.baidu.com* from the US, we received the response without HTTPS deployment, while the response came with HSTS deployment in China.

Incorrect Setting Method. RFC6797 [8] defines that HSTS can not be set via HTTP, thus we counted the invalid settings in Table 5.

Table 5. Invalid HSTS setting via HTTP

Request type	PC-HTTP	PC-HTTPS	Mobile-HTTP	Mobile-HTTPS
Sites of invalid HSTS setting	4,299	533	4,211	566

In order to understand the situation of invalid settings better, we have checked the details of settings. 525 PC sites sent HSTS header for both HTTP requests and HTTPS requests via HTTP, which means they only provided HTTP service but deployed HSTS. Particularly, 8 PC sites sent HTTP pages without HSTS for HTTP requests, and redirected HTTPS requests to HTTP requests of another domain, and that domain would send an HTTP page with HSTS header. For example, for *www.andreicismaru.ro*, we would get normal HTTP page without HSTS header for HTTP request, but also HTTP page with HSTS header from *http://cetin.ro/* after we sent HTTPS request to *www.andreicismaru.ro*. However, if they only provided HTTP service, HSTS policy would be invalid. These sites' owners may have a misconception of the HSTS policy. Namely, HSTS policy does not provide confidentiality of traffic, it just ensures the correct implementation of the HTTPS.

Errors of HSTS Field Settings. In these detected HSTS headers, we found various errors that were contrary to the standard protocol. The protocol points out that *max-age* is a required field but the results showed that both mobile sites and PC sites have several *max-age* setting errors. We took PC websites as an example to avoid duplication.

First, we found errors in headers which were not properly including *max-age=*[5]. For instance, *www.lovdata.no* set the field to *maxage=31,536,000*, which missed the symbol -. *www.mijn-econnect.nl*, *www.xn—-7sbnackuskv0m.xn–p1ai* and *www.bottomline.com* all missed the symbol =.

And *www.chrcitadelle.be* set the field to a single number, like *86,400*.

Then, we checked the headers with *max-age=*. Unfortunately, many formal errors exist and we show them in Table 6. Following this, we found a lot of *max-age* values were not reasonable. If the value is extremely small, HSTS policy will soon expire. To our surprise, 1,484 sites deploy HSTS with *max-age=0*, which means the HSTS policy is invalid. But the big value is not reasonable as well, owing to the fact that sites need to update HSTS policy timely if HTTPS service changes. However, *www.cloudup.com* set the filed to*max-age=100,000,000,000*, and *www.aptopnews.com* set the field to *max-age=9,223,372,036,854,775,807*. Both of them are more than 1,000 years.

Redirection Problems. Usually, lots of redirections exist during our visit to many sites. If HSTS deployment in redirections is incorrect, the final HSTS is equally invalid. We counted the redirection times of the sites where HSTS was deployed in Table 7.

Here we counted every client request until the final visit succeeded. It can be seen from Table 7, most sites who deployed HSTS did not have redirections for HTTPS requests. However, one or more redirections occurred when handling HTTP requests. If the redirections between HTTP and HTTPS did not deal with HSTS deployments well, the attack will be equally easy despite the existence of HSTS. We found that 929 PC sites did not fully deploy HSTS during redirection from HTTP to HTTPS, and the number for mobile sites is 1106. In order to understand the distinction better, we analyzed the detail of redirections. A fact we can not overlook is that many sites provided different domains for mobile requests and PC requests, like *https://m.example* and

[5] *max-age=* is the standard format which is defined by RFC document.

Table 6. Examples of *max-age=* errors

max-age=expireTime
%E2%80%9Cmax-age=31536000%E2%80%B3
xa8xb9max-age=31536000xa8xb9xb3xacN
max-age=31536000%E2%80%9D
max-age="157680000"
max-age="10368000"
max-age=<31536000>
max-age=31536eee
max-age=31556926?
xa8xb9max-age=31536000?
x81gmax-age=31536000x81
max-age=
max-age=0.000001

Table 7. Redirection times of sites with HSTS header

Request type	Redirection times			
	0	1	2	≥3
PC-HTTP	7.9%	60.7%	27.0%	4.4%
PC-HTTPS	57.8%	34.6%	6.0 %	1.6%
Mobile-HTTP	7.5%	58.3%	28.7%	5.5%
Mobile-HTTPS	55.6%	35.6%	6.8%	2%

https://www.example. However, many mobile servers just deployed HSTS in the response of *https://m.example* but not *https://www.example.* In the end, if we still send HTTP requests of *www.example*, HSTS policy would not work. We show this process in Fig. 2. Besides, the same situation occurred when the *www* subdomain did not exist.

Specifically, 12 mobile sites did not deploy HSTS in the first response to *https://m.example.* Moreover, 110 sites first gave a response of *http://m.example*, and then redirected to *https://m.example* with HSTS headers. But it was not enough, for the attacker can hijack the request of HTTP. It needs to be emphasized that sites should deploy HSTS in the first response after requesting domain A, if they want to provide a redirection from domain A to domain B.

4.3 Two Ubiquitous Overlooks

Bookmarks in Browsers. Although many sites have already known the significance of HSTS policy, there are still serious problems as described above. In this section, we investigate two kinds of phenomenas that were easily overlooked. Bookmarks in browsers are often used to record a website that users would like to visit later. Sometimes users add the current page being visited to bookmarks, so the scheme attributes will be preserved. However, if users manually type in

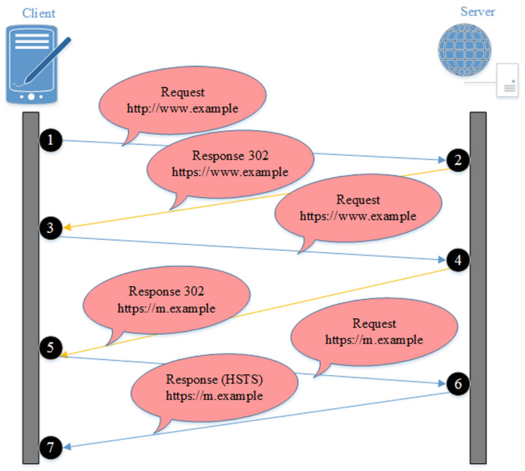
Client
Server
Request http://www.example
Response 302 https://www.example
Request https://www.example
Response 302 https://m.example
Request https://m.example
Response (HSTS) https://m.example
1
2
3
4
5
6
7

Fig. 2. Mobile sites redirection problem: server gives two redirections without HSTS when handling HTTP request from mobile browser in step ❶∼❺, and only deploys HSTS in the response of mobile domain in step ❻∼❼.

the URL, they may forget to enter the scheme part. The browsers will add the URL with the HTTP prefix automatically, which means that the browser will send an HTTP request first when the users click the bookmark. In addition, we have learned that mobile devices' bookmarks keep in sync with Safari browser for iOS users. If there are too many redirections before the final visit to the HTTPS site, there will be a threat. We have checked different browsers (Chrome, Firefox, Safari) and found the same threat. Bookmarks did not have a mechanism to check these URLs. Google showed the popular desktop browsers [15] and we checked the bookmarks and navigation panels of these browsers. Opera, Chrome, Edge, and Firefox all support adding URL manually and add the HTTP prefix by default. However, users cannot insert URLs into IE11 and Safari manually. The users can only add the sites which they are visiting in these two browsers.

Navigation Panels in Browsers. Almost all of the browsers' home pages include navigation panels, and websites offer navigation panels services as well. Unfortunately, after our in-depth investigation, we have found that there is an error in the built-in URL of the navigation site. Many sites which only support HTTPS are inserted with HTTP in navigation pages. Users tend to trust the address of navigation panels and click it instead of typing address in address bar. Therefore, browsers will send HTTP requests for these sites even they support HTTPS and it may contain a threat according to the risks mentioned above. In the next section we will introduce an enhanced HTTPS attack based on the risks in these findings.

5 An Enhanced HTTPS Stripping Attack

HSTS allows a web site to opt in to be HTTPS only. For a site with HSTS, a browser will only send HTTPS requests, eliminating the window of insecurity. Apart from this, HSTS maintains a preload list, which is hard coded into browsers and is supported by Chrome and Firefox. However, very few websites have joined the list, and many have chosen to implement dynamical HSTS. The emergence of HSTS can avoid stripping attacks to a great extent, but we have discovered the flaws in the HSTS deployment and browsers. Based on these defects, HTTPS stripping attack can still work. In this section, we will analyze the reason why original stripping attack tool *sslstrip*, which was developed by Marlinspike [7], does not work in new environment. Then, we implement an enhanced HTTPS stripping attack and verify it on famous sites. We only want to prove that HTTPS can be downgraded easily based on the defects.

5.1 Original Sslstrip and Inspiration

Plenty of attacking tools have integrated *sslstrip*, such as *bettercap* [16], *mitmf* [17]. However, through our tests of various websites, we have found that the success rate of *sslstrip* was very low. To learn the reason in detail, we studied the principles of *sslstrip*. After hijacking the traffic, *sslstrip* will search the *https* strings, and replace all *https* with *http* in traffic. Then we analyzed the source code of webpages and found the answer. Old webpages are usually constituted of static text, and the replacement of *https* is simple. However, the web pages have become more complicated over time, and new webpages contain a large number of dynamic elements. Besides, many take new methods to detect stripping attack, like the *location* in srcipts, but *sslstrip* does not have any solution to handle these scripts. Moreover, the time consumption of replacement in *sslstrip* is very large for the reason that *sslstrip* has to wait for all packets and search the target strings. If too much time has been spent on replacement, the connection will fail. After our tests, we found that the users can not visit the most webpages when *sslstrip* works, indicating that the original *sslstrip* is not suitable for the current web pages.

Researchers have pointed out that front-end hijacking is an effective method in the blog [18]. So in this section, we will take a front-end approach to perform stripping attacks according to the ideas mentioned in the blog. The main principle is derived from this blog, but we have improved the method. The differences between our work and the blog are three-folds: First, we handle the *location* field, which can detect the stripping attack. We modify the *location* field in the script and make it invalid. Second, we handle secure cookie. Secure cookie must be deleted from response headers so that we can get plenty of privacy information. Finally, we do a number of tests to verify the effectiveness of the attack. The tests are done on different browsers and famous websites. In our attack, XSS skill is used, but it does not mean that the attacker can inject any content all the time. If we do not downgrade the HTTPS scheme, the following traffic will be encrypted. Actually, we only want to show the possibility of HTTPS being downgraded based on the defects we find. Designing a new attack tool is not our goal.

5.2 Principles of Enhanced Stripping Attack

Precondition of Attack. What needs to be emphasized is that our attack will be invalid if the first request is an HTTPS request. Also, if the domain has been added to the HSTS preload list, our attack will not work. However, based on our previous sections, it is not difficult for us to get HTTP request first. We summarize the reasons below:

- Firstly, the preload list is so short that it can not include all websites and many browsers still did not support preload list.
- Secondly, many sites only support HTTPS service but not HSTS, so the results will be HTTP requests first if the user type the domain name in address bar without scheme.
- Thirdly, if the users have not visited the site, the first request is still the HTTP request due to the fact that HSTS policy has not worked yet.
- Fourthly, many redirection problems occur during HSTS deployment according to our study. Besides, a mix of HTTP and HTTPS connections exists in plenty of websites. Both of them responded to the HTTP request.
- Finally, the records in bookmarks and navigations panels remembered the HTTP request or old domain name.

Therefore, even the sites provided HTTPS service and deployed HSTS policy, it was not very difficult for the attacker to get HTTP request first. Then the attacker must act as a proxy in our attack. It was easy for the attacker to get traffic data by using ARP spoofing or DNS spoofing tools in local area network or WiFi and the attack is carried out in the process of counterfeiting proxy. What should be noted is that our attack is not a perfect attack tool for all webpages. Because our goal is to show the insecurity of incorrect deployment. Hence, our attack mainly focuses on the common web structures.

Detailed Implementation. The whole idea of attack is ingenious to injecting a JavaScript script at the beginning of the traffic. If users do not click the *https* links, the links will not be effective. Therefore, the key is to replace the link at the moment of the click. *DOM-3-Event* is an event capture mechanism, which can be used to capture the global click event. If the clicks fall on the *https* hyperlink, we intercept them and change *https* to *http* and the time cost is very small.

As for form submission, we can listen to the *submit* instead of *click*, and change the *href* to *action*. For frame pages, this is a problem. We only downgrade the main page into HTTP version, but the frame address is still the original, which will cause a cross domain problem for the different protocols. We use *Content Security Policy* to avoid the HTTPS framework page. In the response from our proxy, we added the following HTTP header.

CSP policy

```
Content-Security-Policy: default-src *
data: ‘unsafe-inline’ ‘unsafe-eval’;
frame-src http://*
```

‘*unsafe-inline*’ allows the page to load inline resource, ‘*unsafe-eval*’ allows the page to load dynamic JS code, and *frame-src* specifies the frame’s load policy. However, after our test, we found that many websites use script, namely, the *location* attribute of the browser [19], to detect whether the site provides HTTPS protocol. Here we take *mail.qq.com* for example. The *mail.qq.com location* program is showed below.

mail.qq.com location

```
<script>
(function()
{if(location.protocol=="http:"){
document.cookie = "edition=;expires=-1;
path=/;domain=.mail.qq.com";
location.href="https://mail.qq.com";
}
})();
</script>
```

If the protocol of the site has been changed to *http*, the cookie and scheme will be changed back by *location*. Drawing on the idea of the original *sslstrip* attack, we can replace the *http* with *https* in the script in the backend proxy. We only search the *http* in scripts, thus the cost of replacement can be ignored. We first search the *location* in script and then replace the *http* with *https*. Even the protocol scheme is *http*, the jump will not occur. So the whole idea is consisted of two parts. The first one is the XSS script of front end, which listens to events and replaces *https* with *http*. The other is the proxy of back end, which can replace the *http* in *location* field to prevent the script from jumping. To launch the stripping attack successfully, we have to solve two problems, which we summarize as follows:

(1) The problem is how to let the proxy know whether the request is sent by HTTPS or HTTP. The proxy actually is a man in the middle. If it modifies the HTTPS resource, it must restore the HTTPS request to the server, otherwise the attack will be detected by the server.
(2) Many websites redirect HTTP requests to HTTPS websites through 302 redirection and have deployed HSTS. We need to forward the redirection, inject the script and delete the HSTS header.

For the first problem, we use the mark method to distinguish whether the link is replaced. When we replace the HTTPS with HTTP, we can add a mark in the modified URL at the same time. In order to hide the mark, we can choose fraudulent marks, like *utf-8*, *?zh_cn*, *?ssl*. Therefore, when the proxy handles the

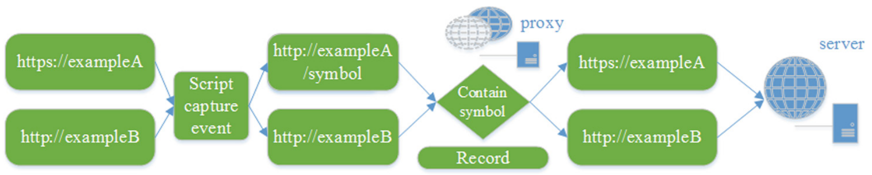

Fig. 3. Proxy mark process: first the XSS script will add a symbol into HTTPS request and HTTPS will be downgraded to HTTP. Then the proxy will change the HTTP request to HTTPS request according to the symbol and forward it to remote server. The HTTP request will be forwarded to the server directly.

URL, it will know how to take measures to forward the requests according to the mark. The proxy also need to record the *https* requests and symbols. The whole processes are described in Fig. 3. For the second problem, we must add a module in proxy to handle it. In that module, we intercept this redirection, obtain the content of redirection with HTTPS request, and finally reply the user with HTTP scheme. The response to the user will include the script and CSP policy we designed. Besides, if we find HSTS header or secure cookie field in responses from server, we delete them in proxy quickly. The redirection problem is handled in Fig. 4.

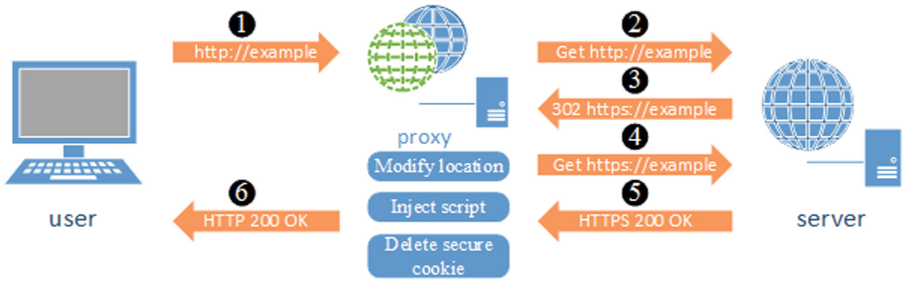

Fig. 4. Handling redirection: the proxy will forward the http request to remote server in step ❶∼❷, and then establish a secure connection with the server in step ❸∼❺. Next, the proxy will modify the location and inject the XSS script in the response packet. Finally, the proxy will return an HTTP page with script to the user in step ❻.

5.3 Experiment Results

In this section we demonstrate our attack against several popular websites to verify the defects we found. These websites are shown in Table 8.

Here *mail.qq.com* represents the *qq.com* due to the fact that *www.qq.com* provides HTTP service[6]. And *m.kaskus.co.id* represents mobile sites which did

[6] *mail.qq.com* is one of the most popular Chinese e-mail services and provides HTTPS services. Although *mail.qq.com* did not deploy HSTS, it used *location* to detect whether the current protocol is downgraded.

Table 8. Sites to test and relevant security measures

Alexa ranking	Domain	HTTPS	Dynamic HSTS	Preload list
4	www.baidu.com	Y	Y	N
6	www.amazon.com	Y	Y	Y
8	mail.qq.com	Y	N	N
12	www.taobao.com	Y	Y	N
335	m.kaskus.co.id	Y	Y	N

not deploy HSTS in the first response to *https://m.example*. We launched our attack on different scenarios and different browsers[7].

The results of attack are showed in Table 9. Bookmarks and navigation imply default HTTP requests in these mechanisms. Automatic domain means that users typed domain manually and the browser complemented a domain name automatically. These three tests were conducted on Firefox browser, which supports preload list. We show the examples of *www.taobao.com* and *www.amazon.com* in Figs. 5 and 6, which show the hijacked URLs in address bars.

Table 9. The results of new stripping attack

Domain	Scenarios			
	Bookmarks	Navigation	Automatic domain	Other browsers
www.baidu.com	Y	Y	Y	Y
www.amazon.com	N	N	N	Y
mail.qq.com	Y	Y	Y	Y
www.taobao.com	Y	Y	Y	Y
m.kaskus.co.id	Y	Y	Y	Y

According to the results, even the site has deployed HSTS, there is a possibility of being hijacked. As for the websites only provide HTTPS and did not deploy HSTS, the attack will succeed every time when users visit it with HTTP request. These websites which have deployed HSTS but were not in the preload list or did not fully deploy HSTS during the redirection process, will be in danger as well. And the websites in the preload list would be hijacked in the browser that did not support preload list.

[7] In this test, other browsers means those which did not support preload list, like UC browser, Sogou browser.

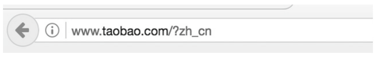
www.taobao.com/?zh_cn

Fig. 5. taobao.com

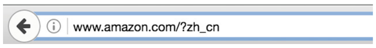
www.amazon.com/?zh_cn

Fig. 6. amazon.com

6 Discussion

To deploy HTTPS and HSTS in a more secure way, we must take measures from both ends.

6.1 Browser/User

Browsers should provide a mechanism which can check the scheme of domain. If the response of the real site is contrary to the check, the browser should give a strict warning and it can prompt the user to manually enter the URL with scheme so as to avoid be hijacked. A preload list is not enough and it is so strict that many websites did not meet the requirements. Many serious problems occur in preload list, so the browsers vendors have to adapt the list to cope with the dynamic change. The list should be accepted by more browsers. Apart from these, active defense should be considered by browsers. Browsers should actively establish a secure bookmark and navigation mechanism. They can not allow users to modify the bookmarks casually, which has security risks. The browsers should check whether the scheme in the bookmark has changed and the navigation in browsers should be updated in time to avoid outdated URL.

As for users, we strongly recommend that users observe the scheme carefully when adding a bookmark. Most of times users should not tend to click the navigation links on unfamous sites, owing to the deceptive attack, particularly financial and other sensitive sites, like online banking, electronic commerce. Users should go to the HTTPS version of the site from users' machine while using a secure network, and then bookmark that page. Besides, always open the site by accessing the bookmark whenever users want to visit that page. At the same time, users should pay attention to the jumps from HTTPS to HTTP or vice versa. It will do great help for users to install the software plugins, like HTTPS Everywhere or ForceTLS, which may reduce the occurrence of stripping attack.

6.2 Server/Website

As for sites owners, first, they should enable SSL site wide and use HTTPS as much as possible. Then the sites should enable HSTS policy and Cert Pinning, but also be careful when dealing with each step of process, i.e., sites should better deploy HSTS in every packet of HTTPS response. In order to let HTTPS and HSTS work better, the sites should enable secure cookies and use mixed content in HTTPS pages as less as possible. Ensure that all cookies are served with the secure attribute, so that user's browsers will only send those cookies back over SSL-protected connections and never disclose them over any non-SSL links. Finally, the sites should use HTTPS everywhere and join the preload list as soon as possible.

7 Related Work

7.1 HTTPS Security

Many researchers studied about HTTPS security in recent years and most focused on TLS/SSL security. Client-end TLS software and non-browser software have defects on implementation and the root causes of these vulnerabilities are badly designed APIs of SSL implementations or negligence [20,21]. Great security threats are present in SSL proxys as well [10,20,22], where proxy can break the end to end security. Another thing that affects SSL security is the certificate. Many studies were dedicated to solve the problems of certificate management, the private key management and certificate validity [23,24]. There are several evaluations of large-scale SSL deployments problems [25,26]. Warnings from browsers are pivotal for users to avoid attacks, so several researchers have investigated the effectiveness of warnings [27,28]. This paper is different from them and we mainly focus on the threat of HTTPS stripping attack.

7.2 HSTS Security

HSTS was born to ensure that HTTPS performs better. However, many flaws exist in deployment of HSTS as well. Researchers have found that even though these protocols are implemented, bad practices prevent them from actually providing the additional security they are expected to provide [29]. They studied the implementations of HSTS in Firefox, Chrome and IE, and found several potential attack scenarios. Kranch and Bonneau [12] have done a measurement about HSTS and HPKP policy. They mainly found errors for sites with HSTS headers and analyzed the security of cookies. Our work is largely distinct from them. We focused on redirection problems and carried on the detailed classification to the setting errors. Besides, we have done the experiments on mobile platforms and have found defects in bookmarks and navigations panels.

7.3 Stripping Attack Studies

Since Marlinspike et al. [7] published the *sslsttrip* attack, researchers have been working on it for years. To overcome *sslstrip* attacks, many schemes have been proposed. ForceHTTPS [30] is a simple browser security mechanism that web sites or users can use to opt in to stricter error processing, but it needs users to install extra plugins. Zhao et al. [31] presented a new defense scheme according to secure cookie as well. However, these defense schemes can only succeed under the specific environment, which can not defend against the attack we implemented perfectly. In our paper, we have strengthened the previous *sslstrip* attack and successfully launched the enhanced stripping attack to various websites in simulated scenarios.

8 Conclusion

In this paper, we have found many sites owner or developers did not understand the HSTS policy well. We have exposed that a lot of top-ranking sites had incorrect deployment and redirection problems. Many websites did deploy HSTS policy, but several redirections occurred when we visited them. Unfortunately, the HSTS is not fully deployed during the redirections, still left the possibility of being attacked. In addition, lots of instructions field setting errors were found by us. Moreover, schemes in bookmarks and navigation are forgotten by users easily. After our investigation, we found that the default HTTP supplementation mode of bookmarks has a security problem and the default address in the navigation is at risk of being downgraded as well. To test the risk of these defects, we designed an enhanced HTTPS stripping attack, which strengthened the previous *sslstrip* attack. The success rate is high based on the pitfalls we found.

In summation, our paper can give some guidance to the sites who want to deploy HSTS correctly. Besides, due to the fact that the defects we found contribute to the stripping attack, our work is able to help users to reduce the risks of being attacked. We hope our research enables HTTPS and HSTS protocol to provide more efficient service and make users' information more safe compared with now.

Acknowledgements. This work was partly supported by the National Key Research and Development Program of China under No. 2016YFB0800102 and 2016YFB0800201, the National High Technology Research and Development Program of China under No. 2015AA015602 and 2015AA016103, the Key Research and Development Program of Zhejiang Province under No. 2017C01064 and 2017C01055, the Fundamental Research Funds for the Central Universities, the NSFC under No. 61772466, the Alibaba-Zhejiang University Joint Research Institute for Frontier Technologies (A.Z.F.T.) under Program No. XT622017000118, and the CCF-Tencent Open Research Fund under No. AGR20160109.

References

1. Rescorla, E.: HTTP over TLS (2000)
2. Berners-Lee, T., Fielding, R., Frystyk, H.: "RFC 1945: hypertext transfer protocol? HTTP/1.0, May 1996." Status: INFORMATIONAL 61 (2005)
3. Freier, A., Karlton, P., Kocher, P.: The secure sockets layer (SSL) protocol version 3.0 (2011)
4. Dierks, T., Rescorla, E.: "RFC 5246: the transport layer security (TLS) protocol." The Internet Engineering Task Force (2008)
5. Google Transparency Report. https://www.google.com/transparencyreport/https/?hl=zh-CN
6. Let's Encrypt. https://letsencrypt.org
7. Moixe, M.: New tricks for defeating SSL in practice. Technical report, BlackHat Conference, USA (2009)

8. Hodges, J., Jackson, C., Barth, A.: "RFC 6797: HTTP strict transport security (HSTS)". IETF (2010). https://tools.ietf.org/html/rfc6797
9. Sivakorn, S., Polakis, I., Keromytis, A.D.: The cracked cookie jar: HTTP cookie hijacking and the exposure of private information. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 724–742. IEEE Press (2016)
10. Chen, S., et al.: Pretty-bad-proxy: an overlooked adversary in browsers' HTTPS deployments. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 347–359. IEEE Press (2009)
11. Extract DNSSEC support statistics for the top 1 million hosts of the Alexa database. https://github.com/jefmathiot/dnssec-stats
12. Kranch, M., Bonneau, J.: Upgrading HTTPS in mid-air: an empirical study of strict transport security and key pinning. In: NDSS (2015)
13. HSTS Preload List Submission. https://hstspreload.appspot.com
14. Run a daily status scan of the official preload list. http://github.com/chromium/hstspreload.org/issues/35
15. Desktop Browser Market Share. https://www.netmarketshare.com/browser-market-share.aspx?qprid=0qpcustomd=0
16. Bettercap. https://www.bettercap.org
17. Framework for Man-In-The-Middle attacks. https://github.com/byt3bl33d3r/MITMf
18. SSL Frontend hijack. https://www.cnblogs.com/index-html/p/ssl-frontend-hijack.html
19. HTML5. https://www.w3.org/TR/2014/REC-html5-20141028/browsers.html#window
20. de Carnavalet, X.C., Mannan, M.: Killed by proxy: analyzing client-end TLS interception software. In: Network and Distributed System Security Symposium (NDSS 2016), San Diego, CA, USA (2016)
21. Georgiev, M., et al.: The most dangerous code in the world: validating SSL certificates in non-browser software. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 38–49. ACM (2012)
22. Soghoian, C., Stamm, S.: Certified lies: detecting and defeating government interception attacks against SSL (short paper). In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 250–259. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27576-0_20
23. Szalachowski, P., Matsumoto, S., Perrig, A.: PoliCert: secure and flexible TLS certificate management. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 406–417. ACM (2014)
24. Cangialosi, F., et al.: Measurement and analysis of private key sharing in the https ecosystem. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 628–640. ACM (2016)
25. Bates, A., et al.: Forced perspectives: evaluating an SSL trust enhancement at scale. In: Proceedings of the 2014 Conference on Internet Measurement Conference, pp. 503–510. ACM (2014)
26. Holz, R., et al.: 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, pp. 427–444. ACM (2011)
27. Sunshine, J., et al.: Crying wolf: an empirical study of SSL warning effectiveness. In: USENIX Security Symposium, pp. 399–416 (2009)
28. Akhawe, D., Felt, A.P.: Alice in Warningland: a large-scale field study of browser security warning effectiveness. In: Usenix Security, pp. 257–272 (2013)

29. de los Santos, S., Torrano, C., Rubio, Y., Brezo, F.: Implementation state of HSTS and HPKP in both browsers and servers. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 192–207. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_12
30. Jackson, F., Barth, A.: Protecting high-security web sites from network attacks. In: Proceedings of the 17th International World Wide Web Conference (WWW 2008) (2008)
31. Zhao, S., Yang, W., Wang, D., Qiu, W.: A new scheme with secure cookie against SSLstrip attack. In: Wang, F.L., Lei, J., Gong, Z., Luo, X. (eds.) WISM 2012. LNCS, vol. 7529, pp. 214–221. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33469-6_30