Journal of Current Science and Technology, October-December 2025 Vol. 15 No. 4, Article 146
Copyright ©2018-2025, Rangsit University ISSN 2630-0656 (Online)

Cite this article: Chanbuala, K., Puangpronpitag, D., Puangpronpitag, E., & Puangpronpitag, S. (2025). Security Analysis and mitigation of ssl stripping, homograph redirection, and keylogging attacks: A case study on Thai web platforms. *Journal of Current Science and Technology*, *15*(4), Article 146. https://doi.org/10.59796/jcst.V15N4.2025.146

Journal of Current Science and Technology

Journal homepage: https://jcst.rsu.ac.th

Security Analysis and Mitigation of SSL Stripping, Homograph Redirection, and Keylogging Attacks: A Case Study on Thai Web Platforms

Khathawut Chanbuala[1], Darunee Puangpronpitag[1], Egachai Puangpronpitag[2], and Somnuk Puangpronpitag[1,*]

[1]Information Security & Advanced Network (ISAN) Research Group, Mahasarakham University, Mahasarakham 44150, Thailand.
[2]Department of Special Investigation (DSI), Ministry of Justice, Bangkok 10210, Thailand.

*Corresponding author; E-mail: somnuk.p@msu.ac.th

Received 30 January 2025; Revised 3 May 2025; Accepted 23 July 2025; Published online 20 August 2025

Abstract

The cybersecurity of critical Thai digital infrastructure is a pressing concern for national security. This research, conducted in collaboration with Thailand's Department of Special Investigation (DSI), presents a comprehensive security assessment of 27 specifically selected websites across financial, commercial, and educational sectors. Our investigation focuses on three critical attacks: SSL stripping, homograph redirection attacks, and keylogger injection. The findings reveal that 96.3% (26/27) of the examined websites are vulnerable to SSL stripping attacks due to inadequate HTTP Strict Transport Security (HSTS) implementation. Notably, even the sole website with proper HSTS Preload configuration demonstrated susceptibility to homograph attacks. Furthermore, all examined websites were susceptible to keylogger injection after successful Man-in-the-Middle (MITM) attacks, even when password hashing was used. To counter these threats, we propose an enhanced security framework integrating a Time-based Salted Hash Password (TSHP) mechanism and an On-Screen Keyboard (OSK) for login interfaces. Experimental evaluation shows that TSHP improves resistance to brute-force and replay attacks by generating dynamic, time-variant hashes, while OSK input prevented 100% of JavaScript keylogger captures when used exclusively. These countermeasures offer practical, low-cost solutions to strengthen Thailand's digital services, enabling immediate deployment without infrastructure overhaul. Our findings provide actionable recommendations for policymakers and system administrators to enhance the cybersecurity posture of Thai web platforms, with broader implications for securing digital economies globally.

Keywords: *SSL stripping; keylogging; homograph attack; HSTS implementation; time-based salted-hash password*

1. Introduction

Thailand's rapid digitalization of financial, commercial, and educational services has heightened concerns about cybersecurity vulnerabilities. Despite the widespread adoption of HTTPS protocols, many critical web platforms remain exposed to input-level threats that exploit weaknesses prior to encryption, particularly Man-in-the-Middle (MITM) attacks that leverage Secure Sockets Layer (SSL) stripping, homograph redirection, and JavaScript-based keylogger injection. These threats bypass conventional transmission-layer defenses, posing significant risks to user privacy and credential security, especially during authentication.

Several studies have evaluated SSL stripping attacks. However, most have overlooked the practical implications of input-level interception under active MITM conditions. Recent assessments by Thai government cybersecurity units, such as Puangpronpitag & Puangpronpitag (2022), Khachenrum et al. (2023), and Chanbuala et al. (2024), have revealed persistent weaknesses in HTTPS deployment and input protection across critical public and private web services. Although HTTPS is widely adopted, its effectiveness is compromised when HTTP Strict Transport Security (HSTS) is improperly configured. Puangpronpitag & Puangpronpitag (2022), Khachenrum et al. (2023), and our previous study (Chanbuala et al., 2024) focused on SSL stripping in Thai e-banking websites. However, they did not consider homograph redirection attacks, which can bypass even preloaded HSTS mechanisms. Puangpronpitag & Puangpronpitag (2022) and Khachenrum et al. (2023) also failed to address the threat of keylogger injection, which renders client-side password hashing ineffective. Moreover, their experiments were limited in scope, covering only a subset of websites. In contrast, the present study evaluates all 27 high-priority websites (spanning financial, educational, and commercial sectors) identified by the Technology and Cyber Crime Division of Thailand's Department of Special Investigation (DSI). The DSI has emphasized the urgent need for a comprehensive assessment of SSL stripping, homograph redirection, and keylogger injection, as well as robust defenses against hashed password brute-force attacks and input-level interception.

To address this gap, this study sets out three main objectives. First, we aim to identify and assess the prevalence and root cause of input-level attacks, specifically SSL stripping, homograph redirection, and JavaScript-based keylogger injection attacks on 27 Thai websites (selected by Thailand DSI's Technology and Cyber Crime Division) across financial, educational, and commercial sectors. Second, we systematically analyze the effectiveness of current mitigation techniques, including HSTS configurations and password hashing, under active MITM conditions. Third, based on the findings, we propose and evaluate two practical countermeasures: (1) a Time-based Salted-Hash Password (TSHP) mechanism to resist brute-force and replay attacks, and (2) an On-Screen Keyboard (OSK) to defend against JavaScript-based keylogging. The following research questions guide these objectives: Q1) To what extent are Thai web platforms vulnerable to SSL stripping and keylogging attacks despite HTTPS adoption? Q2) How effective are current HSTS and password protection mechanisms in preventing input-level credential theft? Q3) Can TSHP and OSK significantly enhance input-layer security in practical deployment scenarios? We hypothesize that H1) most Thai websites lack robust HSTS implementation, rendering them susceptible to SSL stripping and homograph attacks, and H2) the proposed TSHP and OSK mechanisms will significantly reduce the success rate of credential theft in controlled attack scenarios.

The impact of our work is threefold: (1) it provides actionable findings to DSI's IT Crime department, supporting targeted remediation across Thai web infrastructure; (2) it offers system administrators practical guidelines for deploying input-level protections and securing HSTS configurations against downgrade and injection attacks; and (3) it presents a hybrid authentication model that integrates time-sensitive hashing and OSK-based input isolation, offering a viable defense for password-based authentication.

The remainder of this paper is organized as follows. Section 2 presents background information and related work. Section 3 describes the research method, experimental setup, tools, experimental test-bed, ethical considerations, attack execution and validation, and proposed solutions along with evaluation frameworks. Section 4 presents the results of our attacking experiments. Sections 5 and 6 elaborate on the design and evaluation of our proposed solutions (TSHP and OSK). Section 7 discusses implications, limitations, and future work, followed by conclusions in Section 8.

2. Background & Related Work

2.1 Hypertext Transfer Protocol Secure (HTTPS) and SSL Stripping Attacks

HTTPS (Rescorla, 2000) is a secure version of HTTP, utilizing Transport Layer Security (TLS), formerly known as Secure Sockets Layer (SSL), to prevent eavesdroppers from accessing sensitive information. However, there have been several proposals to intercept HTTPS connections, particularly through SSL stripping attacks by Marlinspike (2009). SSL stripping is an MITM attack that downgrades a secure HTTPS connection to an unsecured HTTP connection. Together with user ignorance, even if a "not secure" warning appears, this attack enables attackers to capture usernames and passwords in plaintext, gaining unauthorized access to accounts without alerting the victim.

2.2 HTTP Strict Transport Security (HSTS)

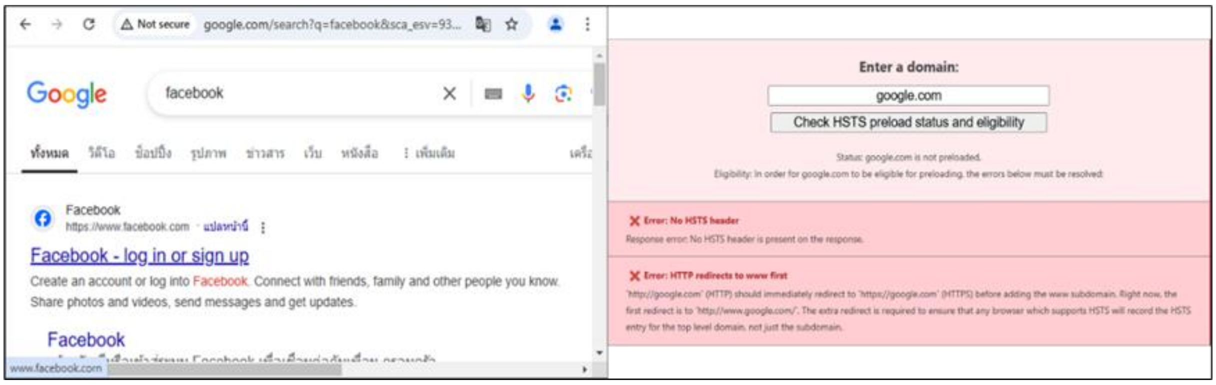
HTTP Strict Transport Security (HSTS) (Hodges et al., 2012) is a critical web security protocol that enforces HTTPS connections between browsers and web servers, effectively preventing connection downgrades to insecure HTTP and mitigating SSL stripping attacks. The mechanism operates by transmitting a "Strict-Transport-Security" HTTP header containing a policy duration parameter, instructing browsers to maintain HTTPS connectivity. However, the standard HSTS implementation proved vulnerable to HSTS hijacking attacks (Buffermet, 2024), which could circumvent security enforcement and facilitate Denial of Service (DoS) attacks on HSTS-configured websites. These vulnerabilities led to the deprecation of non-preloaded HSTS configurations and the subsequent introduction of the preload mechanism. While preloaded HSTS with preload lists represents an essential defense against SSL stripping attacks, its adoption remains limited. Recent studies have documented that numerous websites either lack HSTS configuration entirely or utilize non-preloaded HSTS implementations, leaving them susceptible to SSL stripping attacks and unauthorized data interception.

2.3 Homograph Attacks

Phishing websites often use domain names that closely resemble legitimate websites to deceive potential victims. These fraudulent sites typically appear as advertisements (ads) in Google search results when users cannot remember the full URL of the website and search for it. Without noticing the anomalies, users can become victims. There were several phishing cases of Thai banks via Google ads, such as Jirawankul (2015), Bangkok Bank (2023). Although Google has since implemented enhanced measures to prevent fraudulent advertising, Google Search remains vulnerable to similar security risks. Specifically, the Google Search page lacks HSTS Preload implementation (as shown in the right part of Figure 1), creating opportunities for SSL stripping attacks. Additionally, attackers can employ homograph redirection attack techniques by modifying the Top-Level Domain (TLD) from '.com' to '.corn' to redirect users to fraudulent websites, even when the legitimate sites implement HSTS Preload (as demonstrated in Figure 1). This attack scheme represents a persistent threat to user security, bypassing robust security implementations.

2.4 Keylogger Injection Attacks

Keylogger injection represents a client-side attack vector wherein malicious JavaScript code is inserted into webpages to intercept user input before encryption or hashing. These attacks typically follow successful MITM compromises, leveraging tools such as Bettercap's keylogger.js module and associated caplets (Bettercap, 2024) to execute the attack. Unlike system-level keyloggers, which operate at the operating system layer, browser-based keyloggers function within the client's browser environment. The attacks can be launched through intercepted network traffic. They are commonly introduced through MITM, Cross-Site Scripting (XSS), or compromised third-party resources. The injected scripts attach event listeners (e.g., keydown, input) to form fields of interest, particularly authentication components. This allows real-time capture of sensitive data, including credentials. This attack is particularly concerning because interception occurs before encryption or hashing processes, rendering properly secured sites vulnerable. Figure 2 illustrates a JavaScript payload injected into the Document Object Model (DOM) designed to capture and exfiltrate user credentials from a standard login form. This study also investigates the security implications of this attack.


Not secure google.com/search?q=facebook&sca_esv=93...

facebook

ทั้งหมด วิดีโอ ช็อปปิ้ง รูปภาพ ข่าวสาร เว็บ หนังสือ ⋮ เพิ่มเติม

Facebook
https://www.facebook.com · แปลหน้านี้

Facebook - log in or sign up

Create an account or log into Facebook. Connect with friends, family and other people you know. Share photos and videos, send messages and get updates.

Facebook

www.facebook.com

Enter a domain:

google.com

Check HSTS preload status and eligibility

Status: google.com is not preloaded.

Eligibility: In order for google.com to be eligible for preloading, the errors below must be resolved:

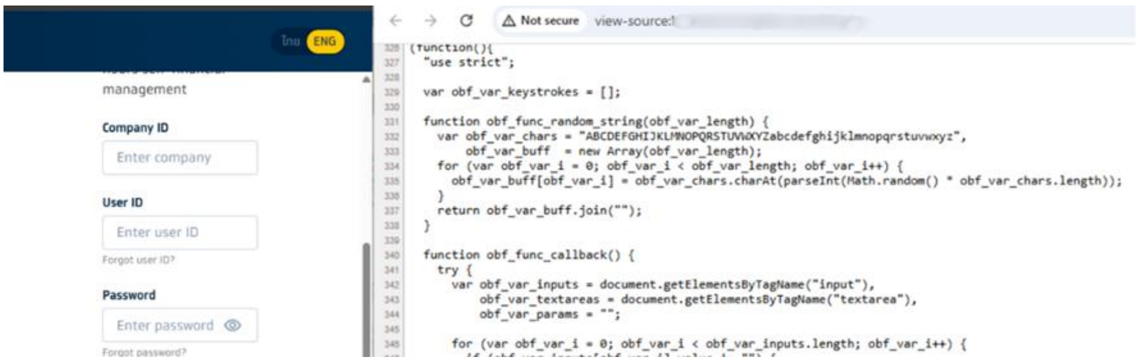
✗ Error: No HSTS header

Response error: No HSTS header is present on the response.

✗ Error: HTTP redirects to www first

'http://google.com' (HTTP) should immediately redirect to 'https://google.com' (HTTPS) before adding the www subdomain. Right now, the first redirect is to 'http://www.google.com/'. The extra redirect is required to ensure that any browser which supports HSTS will record the HSTS entry for the top level domain, not just the subdomain.

Figure 1 Exploiting non-HSTS preload vulnerability in Google Search



management

Company ID
Enter company

User ID
Enter user ID
Forgot user ID?

Password
Enter password
Forgot password?

```
(function(){
  "use strict";

  var obf_var_keystrokes = [];

  function obf_func_random_string(obf_var_length) {
    var obf_var_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz",
        obf_var_buff  = new Array(obf_var_length);
    for (var obf_var_i = 0; obf_var_i < obf_var_length; obf_var_i++) {
      obf_var_buff[obf_var_i] = obf_var_chars.charAt(parseInt(Math.random() * obf_var_chars.length));
    }
    return obf_var_buff.join("");
  }

  function obf_func_callback() {
    try {
      var obf_var_inputs = document.getElementsByTagName("input"),
          obf_var_textareas = document.getElementsByTagName("textarea"),
          obf_var_params = "";

      for (var obf_var_i = 0; obf_var_i < obf_var_inputs.length; obf_var_i++) {
```

Figure 2 JavaScript payload for keystrokes capture on the victim's login page

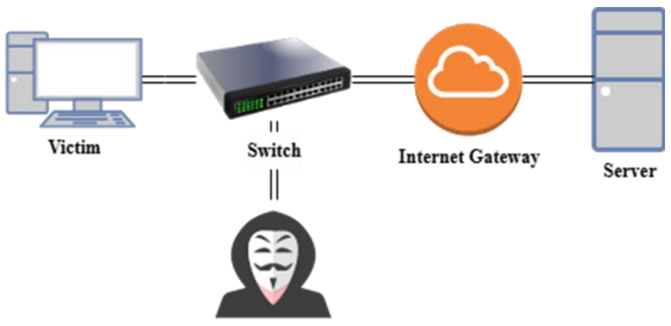


Figure 3 Network test-bed setup for MITM and SSL stripping attacks

3. Research Method

3.1 Experimented Websites and Ethical Considerations

The selection of websites for this study was based on recommendations from the Technology and Cyber Crime Division of Thailand's Department of Special Investigation (DSI). The DSI identified three high-risk sectors where vulnerabilities could significantly impact public trust and national cybersecurity posture: financial institutions, educational platforms, and commercial services. Twenty-seven websites were selected and categorized as follows: 13 e-banking platforms, 12 university registration systems, and two e-commerce websites.

All experiments were conducted under the oversight of the DSI, with written authorization to simulate attacks as a collaborative research project in a controlled environment. Only login pages and endpoints were examined. No server-side modifications or intrusion attempts were made. The experiments were conducted under strictly controlled conditions without engaging in any real attacks on actual users. Although SSL stripping and traffic interception were employed, no sensitive information was accessed. The captured usernames and passwords were simulated data (not real user data) for testing purposes. All data collected during testing were limited to browser-side events, HTTP traffic patterns, and other traffic traces necessary for evaluating SSL stripping, homograph deception, and keylogger injection scenarios. Furthermore, this paper anonymizes the names of experimented websites using abbreviations and does not disclose the actual websites.

3.2 Tools and Test-bed

Figure 3 illustrates our experimental test-bed, which consists of a simple LAN connected to the internet, with special controls in place to enable ARP poisoning of both the gateway and the victim for MITM testing. The victim machine (running Windows 10) uses Chrome to access university registration, e-banking, and e-commerce websites via a gateway switch. The attacker, connected to the same network switch, uses Bettercap v.33 (Bettercap, 2024) on Kali Linux v.2024.3 (Offensive Security, 2024) to perform MITM attacks, including SSL stripping, homograph redirection, keylogger injection, and data sniffing. Wireshark v.4.4.0-1 (Wireshark Foundation, 2023) is employed for packet inspection. Figure 4 demonstrates a compromised login scenario. After SSL stripping is executed, the victim's browser displays a "Not secure" warning, an alert frequently ignored by users.

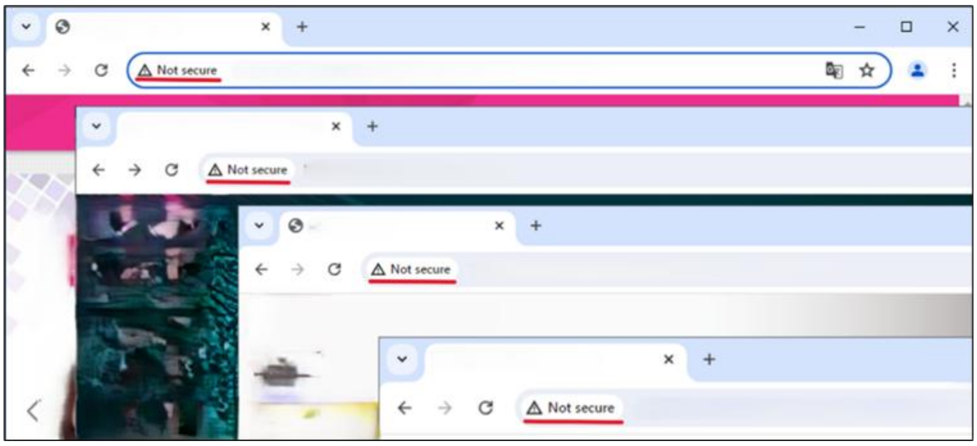


Figure 4 Browser security warnings triggered during SSL stripping attack

3.3 Attack Execution and Validation Criteria

Each selected website was simulated with three attack scenarios: SSL stripping, homograph redirection, and JavaScript-based keylogger injection. SSL stripping was considered successful if the HTTPS upgrade could be downgraded to HTTP. We assumed users might ignore the browser's "Not Secure" warning, as documented in several real-world cases and studies. Before each test, the victim's browser history was cleared to simulate first-time access, replicating real-world attack scenarios where malware reset session states to bypass HTTPS protections. Homograph redirection was validated by replacing a legitimate domain with a visually similar one (e.g., ".com" to ".corn") and observing whether users were redirected to the impersonated domain via modified search results. Keylogger injection was confirmed when the attacker's machine captured keystrokes from input fields in plaintext before submission.

All attacks were carried out using customized Bettercap caplets, modified to optimize SSL downgrading, enable keylogger JavaScript payload injection, and facilitate real-time DNS response redirection. Each test was repeated 10 times per website to ensure consistency. The success criteria for each attack were binary (Success/Fail), and results were documented in tabular form for analysis. Browser behaviors (e.g., security warnings, redirections) were logged using screen capture tools and HTTP packet inspection via Wireshark.

3.4 Our Proposed Solutions and Evaluation Frameworks

To address the weaknesses found from our attacking experiments, we propose two mechanisms:

1. Time-based Salted Hash Password (TSHP): A method that combines time-based one-time password (TOTP) with salted hashing on the client side, preventing credential reuse and reducing susceptibility to replay and brute-force attacks.
2. On-Screen Keyboard (OSK): A browser-based virtual keyboard that mitigates keylogger injection attacks by bypassing traditional input event capture via JavaScript.

Sections 5 (TSHP) and 6 (OSK) thoroughly present the details of the mechanisms and their evaluations. An evaluation framework was designed to assess the effectiveness of the proposed solutions (TSHP and OSK), focusing on three key aspects: attack resistance, performance, and usability. They can be explained as follows.

Attack Resistance: In MITM scenarios, TSHP was evaluated against replay, brute-force, and rainbow-table attacks. Similarly, OSK was tested by attempting JavaScript-based keylogger injections, with success criteria for preventing plaintext keystroke leakage during interactions with the DOM.

Performance: Each experiment was run 30 times, and the results were averaged with a 95% confidence interval. The average login latency was used to evaluate computational impact, including client-side hashing and Time-based One Time Password (TOTP) (M'Raihi et al., 2011) generation. For OSK, the typing delay and total submission time were recorded.

Usability: Small-scale tests were conducted with 30 participants to observe input accuracy and interaction comfort. Both secured and unsecured login forms were tested for comparison.

Table 1 SSL stripping and password sniffing results

Site ID	HSTS Config	SSL Stripping Success	Password Sniffing
Reg1	×	Yes	Cleartext
Reg2	×	Yes	Cleartext
Reg3	Non-preloaded	Yes	Cleartext
Reg4	×	Yes	Cleartext
Reg5	Non-preloaded	Yes	Cleartext
Reg6	Non-preloaded	Yes	Cleartext
Reg7	×	Yes	Cleartext
Reg8	×	Yes	Cleartext
Reg9	×	Yes	Cleartext
Reg10	Non-preloaded	Yes	Cleartext
Reg11	×	Yes	Cleartext
Reg12	×	Yes	Cleartext
Ebank1	Non-preloaded	Yes	Cleartext
Ebank2	Non-preloaded	Yes	Cleartext
Ebank3	Non-preloaded	Yes	Cleartext
Ebank4	Non-preloaded	Yes	Hashed
Ebank5	Non-preloaded	Yes	Cleartext
Ebank6	Non-preloaded	Yes	Hashed
Ebank7	Non-preloaded	Yes	Cleartext
Ebank8	×	Yes	Hashed
Ebank9	Non-preloaded	Yes	Cleartext
Ebank10	Non-preloaded	Yes	Cleartext
Ebank11	×	Yes	Cleartext
Ebank12	Non-preloaded	Yes	Cleartext
Ebank13	✓ (preloaded)	No	-
Ecommerce1	Non-preloaded	Yes	Cleartext
Ecommerce2	×	Yes	Hashed

4. Results

4.1 Results of Experiment 1: SSL Stripping

The first set of experiments focused on SSL stripping attacks and password sniffing on the login pages of 27 targeted websites. The experimental results, as presented in Table 1, indicated that 26 websites were vulnerable to SSL stripping attacks, with 22 of these 26 websites having their passwords exposed in cleartext. Four websites (ebank4, ebank6, ebank8, ecommerce2) employed password hashing to protect against attackers. Only one website (ebank13) successfully withstood both SSL stripping and password sniffing attacks. A significant observation is that 11 compromised sites lacked HSTS implementation. The remaining 15 websites implemented HSTS with a non-preloaded configuration, which was obsolete and no longer functional. Only one website (ebank13) utilized the secure preloaded HSTS configuration.

4.2 Results of Experiment 2: Homograph Redirection

This second set of experiments aimed to evaluate the impact of homograph redirection attacks. We selected an e-banking website from

Table **1** that effectively prevents SSL stripping attacks through HSTS Preload implementation. Before initiating the attack, the attacker configured the hstshijack.cap file, located in Bettercap's caplets folder. The configuration parameters included:

hstshijack.targets: specifying attack targets

hstshijack.replacements: defining website content modifications

dns.spoof.domains: specifying domains to be spoofed

To execute the attack, the attacker added relevant domains, including Google's domain and the target website's domain, modifying their TLD from '.com' to '.corn'. These configurations enabled the attacker to bypass HSTS Preload restrictions and conduct effective attacks. The detailed configuration file example is shown in Figure 5.

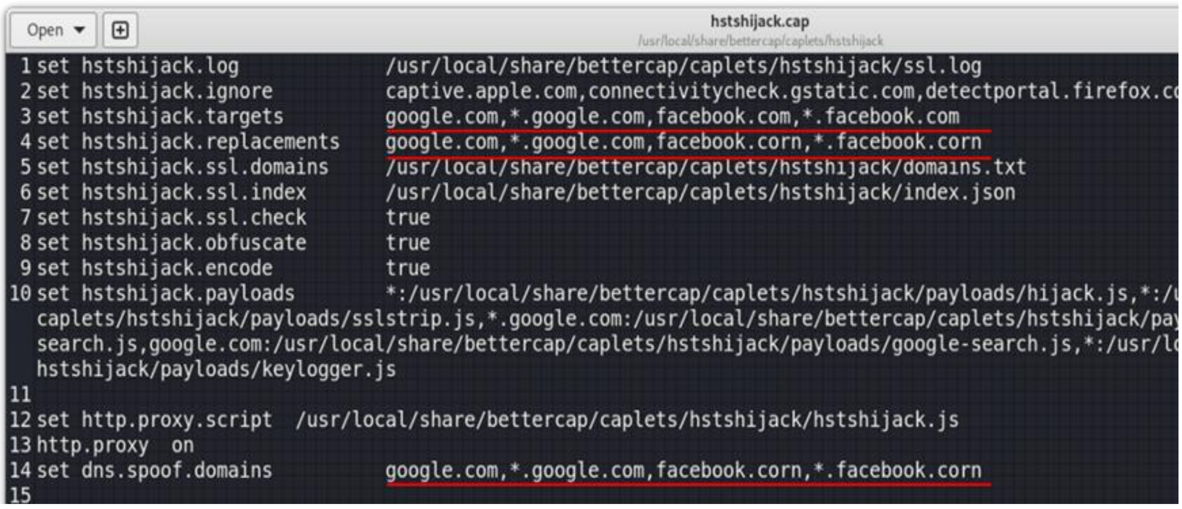
The experimental results (illustrated in Figure 6) demonstrated that websites implementing HSTS Preload effectively mitigate SSL Stripping attacks but remain vulnerable to homograph redirection attacks. When users are successfully deceived and access the fraudulent website, attackers can capture sensitive information, including passwords.

To address these issues, we suggest that domain names be easy to remember and that users be encouraged to access websites directly to reduce the risks associated with searching through Google. Additionally, Google should enforce HSTS Preload on all its services and monitor for fake advertisements and domains. Browsers should detect unusual TLDs, such as '.corn', and display warnings to users.

4.3 Results of Experiment 3: Keylogger Injection

This experiment assessed the injection of JavaScript-based keyloggers through MITM manipulation of unsecured HTTP traffic.

Table 2 summarizes the outcome of keylogger injection tests. All 27 websites, including 13 financial platforms, 12 university registration systems, and two e-commerce portals, were found to be vulnerable. The script keylogger.js successfully captured login credentials before submission in every case, including those with hashed password implementation and HSTS configuration. No platform demonstrated any form of input-level protection.

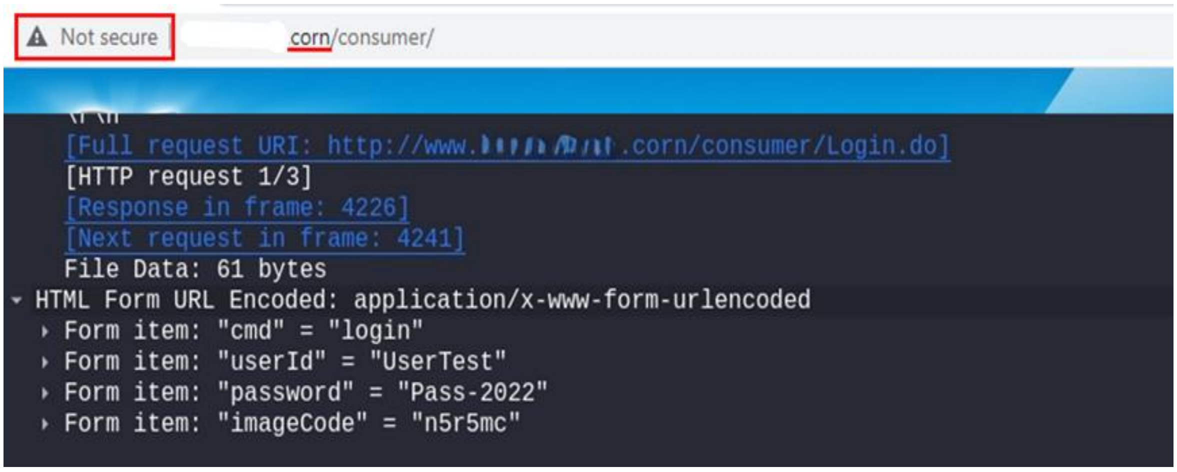


```
set hstshijack.log             /usr/local/share/bettercap/caplets/hstshijack/ssl.log
set hstshijack.ignore          captive.apple.com,connectivitycheck.gstatic.com,detectportal.firefox.c
set hstshijack.targets         google.com,*.google.com,facebook.com,*.facebook.com
set hstshijack.replacements    google.com,*.google.com,facebook.corn,*.facebook.corn
set hstshijack.ssl.domains     /usr/local/share/bettercap/caplets/hstshijack/domains.txt
set hstshijack.ssl.index       /usr/local/share/bettercap/caplets/hstshijack/index.json
set hstshijack.ssl.check       true
set hstshijack.obfuscate       true
set hstshijack.encode          true
set hstshijack.payloads        *:/usr/local/share/bettercap/caplets/hstshijack/payloads/hijack.js,*:/u
caplets/hstshijack/payloads/sslstrip.js,*.google.com:/usr/local/share/bettercap/caplets/hstshijack/pay
search.js,google.com:/usr/local/share/bettercap/caplets/hstshijack/payloads/google-search.js,*:/usr/l
hstshijack/payloads/keylogger.js

set http.proxy.script  /usr/local/share/bettercap/caplets/hstshijack/hstshijack.js
http.proxy  on
set dns.spoof.domains          google.com,*.google.com,facebook.corn,*.facebook.corn

```

Figure 5 Bettercap HSTS hijack configuration for combined SSL stripping and homograph attack



```
    [Full request URI: http://www.[redacted].corn/consumer/Login.do]
    [HTTP request 1/3]
    [Response in frame: 4226]
    [Next request in frame: 4241]
    File Data: 61 bytes
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▸ Form item: "cmd" = "login"
  ▸ Form item: "userId" = "UserTest"
  ▸ Form item: "password" = "Pass-2022"
  ▸ Form item: "imageCode" = "n5r5mc"
```

Figure 6 Homograph attack results on HSTS-preloaded website

Table 2 Keylogger injection results

Site ID	HSTS Preload	Password Hashed	Keylogger Success	Note
Reg1	No	No	Yes	Plaintext credential captured
Reg2	No	No	Yes	Plaintext credential captured
Reg3	No	No	Yes	Plaintext credential captured
Reg4	No	No	Yes	Plaintext credential captured
Reg5	No	No	Yes	Plaintext credential captured
Reg6	No	No	Yes	Plaintext credential captured
Reg7	No	No	Yes	Plaintext credential captured
Reg8	No	No	Yes	Plaintext credential captured
Reg9	No	No	Yes	Plaintext credential captured
Reg10	No	No	Yes	Plaintext credential captured
Reg11	No	No	Yes	Plaintext credential captured
Reg12	No	No	Yes	Plaintext credential captured
Ebanking1	No	No	Yes	Plaintext credential captured
Ebanking2	No	No	Yes	Plaintext credential captured
Ebanking3	No	No	Yes	Plaintext credential captured
Ebanking4	No	Yes	Yes	Plaintext credential captured
Ebanking5	No	No	Yes	Plaintext credential captured
Ebanking6	No	Yes	Yes	Plaintext credential captured
Ebanking7	No	No	Yes	Plaintext credential captured
Ebanking8	No	Yes	Yes	Plaintext credential captured
Ebanking9	No	No	Yes	Plaintext credential captured
Ebanking10	No	No	Yes	Plaintext credential captured
Ebanking11	No	No	Yes	Plaintext credential captured
Ebanking12	No	No	Yes	Plaintext credential captured
Ebanking13	Yes	No	Yes	Redirected via homograph
Ecommerce1	No	No	Yes	Plaintext credential captured
Ecommerce2	No	Yes	Yes	Plaintext credential captured

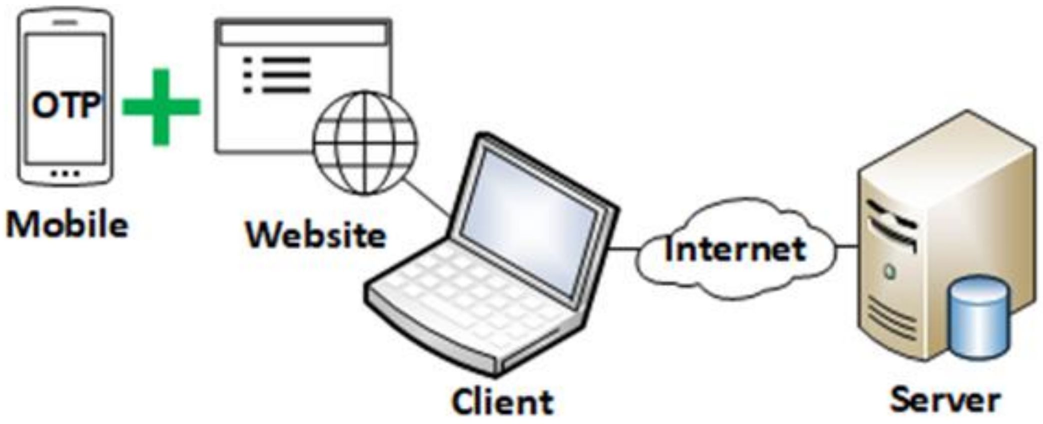


Figure 7 Salted Hash Password authentication with mobile OTP as a dynamic salt

Using Bettercap, we injected a keylogger.js script into intercepted login pages. This script captured input events from the username and password fields. It transmitted the data to a local logging server before form submission, allowing us to acquire credentials in plaintext in every case.

Notably, websites (ebank4, ebank6, ebank8, ecommerce2) utilizing password hashing mechanisms provided no adequate defense, as the keylogger intercepted data before client-side hashing occurred. This demonstrates that hashing alone cannot protect against input compromise when JavaScript injection is possible.

Additionally, even websites with proper HSTS Preload (ebank13) were still vulnerable due to homograph redirection attacks that led users to spoofed domains outside the preload scope. Thus, they were ultimately vulnerable to MITM and keylogger injection.

5. Time-based Salted Hash Password (TSHP) Mechanism

5.1 Concept and Motivation

From the first set of experiments (in Section 4), it is clear that hashing the password can be helpful as a second layer of prevention against data sniffing, particularly when SSL is stripped. Furthermore, the Salted Hash Password (SHP) effectively enhances system security by incorporating salt into the password hashing process. This technique protects against rainbow-table attacks, which utilize precomputed tables to crack hashes from all possible passwords. SHP generally makes rainbow-table attacks more challenging, as using unique salts results in different hash values even for the same password. However, while salt usage can mitigate the risk of such attacks, certain limitations may still allow successful attacks in specific scenarios: (1) Fixed Salt: If the salt is fixed or non-random, rainbow-table attacks can become easier. Precomputed rainbow tables that incorporate the fixed salt can be created in advance and utilized effectively. (2) Predictable Salt: If the salt is predictable, such as using user-specific data like usernames or system-defined constants, rainbow-table attacks may be feasible. (3) Weak Salt: If the salt is not complex or lengthy enough, such as using short or non-random salts, it can simplify hash cracking through pre-computation or the use of rainbow tables.

We propose a TSHP mechanism that integrates dynamic, time-sensitive values into the hashing process to address these issues. Instead of using a fixed salt, our method leverages a TOTP generated on the client side as a dynamic salt. This results in a unique hash each time the user logs in, even with the same password. Furthermore, since the TOTP value changes every 30 seconds and is never transmitted directly, this approach mitigates the risks posed by keyloggers and credential replay. The server validates the submitted hash by regenerating the same TOTP within a permissible time window.

5.2 Implementation Design

The TSHP mechanism enhances traditional password authentication by combining a static user password with a dynamic salt generated via TOTP. Figure 7 illustrates the login flow using TSHP. The following steps outline the client–server interaction:

TOTP Generation (Client-Side): The TOTP value is generated based on a shared secret key and the current time, following the TOTP algorithm (RFC 6238). The TOTP can be generated using a mobile authenticator app (e.g., Google Authenticator or FreeOTP), synchronized with the user's account. The app displays a six-digit code that refreshes every 30 seconds, which is then programmatically accessed or scanned (via QR or API) by the client interface.

Hashing Process: The client-side application concatenates the user's password with the TOTP value (used as a dynamic salt) and computes a secure hash (e.g., SHA3-512). This results in a time-variant hash even if the password remains unchanged.

Data Transmission: Only the computed hash is transmitted to the server. The TOTP value is not sent, minimizing the risk of interception and replay.

Server Verification: Upon receiving the hash, the server regenerates the TOTP using the known secret and current time, and verifies it against the received hash using a ± 30-second window. If a match is found, authentication is successful.

This design supports a hybrid approach, where mobile apps generate the TOTP externally while the browser or web client uses it as a local salt for hashing. It enables strong, time-based authentication without sending OTPs over the network.

5.3 Security Analysis

The TSHP mechanism significantly enhances input-level security compared to traditional password hashing. The inclusion of a time-based dynamic salt mitigates several critical attacks:

Replay Attack Mitigation: Each login attempt produces a different hash due to the changing TOTP salt. Even if an attacker captures the hash, it cannot be reused in future sessions because the corresponding TOTP will have expired.

Brute-Force and Rainbow Table Resistance: The dynamic salt makes precomputed hash tables infeasible. Using TOTP-based dynamic salts reduces the effectiveness of brute-force and dictionary attacks by limiting the valid hash window. A detailed evaluation is provided in Section 5.5.

Keylogger Bypass (Partial): While TSHP does not prevent keylogging, it ensures that the captured password alone is insufficient. Without access to a valid TOTP at the time of login, an attacker cannot recreate the correct hash for reuse.

Backward Compatibility: The scheme can be implemented on top of existing authentication systems, requiring only a minor change in how the server verifies incoming hashes.

However, some limitations exist. The mechanism relies on client-side scripting, which can be compromised if the site is hacked. It also requires time synchronization between client and server, which may cause false negatives in hash validation if the clocks drift significantly.

5.4 Performance and Compatibility

To evaluate TSHP, we measured its impact on login performance and compatibility with standard web platforms. The results are as follows:

Login Latency: Client-side hashing with a dynamic TOTP salt introduces negligible delay. In our prototype implementation using SHA3-512, the average login time increased by only 16-22 milliseconds compared to standard password submission. This overhead is acceptable for most web applications and remains imperceptible to end-users.

Device and Browser Support: TSHP is fully compatible with major web browsers and operating systems.

Infrastructure Requirements: Server-side deployment requires minimal changes. It requires only the ability to regenerate TOTP values and validate incoming hashes within a time window. Only an OTP initial secret key for each user must be stored.

Deployment Results: We deployed TSHP on a secure login prototype and tested it across different browser–device combinations. All logins were successful within the expected time window. Results are summarized in Table 3, which reports successful login rate, hash validation time, and compatibility by device class.

These findings confirm that TSHP can be deployed in real-world systems without degrading performance or requiring extensive infrastructure upgrades. All tests were conducted using the default browser configurations, without requiring any additional plugins or browser extensions. TOTP values were generated using standard mobile applications (e.g., Google Authenticator), and time synchronization was verified across all devices to ensure consistent hash validation. Security performance under brute-force conditions is evaluated in Section 5.5.

Table 3 TSHP authentication deployment results across multiple devices

Device Type	Browser	Avg. Hash Validation Time (ms)	Login Success Rate
Windows Laptop	Chrome (v120)	19.7	100%
Android Smartphone	Chrome Mobile	21.2	100%
iPhone	Safari (v16)	20.5	100%
macOS	Firefox	18.9	100%
Linux Workstation	Edge (Linux)	22.0	100%

Table 4 Estimated time to crack SHA3-512 password hashes with dynamic MOTP-based salt using RTX 4090

Password Type	Character Set	Total Hashes to Compute	Time to Crack (seconds)
?d?d?d?d?d?d	Digits (6 characters)	$10^6 \times 10^6 = 10^{12}$	197.54
?d?d?d?d?d?d?d?d	Digits (8 characters)	$10^8 \times 10^6 = 10^{14}$	19,800
?l?l?l?l?l?l?l?l	Lowercase letters (8 characters)	$26^8 \times 10^6 = 2.0882706 \times 10^{17}$	41,296,287.09

5.5 Brute-force Evaluation

To evaluate the proposed TSHP mechanism's brute-force resistance, we conducted offline hash-cracking simulations using SHA3-512 and dynamic salts generated from Mobile OTP (MOTP), refreshed every 30 seconds based on the TOTP standard. The experiment was performed using Hashcat (mode 17600) on an NVIDIA GeForce RTX 4090, with a measured cracking speed of 5056.8 million hashes per second (MH/s) (Croley, 2022). The estimated time to crack each hash was calculated using:

$$\text{Time to Crack}=\frac{\text{Total Hashes to Compute}}{\text{GPU Speed (MH/s)}}$$

Three password types were tested. The time-to-crack estimates presented in Table 4 highlight the robustness of the TSHP mechanism against brute-force attacks, particularly when combined with SHA3-512 and dynamic salts refreshed via Mobile OTP (MOTP). They also illustrate how password complexity influences resistance. For example, a weak 6-digit numeric password such as “123456” can be cracked in under 4 minutes (197.54 seconds). In real password policies, these very weak ones (or even the weaker ones) are still allowed in some systems, e.g., some tested university registration systems and some tested e-banking systems. The 30-second refresh cycle can prevent cracking for this sample. However, the real password can be weaker than this and may be cracked within 30 seconds. This finding highlights the vulnerability of systems, even when TSHP is implemented, and underscores the importance of robust password practices.

However, several organizations, including Thai financial and educational platforms, have enforced password policies that require a minimum of 8 characters with a mix of alphanumeric and special characters, following NIST SP 800-63B guidelines. Such policies align with the "complex" password type tested, which includes 95 possible characters (lowercase, uppercase, digits, and special characters). For an 8-character password, Table 4 estimates a cracking time of approximately 1.3 years. Hence, MOTP's 30-second refresh cycle ensures that even if a hash is compromised, its validity is short-lived, adding a critical layer of protection against offline attacks. While traditional salted hash mechanisms (e.g., fixed or predictable salts) remain susceptible to precomputed attacks (such as rainbow-table attacks), using time-variant salts from MOTP significantly reduces this risk.

These findings emphasize the importance of aligning password policies with the cryptographic strengths of TSHP. Organizations should enforce complex passwords with at least eight characters and consider integrating TSHP with user education to promote stronger credential creation. Additionally, the negligible performance overhead of TSHP (as shown in Table 3) makes it feasible for widespread adoption without compromising user experience, thereby enhancing security against brute-force threats in real-world deployments.

6. On-Screen Keyboard

6.1 Concept and Motivation

While TSHP can protect credentials during transmission, it cannot prevent input-level attacks, such as keylogging, before the data reaches the TSHP process. JavaScript-based keyloggers, like those included in Bettercap, intercept user input at the client side and capture passwords in cleartext, even when password hashing is employed. Our experiments (aforementioned in Section 4.3) confirmed that all tested websites, including those with client-side hashing, were vulnerable to such keylogger injection after SSL stripping and MITM attacks. This highlights a critical gap in input-level protection that TSHP alone cannot address. To mitigate this risk, we introduce an On-Screen Keyboard (OSK) as an additional defense layer. The OSK allows users to enter passwords via mouse clicks, avoiding physical keystrokes that keyloggers rely on. When combined with SHP, the OSK ensures that passwords are neither intercepted during entry nor reused after transmission, providing end-to-end input protection.

6.2 Testing the Effectiveness of OSK in Preventing Keylogger Attacks

We tested the OSK against JavaScript-based keylogger scripts by entering the password “Password2024” using four different input methods. First, using the keyboard only failed to prevent attacks, consistent with results from previous experiments. Second, using the OSK alone successfully prevented keystroke capture. Third, combining OSK and keyboard input revealed that if the keyboard is used at the end, keyloggers can capture previously entered OSK data, indicating that ending with keyboard input is insecure. Finally, when the OSK was used for the final input after the keyboard, it successfully prevented the sniffing of OSK-entered portions of the password. These results are summarized in Table 5.

Table 5 Password input methods and keystroke logging results

Password Input Method	**Keystroke Logging**
Keyboard Only (“**Password2024”**)	“Password2024”
OSK Only (“**Password2024”**)	Not Detected
OSK (“**Password”**)+Keyboard (“**2024”**)	“Password2024”
Keyboard (“**Password”**)+OSK (“**2024”**)	“Password”
OSK (“**Pass”**)+Keyboard (“**word”**)+OSK (“**2024”**)	“Password”

These findings highlighted that OSK was effective only when used exclusively, without mixing input from a physical keyboard. Further analysis of Bettercap’s keylogger scripts revealed three primary data capture mechanisms: *Capture Method 1 (Keystroke Logging)*, which intercepts individual characters as they are typed; *Capture Method 2 (Keystroke Triggers)*, which uses keypress events to extract values from text fields; and *Capture Method 3 (Submit Button Triggers)*, which captures the full contents of input fields when a form is submitted. The OSK successfully mitigated *Capture Methods 1 and 2* by eliminating keystroke events. However, *Capture Method 3* remained a vulnerability, as submitting a form using a standard <input type="submit"> element can still trigger keylogger data extraction. To reduce this risk, developers could replace the default submit input with a <button> element and a secure JavaScript onclick handler (e.g., onclick='submitForm()'), which may limit keyloggers’ ability to intercept data. Nonetheless, adaptive keylogger scripts could still target this method. In summary, while the OSK significantly reduced the risk of input interception, it was not a comprehensive solution to the problem. Its effectiveness is maximized when combined with dynamic authentication mechanisms (e.g., TSHP) and robust protection against SSL stripping

6.3 Usability and Compatibility

To evaluate the real-world applicability, the OSK prototype was tested across major browsers (Chrome, Firefox, Safari, and Edge) and devices (Windows, macOS, Android, and iOS). The component functioned consistently across platforms, requiring no plugins or browser extensions, and utilized only standard HTML and JavaScript APIs. A lightweight JavaScript module integrates the OSK directly into existing login forms, supporting both desktop and mobile interfaces while remaining compatible with Content Security Policies (CSP). Integration is straightforward, requiring minimal developer effort. A small-scale usability test with 30 participants showed that 90% found the OSK intuitive and easy to use, and 80% appreciated layout randomization for added security, while login time increased by only 4-6 seconds.

7. Recommendation, Discussion, Limitations, and Future Work

7.1 Recommendation for Different Stakeholders

We provide stakeholder-specific recommendations to support actionable improvements, categorized into short-term and long-term measures. For web developers, the short-term recommendations are: (1) implementing HSTS with preload across all production domains, (2) deploying the TSHP mechanism and OSK for user login interfaces, and (3) hardening form submission processes to reduce client-side attack vectors. The long-term recommendations are: (1) transitioning toward passwordless authentication systems (e.g., FIDO2, passkeys), and (2) integrating behavioral anomaly detection to detect phishing or homograph attacks. For policymakers and regulators, our short-term recommendations are: (1) issuing national guidelines for HSTS configuration and basic MITM defenses, (2) promoting awareness campaigns about domain spoofing and HTTPS warnings. Also, the long-term recommendations are: (1) motivating adoption of modern passwordless authentication standards through policy or funding, (2) establishing mandatory cybersecurity baselines for financial and educational institutions.

7.2 Performance Trade-offs of Our Solutions

The primary security benefit of TSHP was its resistance to brute-force attacks, with cracking times for complex passwords exceeding one year (Table 4, Section 5.5). This robustness came at the cost of minor computational overhead for hash generation and validation, which is justified for high-security applications like financial services. Similarly, the OSK’s protection against keyloggers addressed a critical vulnerability in Section 4.3. However, its usability impact may deter adoption in scenarios prioritizing speed over security, such as casual e-commerce platforms. Organizations must balance these trade-offs based on their risk profiles. For instance, Thai banks facing frequent phishing and

MITM attacks may prioritize TSHP and OSK despite minor performance costs. At the same time, educational platforms with lower risk may opt for simpler defenses to preserve user experience.

7.3 Practical Implementation Challenges of Solutions

Implementing TSHP and OSK in real-world systems present some challenges. First, TSHP requires precise time synchronization between clients and servers to ensure TOTP accuracy. Misconfigurations, common in distributed systems, could lead to login failures, particularly in regions with unreliable network connectivity, such as rural Thailand. Second, integrating OSK into existing login forms requires developer effort to adapt front-end interfaces, especially for legacy systems lacking modern JavaScript support. This could increase deployment costs for smaller organizations, such as local educational institutions.

Additionally, user adoption posed a challenge. While enhancing security, the OSK's randomized layout may confuse less tech-savvy users, a significant concern given Thailand's diverse digital literacy levels. Training and user education campaigns were essential to mitigate resistance, particularly for older demographics accessing financial services.

7.4 Comparing TSHP with Traditional Password-based and Modern Passwordless Approaches

Table 6 presents a feature-based comparison of our proposed TSHP mechanism against traditional password-based and modern passwordless authentication approaches. Our TSHP approach demonstrated significant improvements over conventional systems. It employed time-based dynamic salting using TOTP, which provided robust replay protection. The optional integration of an OSK added partial resistance to keylogger attacks, enhancing security at the input layer. Particularly, TSHP supported client-side hashing, reducing exposure of plaintext passwords during transmission, and incurred low infrastructure cost, making it suitable for integration with existing systems. By contrast, the traditional fixed salt method lacks dynamic salting and replay protection and offers no defense against keyloggers. Although it supported client-side hashing and remains inexpensive to deploy, its static nature leaves it vulnerable to rainbow-table attacks. The TOTP + Password model, while offering dynamic salting and replay protection, does not incorporate client-side hashing or defenses against keyloggers.

Modern passwordless approaches such as WebAuthn (Hodges et al., 2021), FIDO2 (Kuchhal et al., 2023) and Passkeys offer strong protection against keyloggers, replay attacks, and eliminate the need for passwords. These approaches relied on asymmetric cryptography and Public Key Infrastructure (PKI) or secure hardware elements (e.g., YubiKeys, platform authenticators). However, they come with a high infrastructure cost (as shown in Table 6), requiring hardware tokens, secure enclave integration, or cross-device credential management.

While these passwordless authentication approaches are gaining momentum globally, the adoption within Thailand remains limited due to legacy infrastructure, inconsistent device support, and low user readiness. Most financial, commercial, and educational platforms examined in this study still rely on traditional password-based systems. As a result, our proposed TSHP mechanism served as a practical transitional solution. By incorporating time-based dynamic salting and optional secure input via OSK, it mitigates threats such as SSL stripping, password sniffing, and keylogging without requiring significant infrastructure changes. This makes it especially suitable for resource-constrained environments aiming to enhance security without complete migration to passwordless systems.

Table 6 Feature-based functional comparison of TSHP, traditional, and modern authentication mechanisms

Method	**Dynamic Salt**	**Replay Protection**	**Keylogger Resistant**	**Passwordless**	**Client-side Hashing**	**Infrastructure Cost**
TSHP (Proposed)	Yes (TOTP)	Yes	Partial (with OSK)	No	Yes	Low
Traditional Fixed Salt	No	No	No	No	Yes	Low
TOTP + Password (no hashing)	Yes	Yes	No	No	No	Low
WebAuthn/FIDO2	N/A (PKI)	Yes	Yes	Yes	No	High
Passkeys	N/A (Device sync)	Yes	Yes	Yes	No	High

7.5 Other Limitations and Future Work

The 27 selected Thai websites provided valuable test samples as they aligned with the targets specified by the Technology and Cyber Crime Division of DSI. However, the dataset size was limited, and testing on a broader range of sectors could provide deeper insights. This could be a future research direction.

This study was confined to web-based login pages and simulated MITM scenarios. Advanced threats were outside its scope, such as screen capture malware and clipboard sniffers. Further investigation could focus on developing robust client-side security measures, integrating machine learning for detecting homograph redirection attacks, and enhancing browser security features to counter TLD manipulation.

Moreover, this research concentrated exclusively on input-level threats, including SSL stripping, homograph redirection, and JavaScript-based keylogger injection. Other SSL/TLS-related vulnerabilities were not addressed, including certificate spoofing, TLS version downgrade attacks, and manipulation of certificate authority (CA) trust. These attack vectors required distinct testing methodologies, tools, and assumptions, making them promising areas for future exploration.

8. Conclusions

This study examined the security posture of 27 critical Thai websites across the financial, commercial, and educational sectors, focusing on SSL stripping, homograph redirection attacks, and keylogger injections after MITM. The experimental results demonstrated that 96.3% (26/27) of the examined websites were vulnerable to SSL stripping attacks, primarily due to improper HSTS implementation. Only one website implemented HSTS Preload correctly, yet it remained susceptible to homograph redirection attacks through TLD manipulation. Furthermore, all examined websites, including those implementing password hashing, proved vulnerable to keylogger injection following successful MITM attacks. These vulnerabilities stemmed largely from inadequate HSTS implementation, reliance on traditional password-based authentication, and a lack of defenses against client-side threats. These findings also reflected a critical gap in security awareness and implementation across Thai digital platforms. Although HTTPS and hash-based passwords are widely adopted, critical safeguards, such as HSTS preloading and input-layer defenses, are neglected. To address these gaps, Thai government agencies and financial institutions should adopt standardized hardening practices and conduct regular penetration testing to mitigate such risks.

To address the vulnerabilities, we also proposed and evaluated two practical, low-cost countermeasures: (1) a Time-based Salted Hash Password (TSHP) mechanism using dynamic Time-based OTPs as salting material, and (2) an On-Screen Keyboard (OSK) to resist JavaScript-based keyloggers. These methods enhance existing systems without requiring significant infrastructure changes.

Although our study focused on Thai digital infrastructure, the vulnerabilities identified, such as poor HSTS configuration, are prevalent in many countries. Therefore, the proposed TSHP and OSK mechanisms offer a globally applicable framework for strengthening legacy web authentication systems. While modern standards like WebAuthn, FIDO2, and passkeys provide more robust protection through public-key cryptography and hardware-based authenticators, their widespread adoption remains limited in regions with infrastructure or budget constraints. Our solutions served as a transitional step toward adopting these standards.

9. Acknowledgements

This research was supported and funded by Thailand Science Research and Innovation (TSRI), the Department of Special Investigation (DSI) of Thailand, and Mahasarakham University. We are grateful to Professor Karim Djemame and other members of the Distributed Systems and Services Research Group at the University of Leeds, UK, for their valuable feedback on the manuscript and their collaboration on our Newton Grant project.

10. CRediT Statement.

Khathawut Chanbuala: Conceptualization, methodology, software, validation, writing – original draft, visualization.
Darunee Puangpronpitag: Writing, review & editing, supervision, project administration.
Egachai Puangpronpitag: Resources, investigation, data curation (defined sample group of 13 e-banking platforms, 2 e-commerce sites, and 12 university registration systems used in the assessment).
Somnuk Puangpronpitag: Conceptualization, methodology, writing – review & editing, supervision, funding acquisition.

11. References

Bangkok Bank. (2023). *Customers are advised to be cautious of fake websites impersonating Bangkok Bank's official site*. Retrieved from https://www.bangkokbank.com/en/Personal/Tips-and-Insights/Fake-Web

Bettercap. (2024). *bettercap (Version 2.33.0): The Swiss Army knife for 802.11, BLE, IPv4 and IPv6 network reconnaissance and MITM attacks* [Computer software]. https://www.bettercap.org/

Buffermet. (2024, September 9). *HSTS hijacking caplets* [Code]. GitHub Gist. https://github.com/bettercap/caplets/blob/master/hstshijack/hstshijack.cap

Chanbuala, K., Puangpronpitag, E., Puangpronpitag, D., & Puangpronpitag, S. (2024). Evaluating and mitigating HTTPS interception in Thai E-banking websites: Challenges and solutions [Conference presentation]. *Proceedings of the 2024 13th International Conference on Networks, Communication and Computing*, Bangkok, Thailand. https://doi.org/10.1145/3711650.3711662

Croley, S. (2022). *RTX_4090_v6.2.6.Benchmark* [Benchmark results]. GitHub Gist. Retrieved from https://gist.github.com/Chick3nman/32e662a5bb63bc4f51b847bb422222fd

Hodges, J., Jackson, C., & Barth, A. (2012). *HTTP strict transport security (HSTS) (RFC 6797)*. Internet Engineering Task Force. https://doi.org/10.17487/RFC6797

Hodges, J., Jones, J. C., Jones, M. B., Kumar, A., & Lundberg, E. (2021, April 8). *Web authentication: An API for accessing public key credentials—Level 2*. World Wide Web Consortium (W3C). Retrieved from https://www.w3.org/TR/webauthn-2/

Jirawankul, K. (2015). *Phishing analysis of kasikorn.ru* [Blog post]. FOH9. Retrieved from https://foh9.blogspot.com/2015/03/kasikornru.html

Khachenrum, P., Puangpronpitag, D., Puangpronpitag, S., & Puangpronpitag, E. (2023). Problem analysis of HSTS malfunction and SSL stripping attack. *The Journal of King Mongkut's University of Technology North Bangkok*, *33*(2), 626–636. https://doi.org/10.14416/j.kmutnb.2021.07.007

Kuchhal, D., Saad, M., Oest, A., & Li, F. (2023). Evaluating the security posture of real-world fido2 deployments [Conference presentation]. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, Copenhagen, Denmark. Retrieved from https://doi.org/10.1145/3576915.3623063

M'Raihi, D., Machani, S., Pei, M., & Rydell, J. (2011). *TOTP: Time-based one-time password algorithm (RFC 6238)*. Internet Engineering Task Force. https://doi.org/10.17487/RFC6238

Marlinspike, M. (2009). *New tricks for defeating SSL in practice*. Black Hat DC 2009, Washington, DC, US. Retrieved from https://blackhat.com/presentations/bh-europe-09/Marlinspike/blackhat-europe-2009-marlinspike-sslstrip-slides.pdf

Offensive Security. (2024). *Kali Linux 2024.2 release (t64, GNOME 46 & community packages)* [Release notes]. Retrieved from https://www.kali.org

Puangpronpitag, E., & Puangpronpitag, S. (2022). *Development of guidelines for cyber-crime investigation towards electronic banking services and analysis of information technology techniques to build a security-enhanced prototype* [Research report]. Thailand Science Research and Innovation. Retrieved from https://jkb.oja.go.th/home/view/10762

Rescorla, E. (2000). *HTTP over TLS (RFC 2818)*. Internet Engineering Task Force. Retrieved from https://doi.org/10.17487/RFC2818

Wireshark Foundation. (2023). *Wireshark 4.4.0 release notes*. Retrieved from https://www.wireshark.org/docs/relnotes/wireshark-4.4.0.html