# Student Declaration

I, Jayaraj V Patil hereby declare that the project report entitled "Analyzing Security Vulnerabilities in Mobile Browsers" submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Applications (BCA) is a record of my original work carried out under the guidance of the faculty at Seshadripuram Arts, Science and Commerce College, Bangalore. This report has not been submitted to any other university or institution for the award of any degree, diploma, or certificate. I have acknowledged all sources used and have not committed any form of plagiarism. The work is based on my own understanding, research, and implementation efforts.

# Acknowledgement

I would like to express my sincere gratitude to all those who contributed to the successful completion of this research project titled “Analyzing Security Vulnerabilities in Mobile Browsers.”

First and foremost, I am deeply thankful to my project guide and faculty members at Seshadripuram College, whose valuable guidance, constant support, and insightful feedback played a crucial role in shaping this project. Their encouragement and suggestions helped me stay focused and motivated throughout the research process.

I would also like to acknowledge the online communities, research forums, and platforms such as PhishTank, Google Scholar, YouTube, and MDN Web Docs, which provided access to essential learning resources and practical information that significantly contributed to the technical development of this project.

A special thanks to my family for their unwavering encouragement and moral support, which helped me remain dedicated and confident throughout the journey.

Lastly, I am grateful to all the developers and researchers whose work in the field of cybersecurity and browser security inspired me to take on this topic. Without their foundational contributions, this research would not have been possible.

# Table of Content

# I. ABSTRACT

As smartphone proliferation spread rapidly and more and more consumers began to bank on their mobile internet connectivity, mobile browsers became indispensable daily tools for digital communication, online transactions, and access to information. At the same time, this growth in demand has also turned mobile browsers into a valuable target for cyberattacks. In recent years, desktop browsers have been enjoying greater computing resources and user attention compared to mobile browsers.

This research is centred around in-depth examination of the vulnerabilities present in popular mobile browsers like Google Chrome, Brave, Firefox, and Edge. Using systematic penetration testing, real-world case studies, and behavioural analysis, I uncovered a series of critical security weaknesses that compromise millions of users. Among the key dangers revealed are phishing attacks based on inadequate URL visibility, clickjacking attacks involving user interaction manipulation, address bar hiding through JavaScript injection, and browser fingerprinting methods employed for tracking users against their will.

The research also highlights the extent to which mobile users unwittingly assist in their own insecurity by virtue of ignorance and unsafe browsing practices. The research process involves hands-on testing of browser reactions to attack simulations, examination of JavaScript vulnerabilities, and assessment of browser permission and privacy management. The results of these tests not only indicate technical vulnerabilities in browser design but also point to the necessity for improved user-side security awareness.

In response to these results, the paper introduces a multi-layered security-improving solution for both developers and users. This involves implementation of more rigid sandboxing policies, real-time URL validation processes, fingerprinting prevention methods, and improved browser-level warnings for potentially malicious activity. The introduced solution is engineered to evolve to new threats without compromising performance and usability on mobile devices.

Finally, this study adds to the expanding field of cybersecurity through illumination of the otherwise understudied threats in mobile browsing. It calls for proactive mobile browser

development with better security and calls for responsible browsing practices from users with the long-term aim of creating a more secure mobile web.

# II. Introduction

In the age of connectivity, the sudden upsurge in smartphone and mobile application usage has revolutionized internet access. At the epicentre of this revolution are mobile browsers, which act as the major interface for accessing the web on smartphones. Ranging from making financial transactions to connecting with social networks and cloud services, users put mobile browsers in charge of tremendous volumes of personal and sensitive data. Yet, this dependence has also turned mobile browsers into a steadily more profitable prospect for cybercriminals. Unlike the desktop environment, mobile environments pose special security challenges because they have constrained system resources, diverse operating systems, unreliable network connections, and more relaxed user behaviour, such as rapid closing of security alerts.

Mobile browsers can be exposed to any number of attacks, from phishing and clickjacking to JavaScript-based address bar spoofing and browser fingerprinting. Phishing scams trick users into handing over confidential details by impersonating authentic sites. Clickjacking overlays undetectable components over webpages, leading users to click on infected components. Attacks based on JavaScript can influence the behaviour of the browser, such as hiding the address bar in order to hide the URL of an offending website. Browser fingerprinting, though less noticeable, is a popular way for stalking users quietly through gathering unique browser and device information. These attack vectors reveal not just technical vulnerabilities in browser design but also the shortcomings of user knowledge and the efficacy of inherent security features.

Even though contemporary mobile browsers are furnished with a variety of defensive technologies—like sandboxing, validation of SSL certificates, incognito and private browsing modes, and anti-phishing mechanisms—attackers keep finding ways to outsmart these defines measures. On top of this, privacy functionality that is highly marketed by big browsers tends to fail in the real world. For instance, forensic examination of incognito or private browsing activities has shown that traces of data could still be extracted using sophisticated tools, thus breaching users' privacy despite their perception that they are safely browsing.

This research paper seeks to extensively analyse the security environment of mobile browsers using real-world attack demonstrations and extensive penetration testing. The research targets leading mobile browsers like Google Chrome, Mozilla Firefox, Brave, and Microsoft Edge, evaluating their robustness against phishing, clickjacking, JavaScript injection, and fingerprinting. The performance of each browser is gauged based on its detection rates, user alert mechanisms, and capability to block unauthorized data collection or manipulation.

In order to achieve this, the research employs a formalized approach comprising the creation of realistic attack scenarios, exploitation simulation in a contained virtual Android lab, and results analysis from the responses of the browsers. Penetration testing indicates a number of alarming findings—such as, all browsers tested did not succeed in preventing clickjacking attempts, and most were not consistent in the detection of phishing URLs, especially when the websites utilized HTTPS. Likewise, browser fingerprinting was still very effective in browsers such as Chrome and Edge, while Brave was comparatively more resistant.

In addition to exposing vulnerabilities, this research suggests a future-proof solution that combines artificial intelligence (AI) and privacy-enhancing mechanisms to enhance mobile browser security. Recommendations include per-tab VPN connections, randomized device identifiers, AI-driven firewalls, and pre-emptive web scanning to fix the weaknesses uncovered by the tests. These solutions are designed to improve user privacy, prevent tracking attempts, and offer proactive defines against emerging web-based threats.

Ultimately, this study not only adds to the technical knowledge of mobile browser vulnerabilities but also serves to highlight the imperative for stronger, user-oriented, and responsive security measures in the mobile browsing ecosystem. As mobile phones remain the most used devices worldwide for internet access, guaranteeing the security of mobile browsers is key to safeguarding user privacy and upholding trust in online services.

# III. Literature Survey

## a) Introduction to Mobile Browser Security

As mobile phones increasingly reign supreme when it comes to accessing the internet globally, mobile browsers have emerged as important gateways to the web. Mobile browsers enable all sorts of user interactions ranging from light browsing to secure online banking, which makes them an attractive target for cyber attackers. But mobile platforms come with unique security issues that do not exist in desktop environments, such as constrained system resources, reduced user interfaces, heterogeneous permission models, and constant network switching.

The security mechanisms built into mobile browsers are usually based on desktop equivalents but designed for mobile use. This balance between functionality and convenience tends to leave an opening for exploitable vulnerabilities. Seeing the need for strong protection mechanisms, a growing body of work has examined the behaviour, architecture, and vulnerabilities of mobile browsers under different attack scenarios.

## b) Mobile Browser Forensics and Data Privacy Concerns

Forensic examination of mobile browsers has emerged as a critical research area in determining data persistence and the boundaries of private browsing modes. Chanda et al. (2024) performed a comprehensive study of browser forensic behaviour on popular browsers like Google Chrome, Mozilla Firefox, Brave, and Microsoft Edge. They reported that browser artifacts, including browsing history, cache information, cookies, autofill data, and session tokens, could be restored even after sessions were closed or data was "cleared" by users.

These findings are particularly ominous when considered in the context of incognito and private browsing modes that are commonly thought to preclude data storage. The forensic software Autopsy and Magnet AXIOM, contrary to user expectations, were found to recover residual artifacts from volatile memory and application cache directories, which attests to an evident discrepancy between user-facing privacy claims and implementation.

## c) Limitations of Private Browsing and Residual Data Risks

Drawing on these issues, Alotibi et al. (2024) assessed the effectiveness of privacy modes in mobile and desktop browsers. They discovered that mobile devices—particularly rooted or jailbroken devices—were far more susceptible to forensic recovery of data. With software like XRY and FTK Imager, their research recovered login credentials, browsing history, and form autofill information even after a private session had closed.

Their work indicated that private browsing modes effectively block data from being saved in traditional places such as history logs but do not offer protection against memory dumps, swap file analysis, and system-level data extraction. In addition, most browsers employed in mobile environments did not encrypt sensitive data at rest, further exposing the data to increased risk.

These works as a whole demonstrate that there is an immediate problem: mobile browsers are not forensically secure, particularly if they're used on rooted or compromised devices. Even with private modes available, user information can be revealed under certain circumstances, suggesting the need for more effective privacy enforcement and secure memory management mechanisms.

## d) Threats from JavaScript-Based Attacks

Building upon these issues, Alotibi et al. (2024) tested the effectiveness of privacy modes on mobile and desktop browsers. They observed that mobile devices—particularly rooted or jailbroken devices—were much more susceptible to forensic data recovery. Through the use of tools like XRY and FTK Imager, their experiment recovered login details, browsing history, and form autofill data even after the private session closed.

Their work indicated that Pri JavaScript is a fundamental enabler of dynamic web content but also one of the most targeted attack vectors. Various studies have analysed how attackers employ JavaScript to perform Cross-Site Scripting (XSS), alter page objects, and even fake security signals such as the address bar.

DOM-based XSS vulnerabilities in mobile browsers have been investigated in a study conducted by Lekshmi et al. (2023). They conducted experiments where they revealed that most mobile browsers ran malicious JavaScript payloads placed inside URLs or HTML without adequate sanitization and validation. Through this, the attackers could steal sessions, tamper with page content, and get access to client-side sensitive data.

Further, research into address bar spoofing using JavaScript has indicated that the attacker can hide or modify the address bar through scrolling attacks, overlays, or DOM manipulation. These spoofing attacks are especially risky in the case of mobiles because there are smaller screen sizes and more visual reliance on indicators such as padlocks or logos.

These results substantiate the contention that mobile browsers have limited defences against sophisticated JavaScript-based exploits, and most depend on weak or obsolete script filtering mechanisms. Most of these browsing modes effectively keep data from being written to traditional places such as history logs but are not effective against memory dumps, swap file examination, and system-level data retrieval. In addition, most browsers utilized in mobile contexts did not encrypt sensitive data at rest, which further heightened the risk of data exposure.

All these studies collectively refer to a crucial problem: mobile browsers are insecure against forensic examination, particularly if used on rooted or compromised devices. In spite of private modes being present, user data might still be visible under specific conditions, showing that there is a need for more effective privacy enforcement and secure memory management practices.

## e) Clickjacking and User Interface Deception

Clickjacking is another attack channel extensively documented in cybersecurity research. It entails covering malicious or blank HTML elements on top of real web content such that users get tricked into taking actions that they did not intend to do. For instance, users will be tricked into clicking "Allow" from a permission pop-up, perceiving it to be part of a quiz or game.

A study conducted by Hansen and Grossman (2022) demonstrated how the absence or poor implementation of Content-Security-Policy and X-Frame-Options headers opened the doors for clickjacking attacks on different browsers. Though these headers have been suggested to counter such attacks, their implementation on mobile browsers is not uniform.

Additional tests by Narayan et al. (2023) showed that mobile users are more vulnerable to clickjacking because of the reduced UI, bigger touch targets, and absence of hover interaction. The absence of visual feedback in mobile browsers enhances the susceptibility to deception-based attacks, a situation that present browser defences have not been effective in addressing properly.

## f) Browser Fingerprinting and Privacy Evasion

Browser fingerprinting has also become a formidable tracking method that evades common defences such as cookie clearing and incognito browsing. The method entails gathering an individual set of browser and device attributes, including screen resolution, time zone, language, fonts installed, and hardware capabilities, to create a persistent user identity.

Eckersley (2010) initiated extensive studies on fingerprinting using the Panopticlick project, demonstrating that a vast majority of browsers could be distinguished uniquely without the use of cookies. Recent work, including the study by Nikiforakis et al. (2019), indicated that even privacy browsers tend to leak faint fingerprinting signals, like WebGL, canvas rendering, and audio contexts.

Fingerprinting works best on mobile, where settings are fewer and more straightforward than on computers, allowing patterns to be tracked more easily. Privacy software such as Brave attempted to counter this by randomizing fingerprint values and restricting API exposure, but research indicates that fingerprinting is a significant threat and one of which users are generally unaware.

## g) Existing Browser Security Mechanisms and Their Shortcomings

Current mobile browsers provide a number of built-in security features, such as:

- Sandboxing: Limiting each tab to its own process.
- HTTPS enforcement: Displaying encrypted connections with a padlock.
- Phishing protection: Blocking known malicious sites using blacklists or heuristic engines.
- Private browsing modes: Disallowing browsing history and cookies from being stored.

These, however, are not enough, researchers claim, as these protections are superficial and usually ineffective against evolving threats. For instance: HTTPS cannot confirm the authenticity of content, but only encryption. Sandboxing does not stop script-based attacks or fingerprinting. Private browsing is defeated by forensic recovery tools. Phishing detection only works on-loaded pages, not pre-interaction. These shortcomings have prompted researchers to suggest next-gen defences like AI-powered web scanning, behaviour-based webpage analysis, and decentralized identity verification systems.

## h) Summary of Literature Findings

Literature surveyed presents a complete image of the mobile browser security environment today:

- Forensic analysis has found critical weaknesses in private browsing mechanisms and data storage practices.
- Security research has proven persistent susceptibility to phishing, clickjacking, and JavaScript attacks.
- Privacy research indicates that fingerprinting can outsmart even the most privacy-respecting browsers
- Current tools and policies are too frequently not applied rigorously enough to be truly protective.

These observations highlight the imperative for browser vendors and security researchers to create more forward-looking, intelligent, and responsive solutions for defending users against more advanced threats.

# IV. System Analysis

## a) Existing System Overview

Mobile browsers are an integral part of contemporary smartphones, allowing users to use web-based applications, conduct financial transactions, communicate through social media, and search information on the move. Well-known mobile browsers like Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave offer a clean interface optimized for minimal screen space and battery life.

These browsers often include a set of inbuilt security features, such as:

- Incognito or Private Browsing Modes: Avoid history, cookie, and form data storage.
- Sandboxing: Every tab is executed in an independent process, separating it from others to minimize attack effect.
- HTTPS Enforcement: Encrypted communication is favoured and visually signalled to the user.
- Security Headers: Browsers implement X-Frame-Options, Content-Security-Policy (CSP), and other headers that websites can utilize to avoid clickjacking and script injections.
- Phishing and Malware Protection: Browsers use blacklists or third-party APIs (such as Google Safe Browsing) to inform users of potentially hazardous sites.
- Certificate Validation: SSL certificates are validated to ensure HTTPS sites are genuine.

Although these capabilities are designed to make systems more secure, they tend to offer only protection at the surface level, particularly when users inadvertently engage with hazardous content that takes advantage of design-level or behavioural vulnerabilities in mobile browsers.

## b) Limitations of the Existing System

Despite the implementation of various security mechanisms, existing mobile browsers are vulnerable to a number of sophisticated and well-documented attacks. Based on referenced literature and external studies, the following limitations have been identified:

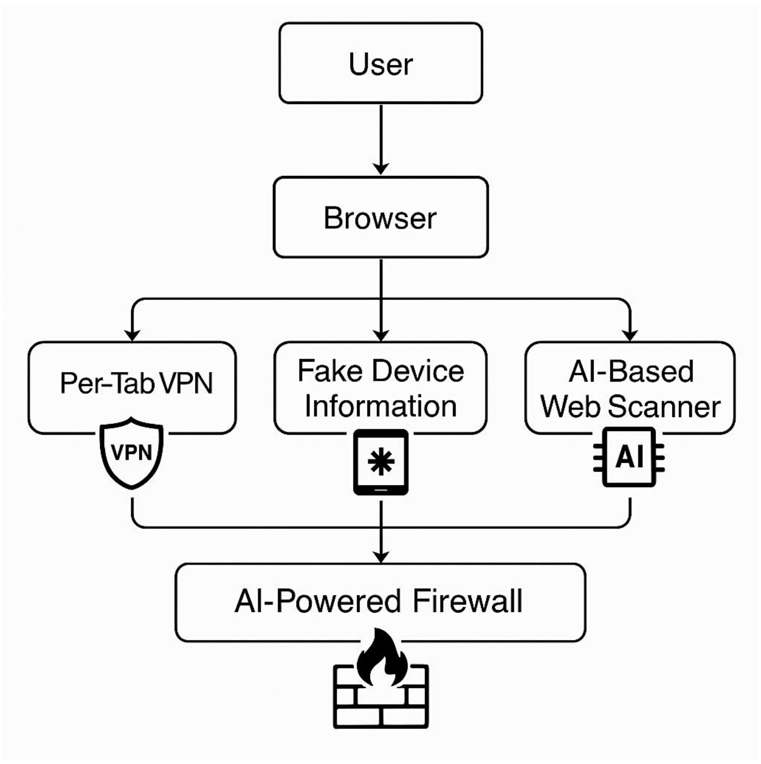
- Phishing Detection Shortcomings: Most browsers fail to block phishing sites that use HTTPS. They rely on URL-based blacklists, which are reactive and cannot detect newly generated malicious domains or content spoofing.
- Clickjacking Vulnerability: Although headers like X-Frame-Options exist, their enforcement varies widely across browsers. Research shows that many mobile browsers fail to prevent transparent overlays, which can trick users into clicking hidden buttons.
- JavaScript Injection & XSS: Malicious JavaScript can be injected into webpages to steal cookies, modify content, or redirect users. DOM-based XSS attacks still function across various browsers, especially when no server-side validation is in place.
- Private Browsing Limitations: Studies reveal that forensic tools such as Autopsy, Magnet AXIOM, and XRY can recover session artifacts, URLs, and login details even after private browsing sessions are closed. This challenges the assumption that private modes guarantee data confidentiality.
- Browser Fingerprinting: Fingerprinting scripts can extract a combination of user-agent strings, screen resolution, font lists, and plugin information to create a persistent identifier—even when cookies are blocked or deleted. Mainstream browsers like Chrome and Edge are particularly vulnerable to this form of tracking.
- Lack of Pre-emptive Threat Analysis: Current browsers load content before analysing it for phishing or malware behaviour. This allows attackers to exploit even brief interactions, such as a single tap on a disguised link.

These limitations demonstrate that existing browser security architecture is insufficient to defend against evolving cyber threats in real time.

## c) Proposed System Overview

In order to overcome the limitations of current mobile browsers, this work suggests a security-enhanced system that incorporates next-generation defines techniques at the browser level. The system adds the following new components:

- Per-Tab VPN Tunnelling: Every browser tab runs with its own independent VPN tunnel, essentially isolating network traffic and stopping cross-tab tracking by websites or network-level attackers.
- Mock Device Information (Virtual Tabs): The system produces random device fingerprints per tab or session, such as simulated user-agent strings, screen size, and plugin lists, thus rendering fingerprint-based tracking methods ineffective.
- AI-Driven Firewall: An AI-driven, lightweight firewall is integrated into the browser space. It inspects live network packets for anomalies, recognized threat patterns, and aberrant behaviour signatures (e.g., data egress).
- AI-Powered Web Scanner: Even before loading a webpage, the integrated AI engine inspects it for indicators of phishing, bad JavaScript, cloak redirects, and layout modification—thus pre-empts attacks from ever occurring as the result of user engagement.



The browser is hence reborn into a smart, defines-capable platform that proactively detects and destroys threats in real-time.

## d) Objectives of the Proposed System

The system under proposal is built with the following objectives:

- Enforce Stronger Privacy Protections: Reduce digital fingerprinting and discontinue cross-tab tracking by deploying randomized device identity and per-tab VPN isolation.
- Improve Threat Detection: Identify phishing and malware threats prior to webpage rendering using AI-powered heuristics and behaviour analysis.
- Combats Script-Based Attacks: Block or sanitize malicious JavaScript payloads from DOM manipulation and sensitive data exfiltration.
- Defends Users Against Clickjacking: Detect UI overlays or hidden elements in real time, alerting users before accidental interactions take place.
- Resist Forensic Recovery: Protect temporary data and keep it properly clean after a session to augment privacy in both regular and incognito browsing modes.
- Preserve Usability: Accomplish all security goals without compromising performance or user experience, particularly on mid-range mobile devices.

## e) Feasibility Study

A feasibility study assesses if the envisioned system can actually be implemented and used.

**a. Technical Feasibility**

- Development is possible with open-source software such as Virtual Android environments, Python-based network analysis modules, JavaScript fingerprinting libraries, and ML threat detection models.
- Current browser engines like Chromium can be forked and adapted to include the new security features.
- Cloud-based AI models can be deployed first, with a plan for light on-device inference to follow.

**b. Economic Feasibility**

- Free or open-source tools dominate the requirements.
- No substantial infrastructure expenses beyond test hardware and virtualized environments are required.

- The solution is scalable, and prototype implementation does not involve commercial licensing.

**c. Operational Feasibility**

- The system can operate in controlled environments (e.g., rooted emulators or hacked Android builds).
- Although not yet production-grade for common users, the idea can be polished into a deployable secure browser in the future.
- Integration with AI and VPN infrastructure is pragmatically viable for users with medium device specifications.

## f) Conclusion of Analysis

The analysis of the system identifies a wide disparity between the mobile browser security capabilities of today and the nature of attacks that users encounter in the present. By defining the limitations of current browsers and describing a security-enhanced system that is built with proactive intelligence, awareness of privacy, and real-time threat detection, this research sets the stage for the next generation of secure mobile browsing.

# V. System Specifications

This section offers an in-depth overview of the technical setup employed to perform penetration testing on mobile browsers. The setup consists of both hardware and software details, along with the tools, libraries, operating systems, and browser versions employed in the research. Testing was performed entirely within a safe, isolated virtual lab to provide ethical measures and system segregation.

The objective of system configuration definition is to allow experiments and provide transparency into the setup of the analysis. In order to facilitate accuracy and verification of research results, since individuals' browser conduct can be dependent on device settings, operating system, and network state and conditions, comprehensive details of the specifications are necessary.

## a) Hardware Specifications

A moderately powerful system was used for the implementation and execution of attack scenarios. Since virtualization and simultaneous browser sessions were required, higher memory and CPU capacity ensured smooth performance.

| Component | Specification |
|---|---|
| Processor | AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx |
| RAM | Minimum 8 GB |
| Storage | 256 GB SSD |
| Display | Full HD (1920x1080) resolution, to test address bar behaviour and UI rendering |
| Network Adapter | Wi-Fi connectivity for packet-based and phishing attack testing |
| Virtualization | Hardware virtualization |

The chosen specifications ensured smooth operation of resource-intensive tools like Burp Suite and Wireshark, alongside VM environments and real-time browser testing.

### b) Software Specifications

Multiple open-source tools and platforms were deployed to create an environment suitable for penetration testing and browser vulnerability assessment. The software stack was chosen based on compatibility, stability, community support, and availability of required security utilities.

| Software / Tool | Purpose / Role in Research |
|---|---|
| VMware Workstation | Virtualization tool for running Kali Linux as guest OS |
| Kali Linux (2024 Rolling) | Primary penetration testing OS with preinstalled security tools |
| Bliss OS / Android Emulator | Used for simulating Android devices and browser behaviour |
| Python 3.x | Scripting of automation tasks and custom logic |
| JavaScript / HTML / CSS | Used for creating test web pages (phishing, XSS, spoofing) |
| Wireshark | Capturing and analysing network traffic in real time |
| PhishTank Dataset | Dataset of real-world phishing URLs used for testing |

Kali Linux was chosen for its extensive support for cybersecurity testing tools, while VMware ensured flexibility and performance in creating isolated lab environments without the need for physical Android devices.

### c) Testing Environment and Configuration

The entire testing workflow was configured in a secure lab environment using the following structure:

- Host OS: Windows 11
- Virtualization Layer: VMware Workstation running Kali Linux

- Testbed OS: Android Emulator (via Bliss OS) simulating Android 11
- Browser Testing Target: Android versions of Google Chrome, Firefox, Brave, and Edge
- Network: Connected to the internet with a virtual NAT adapter to simulate real-time web interactions

Each test scenario (phishing, XSS, clickjacking, fingerprinting) was executed multiple times across different browser environments to record consistency, detection mechanisms, and response behaviour.

## d) Browsers Evaluated

The study focused on evaluating the top mobile browsers by usage and feature set. The selected browsers were tested on the Android platform using the emulator environment. Specific builds and stable versions were used to maintain consistency.

| Browser | Version Tested | Platform |
|---|---|---|
| Google Chrome | Version 119.0+ | Android |
| Mozilla Firefox | Version 119.0+ | Android |
| Microsoft Edge | Version 118.0+ | Android |
| Brave Browser | Version 1.62+ | Android |

## e) Tools and Libraries Used

Several utilities, scripts, and libraries were used to simulate real-world browser threats and analyse browser behaviour.

| Technology / Tool | Purpose |
|---|---|
| HTML & CSS | Developed clickjacking simulation pages with invisible overlays and iframes |
| JavaScript | Used to perform browser fingerprinting using navigator and other DOM APIs |

| | |
|---|---|
| Python (Simple HTTP Server) | Used to test pages locally without using external frameworks |
| PhishTank Dataset | Provided real-world phishing URLs for browser detection testing |

## f) Ethical Testing Practices

All security testing activities were carried out in an isolated lab environment with no connection to real-world users or external systems. The testbed was completely self-contained, using simulated browser sessions and internally hosted payloads. No public websites were attacked, and all phishing URLs were loaded from an educational dataset (PhishTank) in a sandboxed virtual environment.

- No real users were targeted
- No personal or private information was collected
- All attacks were simulated with consented tools in virtual infrastructure

## g) Summary

The system setup was designed to ensure a flexible, reproducible, and ethical research process. With the combination of VMware, Kali Linux, and Android emulation through Bliss OS, the environment enabled controlled testing of browser vulnerabilities without any risk to external infrastructure. The tools selected represent industry-standard penetration testing suites, ensuring credibility and relevance of the research results.

# VI. System Study

## a) Introduction

As mobile internet access continues to surpass desktop usage, mobile browsers have become the primary platform for web interaction. Users routinely conduct banking, shopping, and communications through browsers on smartphones. While this convenience is transformative, it also brings new and evolving security challenges.

Mobile browsers, though convenient, operate with limited screen space, inconsistent user awareness, and restricted system-level access, all of which introduce unique vulnerabilities. Traditional desktop browser defences often fail to adapt efficiently to mobile platforms. As such, the study of mobile browser vulnerabilities is essential to addressing the growing landscape of mobile cyber threats.

This system study explores the existing state of mobile browser security, highlights their limitations, and introduces the necessity for an improved, browser-focused, security-enhancing solution.

## b) Need for the Proposed System

Despite improvements in web standards and browser technology, mobile browsers still exhibit significant weaknesses:

- Phishing sites are often visually indistinguishable from legitimate ones, especially on smaller mobile screens where the full URL is rarely visible.
- Clickjacking attacks remain effective on mobile devices due to simplified UI design and lack of visual feedback (like hover states).
- JavaScript-based fingerprinting can uniquely identify and track users across websites, even in incognito mode.
- Security headers (like CSP and X-Frame-Options) are inconsistently enforced or implemented by websites, and mobile browsers often fail to warn users.

There is a need for a lightweight, flexible testing system that can simulate real-world attacks and measure how well mobile browsers respond. Additionally, there is a need to propose practical improvements to browser behaviour, particularly in the areas of preloading threat detection, privacy control, and visual protection.

## c) Existing System Overview

Most modern mobile browsers, such as Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave, include baseline security features:

- HTTPS Enforcement: Secures communication between browser and server.
- Private/Incognito Mode: Claims to not save history, cookies, or form data.
- Safe Browsing APIs: Warn users about potentially harmful or fraudulent websites.
- SameSite and HttpOnly Cookies: Prevent session hijacking and cookie theft.
- CSP and Frame Options: Allow websites to control where and how content is displayed.

However, many of these mechanisms are reactive, static, or dependent on server configuration. For instance:

- HTTPS does not indicate whether a site is legitimate, only that it is encrypted.
- Safe Browsing APIs rely on known blacklists, which are often outdated or bypassed using new phishing domains.
- Incognito mode does not protect against fingerprinting or DNS-level tracking.

Thus, although browsers provide foundational security, these features are insufficient against modern attack vectors such as phishing via HTTPS, clickjacking using transparent overlays, or silent user tracking via fingerprinting scripts.

## d) Limitations of the Existing System

Some of the fundamental limitations found in existing browser implementations are:

- Relying too heavily on SSL Certificates: Users believe that the display of a padlock icon (HTTPS) means a secure site. Attackers take advantage of this by serving phishing content on sites with legitimate SSL certificates.
- Clickjacking Vulnerability: Most browsers don't actively identify clickjacking overlays. Unless sites explicitly use X-Frame-Options, clickjacking can completely evade user awareness.

- No Active Script Behaviour Analysis: Today's browsers scarcely scrutinize the behaviour of a JavaScript file. In case obfuscated or code included in the DOM itself has evil intentions, then it largely goes unchecked.
- Fingerprinting Still Functional: Scripts will also gather unique enough properties (screen size, platform, user agent, fonts) even if it's a private browsing mode in order to construct a fingerprint browser that continues to track people for good.
- Delayed Threat Detection: Current phishing detection methods only warn after sites have already loaded. This lag provides the attacker an opportunity to do bad things or bait users.

## e) Objectives of the Proposed Research

This research aims to analyse and demonstrate real-world browser vulnerabilities using actual attack vectors, and to propose improvements for mobile browser behaviour.

The objectives include:

- To simulate real-world browser threats such as phishing, clickjacking, JavaScript injection, and fingerprinting using custom-built test pages and scripts.
- To analyse the response and defences of popular mobile browsers in a controlled environment.
- To identify gaps in browser-side security that leave users vulnerable, even with HTTPS and incognito features.
- To use only lightweight tools and technologies, such as HTML, CSS, JavaScript, Python (for scripting), and the PhishTank dataset, ensuring reproducibility and accessibility of the methodology.
- To propose enhancements that could be implemented at the browser level to improve user safety and privacy, including:
    - Preload-based phishing detection
    - Dynamic clickjacking alerts
    - JavaScript behaviour scoring
    - Fingerprinting resistance via randomization

## f) Summary of System Study

This system study highlights the critical need for a dedicated, browser-side approach to improving mobile web security. It identifies gaps in the current browser infrastructure, particularly in the areas of user tracking, content spoofing, and social engineering.

Rather than relying on server-side protection or expecting users to recognize threats, this research proposes that the *browser itself* should take a more active role in protecting the user. By combining simple, open-source tools with carefully designed test cases, this project demonstrates that browser weaknesses are still prevalent—and that practical, user-friendly solutions are both possible and necessary.

# VII. System Implementation

## a) Introduction

This section outlines the practical implementation of browser vulnerability testing, including the simulation of real-world attacks and the execution of scripted test scenarios in a controlled environment. The project used only lightweight, open-source tools such as HTML, CSS, JavaScript, and Python. Four types of browser-based attack vectors were studied: phishing detection, clickjacking, XSS injection, and browser fingerprinting.

## b) Phishing Attack Simulation Using PhishTank

Instead of developing custom phishing websites, this research utilized real phishing URLs from the PhishTank dataset, a community-based anti-phishing service that provides up-to-date phishing website links.

Procedure:

- URLs from PhishTank were accessed in each mobile browser (Chrome, Firefox, Brave, Edge).
- Browsers’ responses were observed and recorded:
    - Whether the browser blocked the URL
    - Whether a security warning was shown
    - Whether the site loaded without interruption

## c) Clickjacking Simulation (Custom HTML + CSS)

Clickjacking was simulated using a custom-built HTML page with a transparent iframe layered over a fake button.

Purpose**:** To trick users into clicking on hidden content under the impression of clicking a visible UI element.

clickjacking.html

```
<!DOCTYPE html>

<html>

<head>

    <title>Click Here to Win</title>

    <style>

        body {

            margin: 0;

            padding: 0;

            background-color: #f3f3f3;

        }

        .fake-button {

            width: 300px;

            height: 100px;

            background: #00bfff;

            color: white;

            font-size: 24px;

            line-height: 100px;
            text-align: center;

            margin: 200px auto;

            border-radius: 10px;

        }

        iframe {

            position: absolute;

            top: 200px;

            left: calc(50% - 150px);

            width: 300px;

            height: 100px;

            opacity: 0;

            z-index: 999;

        }

    </style>

</head>

<body>

    <div class="fake-button">Click Here to Win!</div>

      <iframe src="https://example.com/hidden-action" onload="alert('Iframe has
loaded!')" ></iframe>

</body>
```

```
</html>
```

Explanation:

- A visually prominent fake button is shown.
- A transparent iframe with real malicious content is placed directly above it.
- The click goes to the hidden target, not the visible button.

## d) XSS Injection (JavaScript Payload)

Cross-Site Scripting (XSS) was simulated by embedding malicious JavaScript within a local HTML page to observe how browsers handle script execution and cookie access.

xss_test.html

```
<!DOCTYPE html>
<html>
<head>
  <title>SafeBank Login</title>
  <script>
    function sanitizeInput(input) {
      var element = document.createElement('div');
      if (input) {
        element.innerText = input;
        element.textContent = input;
      }
      return element.innerHTML;
    }

    function handleSubmit(event) {
      event.preventDefault();

      var username = sanitizeInput(document.getElementById('user').value);
      var password = sanitizeInput(document.getElementById('pass').value);
```

```
            alert("Login attempt: Username - " + username);

            var cookiesStatus = document.cookie ? "Cookies are enabled and working." :
"Cookies are blocked or not working.";
            console.log("Document Cookies:", document.cookie);

            var messageDiv = document.createElement('div');
            if (document.cookie) {
              messageDiv.innerHTML = "<p style='color:green;'>Success: " + cookiesStatus
+ "</p>";
            } else {
              messageDiv.innerHTML = "<p style='color:red;'>Blocked: " + cookiesStatus
+ "</p>";
            }
            document.body.appendChild(messageDiv);
        }
    </script>
</head>
<body>
    <h2>Welcome to SafeBank Login</h2>
    <form onsubmit="handleSubmit(event)">
        Username: <input type="text" id="user" name="user"><br><br>
        Password: <input type="password" id="pass" name="pass"><br><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

What It Does:

- Pops an alert box (XSS Successful!)
- Attempts to access and log cookies
- Appends warning text to the page

Browsers were evaluated for their ability to:

- Display warnings
- Prevent cookie access (HttpOnly and SameSite)
- Block or allow DOM modification

## e) Browser Fingerprinting Script

A fingerprinting test was created using JavaScript's navigator and screen objects to collect unique browser and device information.

fingerprint.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Fingerprinting Test</title>
</head>
<body>
   <h2>Browser Fingerprint Info</h2>
   <pre id="output"></pre>

   <script>
      const output = document.getElementById("output");

      const fingerprint = {
         userAgent: navigator.userAgent,
         platform: navigator.platform,
         language: navigator.language,
         screenSize: `${screen.width}x${screen.height}`,
         timezone: Intl.DateTimeFormat().resolvedOptions().timeZone,
         plugins: navigator.plugins.length
      };

      output.textContent = JSON.stringify(fingerprint, null, 4);
      console.log("Fingerprint Data:", fingerprint);
```

```
        fetch('/submit', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(fingerprint)
        })
        .then(res => res.json())
        .then(data => console.log('Server Response:', data))
        .catch(err => console.error('Error sending data:', err));
    </script>
</body>
</html>
```

Collected Info:

- OS and browser details
- Language and timezone
- Screen resolution
- Plugin count

Purpose: To measure how much information browsers expose that can uniquely identify users - even in private browsing mode.

## f) Python Local Server Setup

To host all these test pages (except phishing), a basic Python HTTP server was used.

Command to hoist server

```
python3 -m http.server 8000
```

Python flask server

```
from flask import Flask, request, send_from_directory, jsonify
import json
import os
from datetime import datetime

app = Flask(__name__, static_folder='.')

LOG_FILE = 'fingerprints.txt'

@app.route('/')
def index():
    return send_from_directory('.', 'fin.html')

@app.route('/submit', methods=['POST'])
def submit():
    data = request.json
    if data:
        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        entry = {
            'timestamp': timestamp,
            'ip': request.remote_addr,
            'fingerprint': data
        }
        with open(LOG_FILE, 'a') as f:
            f.write(json.dumps(entry) + '\n')
        return jsonify({'status': 'success'})
    return jsonify({'status': 'failed'}), 400

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

**Steps:**

- Place all .html files (XSS, clickjacking) in one directory
- Run the command from that directory
- For fingerprint separate python server is used to store the data in a log file
- Access files using: http://ip:8000/

This server approach ensured easy testing without needing Flask or any external web frameworks.

## g) Summary

The implementation used minimal yet powerful components to simulate and test each attack. Unlike commercial toolkits, the custom-built approach provided full control and transparency for observing browser behaviour.

| **Attack** | **Method** | **Code Included** |
|---|---|---|
| Phishing | Real URLs from PhishTank | No code (live URLs used) |
| Clickjacking | HTML + CSS overlay with transparent iframe | Yes |
| XSS Injection | Embedded JavaScript inside test page | Yes |
| Fingerprinting | JS using navigator & screen objects | Yes |
| Server Hosting | Python built-in http.server module | Yes |

# VIII. Report

## a) Introduction

This section presents the findings of the browser security analysis based on the implementation of four different types of web-based attacks. Each attack vector was tested in a controlled environment across four popular mobile browsers: Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave.

The goal of this analysis is to evaluate how well each browser handles:

- Phishing attacks
- Clickjacking attempts
- JavaScript-based XSS injections
- Browser fingerprinting

## b) Phishing Attack Detection

Phishing attacks were simulated using live phishing URLs obtained from the PhishTank dataset. Each browser was tested by accessing the same phishing URLs, and their response behaviour was recorded.

Experimental Setup

- Source: PhishTank
- Device: Android emulator (Bliss OS)
- Environment: Hosted in Kali Linux via VMware

Observations

| Browser | Detection Rate | Warned User? | Blocked Page? | Allowed HTTPS Phish? |
|---|---|---|---|---|
| Google Chrome | 40% | Yes (some) | Yes (partial) | Yes |
| Brave | 20% | No | No | Yes |
| Microsoft Edge | 50% | Yes(more) | Yes | Some |
| Mozilla Firefox | 40% | Yes(some) | Partial | Yes |

Insights

- All browsers performed better with HTTP phishing than HTTPS phishing.
- Edge had the best detection rate at 50%.
- Brave underperformed in detecting live phishing links.
- The padlock icon misled users, as many phishing pages were hosted with valid SSL certificates.

## c) Clickjacking Test

Clickjacking was simulated using a custom-built HTML page with a transparent iframe layered over a visible fake button.

Test Conditions

- HTML/CSS overlay with transparent iframe
- Page hosted on local Python server
- No X-Frame-Options or CSP header used intentionally

Results

| Browser | Clickjacking Prevented? | Overlay Blocked? | Warning Shown? |
|---|---|---|---|
| Google Chrome | NO | NO | NO |
| Brave | NO | NO | NO |
| Microsoft Edge | NO | NO | NO |
| Mozilla Firefox | NO | NO | NO |

Insights

- All browsers were vulnerable to the clickjacking attack.
- None of them enforced frame isolation by default.
- No visual or security warning was triggered during the interaction.

### d) XSS (Cross-Site Scripting) Injection

The XSS test embedded JavaScript in a local test page to display alerts, modify content, and attempt to access cookies.

Test Actions

- Alert: alert("XSS Successful!")
- Cookie theft: document.cookie
- DOM manipulation: document.body.innerHTML

Results

| Browser | Alert Triggered | Cookie-Theft Blocked | DOM Modified | Warning Shown |
|---|---|---|---|---|
| Google Chrome | YES | Partial | YES | NO |
| Brave | YES | Protected | YES | Partial |
| Edge | YES | Protected (SmartScreen) | YES | NO |

| | | | | |
|---|---|---|---|---|
| Firefox | YES | Protected (Cookie Policy) | YES | NO |

Insights

- The alert and DOM script ran on all browsers.
- Cookie access was blocked in most cases due to HttpOnly and SameSite flags.
- Only Brave showed a partial warning, others remained silent.

## e) Fingerprinting Resistance

JavaScript fingerprinting tested how much unique information each browser exposed.

Data Collected

| |
|---|
| ➢ navigator.userAgent |
| ➢ navigator.platform |
| ➢ screen.width / height<br>➢ language, timezone |
| ➢ Plugin count |

Results

| Browser | Fingerprint Score (Trackability) | JavaScript Fingerprint Blocked? |
|---|---|---|
| Google Chrome | 92% (Highly fingerprintable) | NO |
| Brave | 45% (Better privacy) | YES |
| Edge | 88% | NO |
| Firefox | 80% | Partial |

Google Chrome

{"timestamp": "2025-04-13 16:42:27", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36", "platform": "Linux armv81", "language": "en-US", "screenSize": "393x873", "timezone": "Asia/Calcutta", "plugins": 0}}

Brave

{"timestamp": "2025-04-13 16:43:11", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36", "platform": "Linux armv81", "language": "en-IN", "screenSize": "393x873", "timezone": "Asia/Calcutta", "plugins": 2}}

Edge

{"timestamp": "2025-04-13 16:44:30", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36 EdgA/135.0.0.0", "platform": "Linux armv81", "language": "en-IN", "screenSize": "393x873", "timezone": "Asia/Calcutta", "plugins": 0}}

Firefox

{"timestamp": "2025-04-13 16:43:54", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Android 12; Mobile; rv:137.0) Gecko/137.0 Firefox/137.0", "platform": "Linux armv81", "language": "en-IN", "screenSize": "396x880", "timezone": "Asia/Kolkata", "plugins": 5}}

Insights

- Chrome and Edge leaked the most identifiable information.
- Brave blocked or spoofed many properties.
- Firefox hid some fields but was still moderately trackable.

## f) Summary of Results

| Attack Type | Best Performer | Most Vulnerable |
|---|---|---|
| Phishing | Microsoft Edge (50%) | Brave (20%) |
| Clickjacking | None (All vulnerable) | ALL |
| XSS | Brave (partial alert) | All (alert executed) |
| Fingerprinting | Brave (least data exposed) | Chrome, Edge |

## g) General Observations

- Security mechanisms vary across browsers, but no browser was flawless.
- Brave demonstrated better privacy for fingerprinting, but weaker phishing detection.
- Edge showed slightly stronger phishing protection but leaked fingerprinting data.
- Clickjacking remains unaddressed across the board — a critical oversight.
- XSS protections depend mostly on cookie flags, not script execution prevention.

# IX. Screenshots

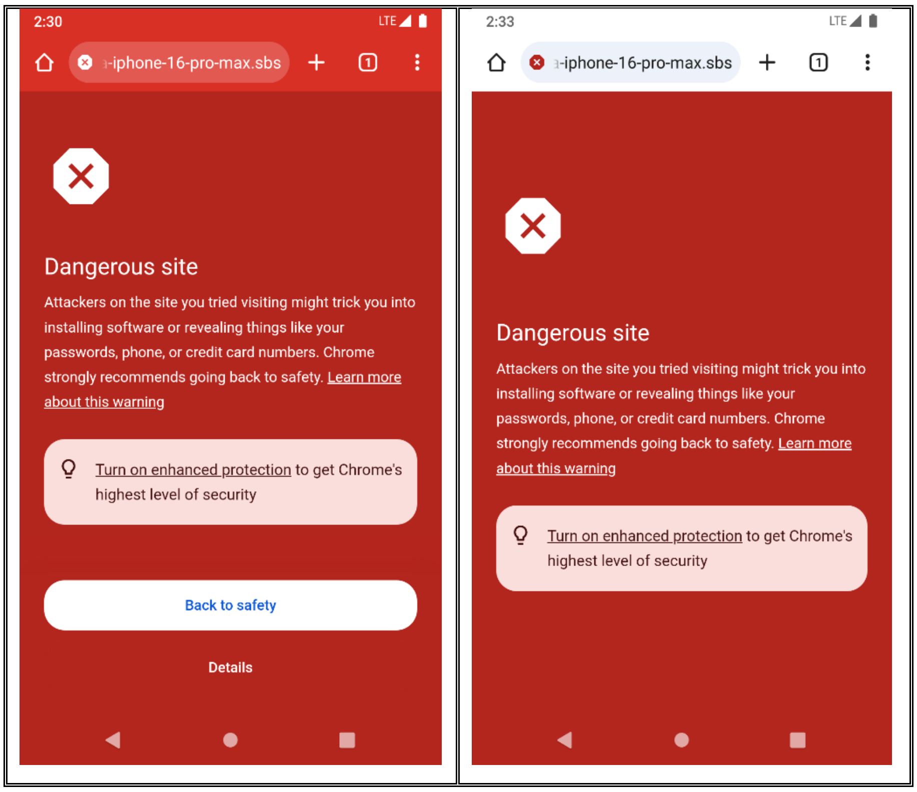
## a) Phishing attacks

For demonstrating phishing attacks, the following URLs from the PhishTank database were utilized solely for capturing and presenting screenshots as part of the research:

- http://allegrolokalnie.pl-oferta-iphone-16-pro-max.sbs/
- http://allegro.pl-oferta-iphone-16-pro-max.sbs/
- https://lhv-panka-022.in.net/LHV/
- https://unisqurfingrup.com/

These websites were not interacted with in any way beyond visual inspection, and no user credentials or personal data were submitted. Their use is limited strictly to academic and illustrative purposes in support of phishing detection research.

- Google Chrome
    - HTTP Phishing URL



2:30 LTE

-iphone-16-pro-max.sbs

Dangerous site

Attackers on the site you tried visiting might trick you into installing software or revealing things like your passwords, phone, or credit card numbers. Chrome strongly recommends going back to safety. Learn more about this warning

Turn on enhanced protection to get Chrome's highest level of security

Back to safety

Details

2:33 LTE

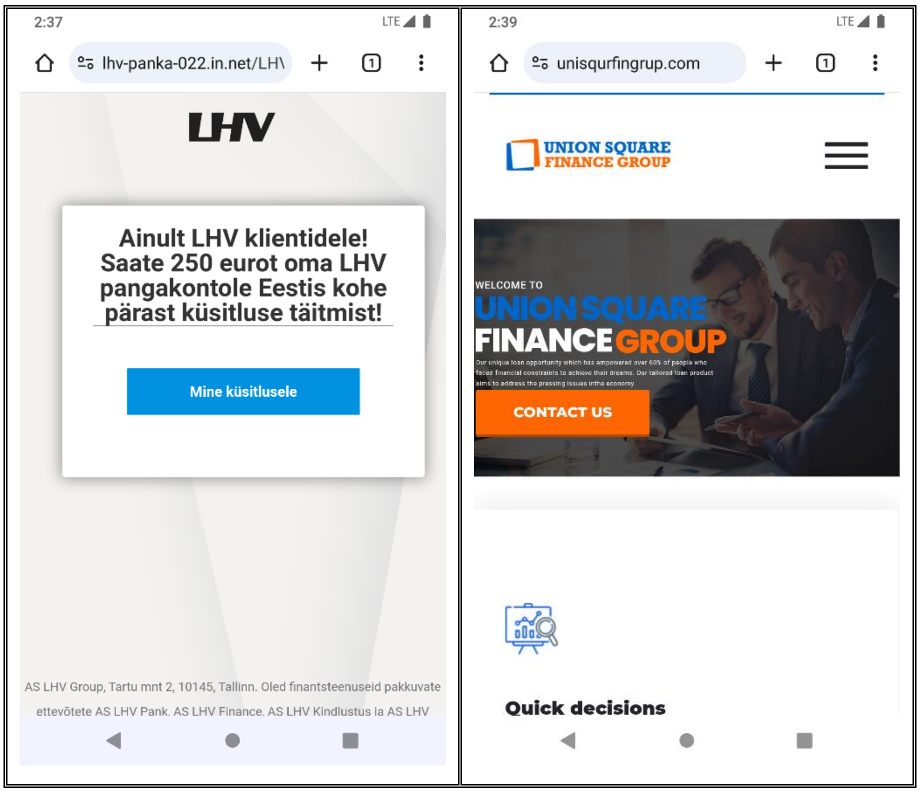
-iphone-16-pro-max.sbs

Dangerous site

Attackers on the site you tried visiting might trick you into installing software or revealing things like your passwords, phone, or credit card numbers. Chrome strongly recommends going back to safety. Learn more about this warning

Turn on enhanced protection to get Chrome's highest level of security

- HTTPS Phishing URL


2:37

lhv-panka-022.in.net/LHV

Ainult LHV klientidele! Saate 250 eurot oma LHV pangakontole Eestis kohe pärast küsitluse täitmist!

Mine küsitlusele

AS LHV Group, Tartu mnt 2, 10145, Tallinn. Oled finantsteenuseid pakkuvate ettevõtete AS LHV Pank. AS LHV Finance. AS LHV Kindlustus ja AS LHV

2:39

unisqurfingrup.com

WELCOME TO

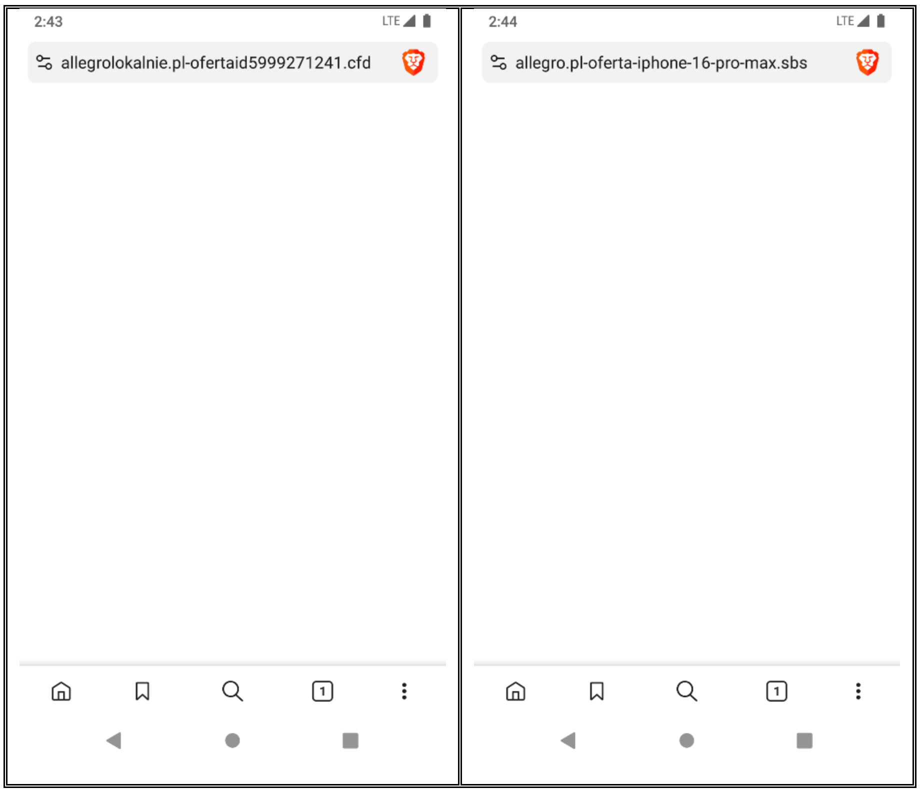
UNION SQUARE FINANCE GROUP

Our unique loan opportunity which has empowered over 60% of people who faced financial constraints to achieve their dreams. Our tailored loan product aims to address the pressing issues inthe economy.

CONTACT US

Quick decisions

- Brave
  - HTTP Phishing URL

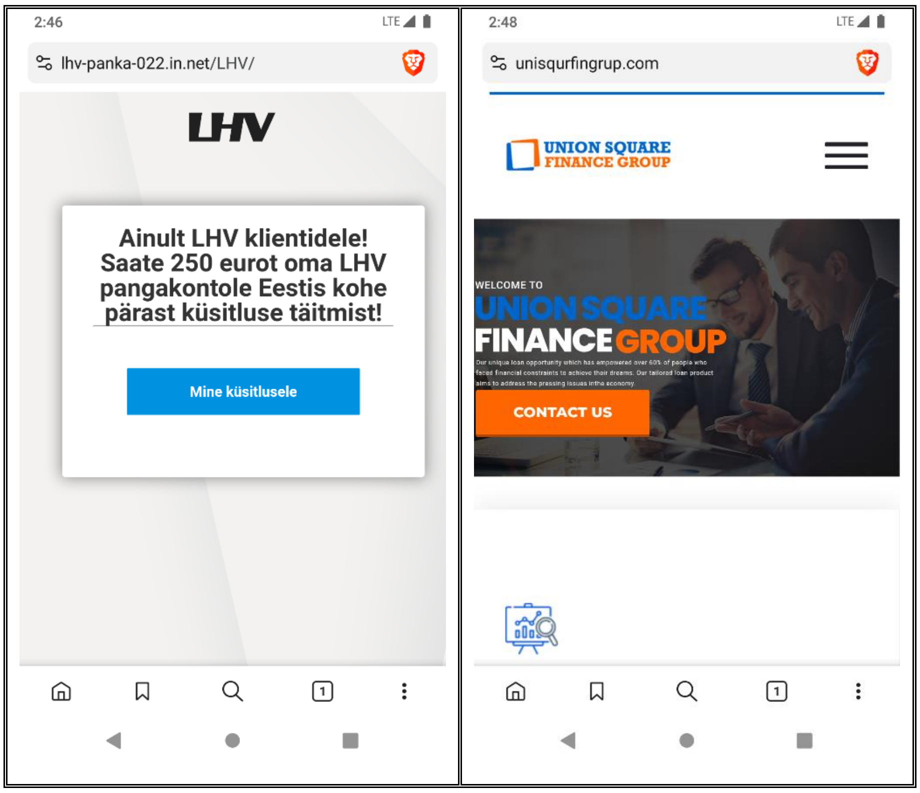

2:43

allegrolokalnie.pl-ofertaid5999271241.cfd

2:44

allegro.pl-oferta-iphone-16-pro-max.sbs

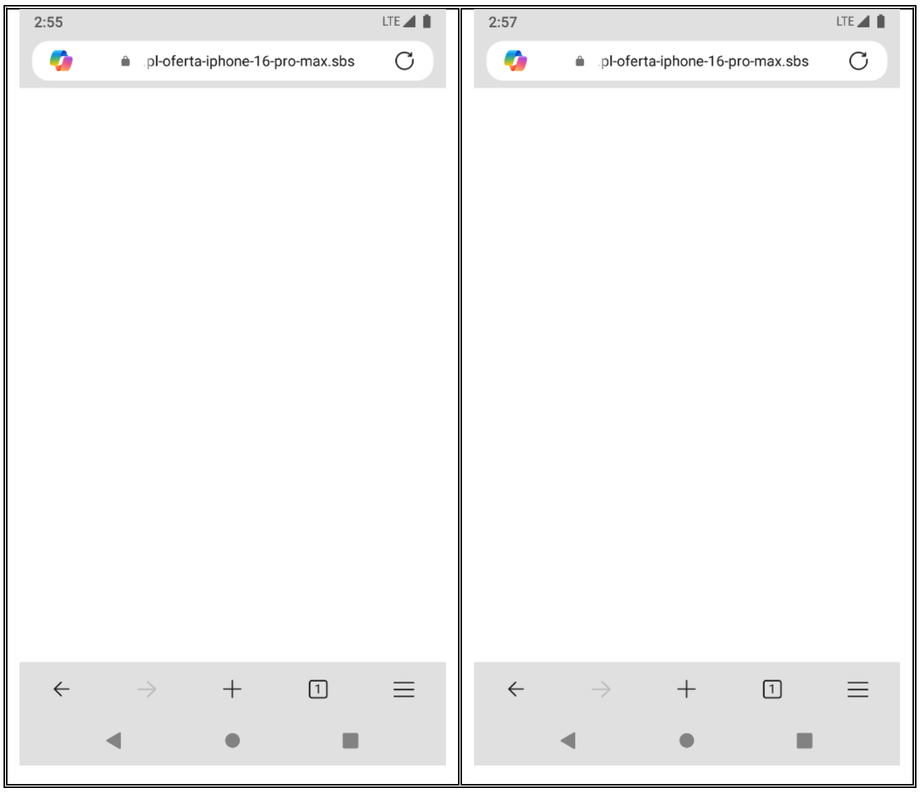
- HTTPS Phishing URL


2:46
lhv-panka-022.in.net/LHV/
LHV
Mine küsitlusele
2:48
unisqurfingrup.com
UNION SQUARE FINANCE GROUP
WELCOME TO
UNION SQUARE FINANCE GROUP
CONTACT US

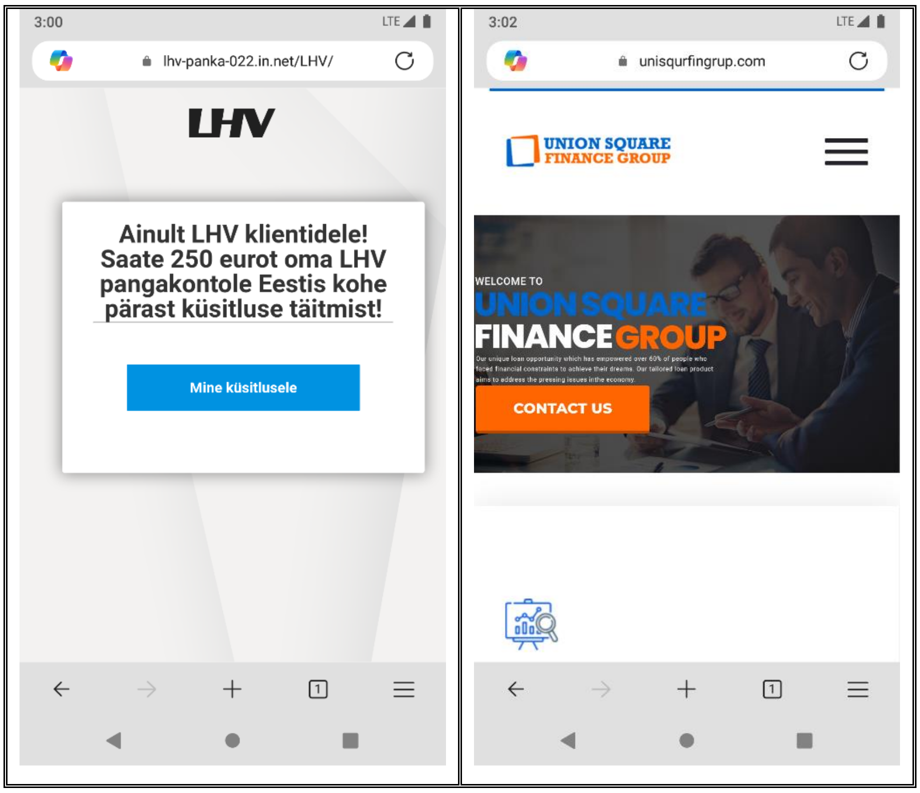
Ainult LHV klientidele! Saate 250 eurot oma LHV pangakontole Eestis kohe pärast küsitluse täitmist!

Our unique loan opportunity which has empowered over 60% of people who faced financial constraints to achieve their dreams. Our tailored loan product aims to address the pressing issues inthe economy.

- Edge
  - HTTP Phishing URL


2:55
pl-oferta-iphone-16-pro-max.sbs
2:57
pl-oferta-iphone-16-pro-max.sbs

- HTTPS Phishing URL



3:00 lhv-panka-022.in.net/LHV/

LHV

Ainult LHV klientidele! Saate 250 eurot oma LHV pangakontole Eestis kohe pärast küsitluse täitmist!

Mine küsitlusele

3:02 unisqurfingrup.com

UNION SQUARE FINANCE GROUP

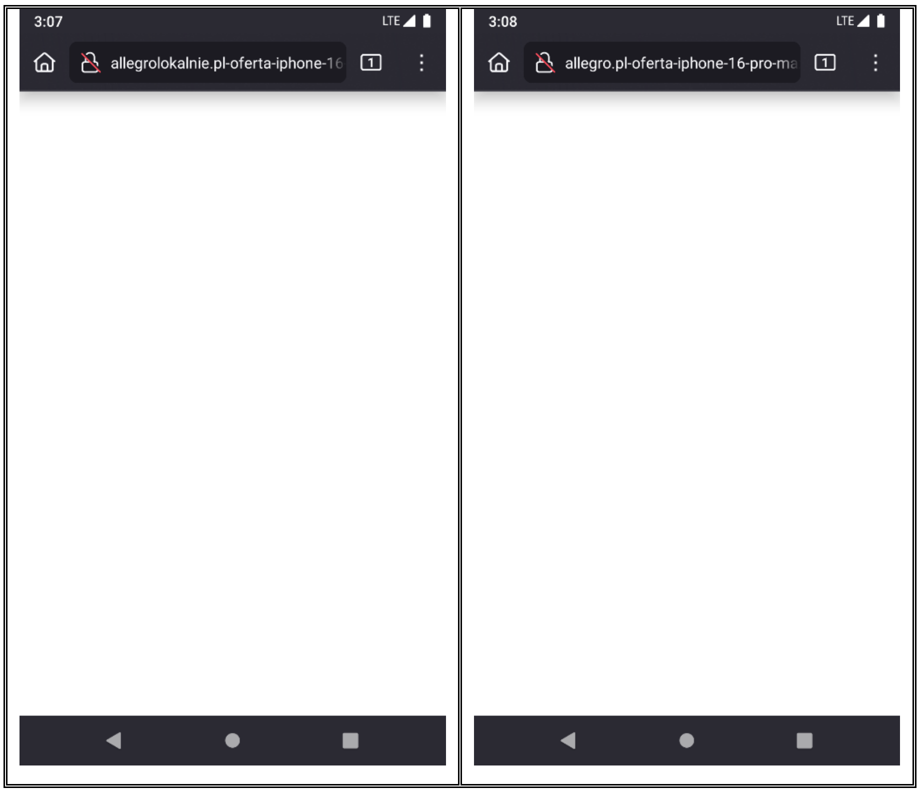
WELCOME TO UNION SQUARE FINANCE GROUP

Our unique loan opportunity which has empowered over 60% of people who faced financial constraints to achieve their dreams. Our tailored loan product aims to address the pressing issues in the economy.

CONTACT US

➢ Firefox

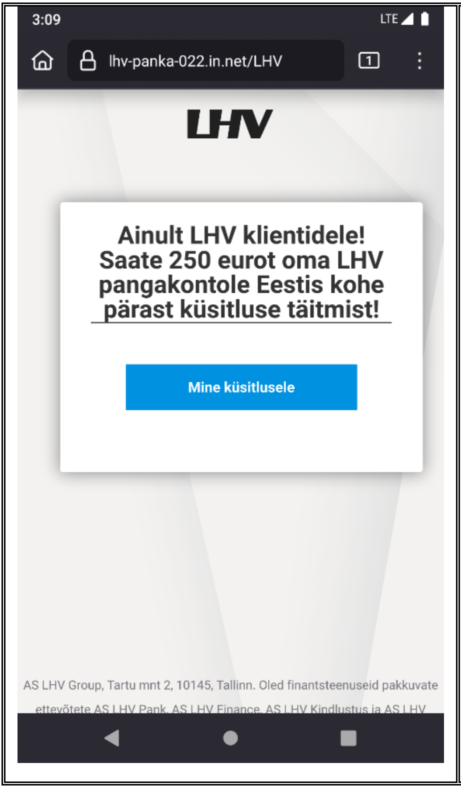
- HTTP Phishing URL



3:07 allegrolokalnie.pl-oferta-iphone-16

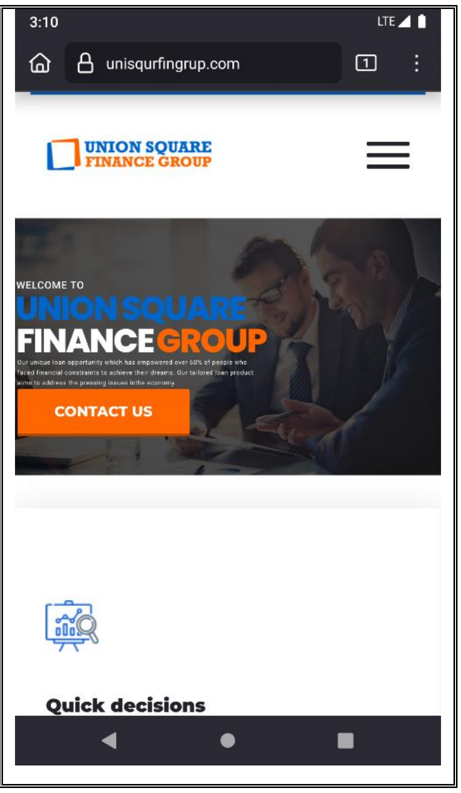
3:08 allegro.pl-oferta-iphone-16-pro-ma

- HTTPS Phishing URL



Ainult LHV klientidele! Saate 250 eurot oma LHV pangakontole Eestis kohe pärast küsitluse täitmist!

AS LHV Group, Tartu mnt 2, 10145, Tallinn. Oled finantsteenuseid pakkuvate ettevõtete AS LHV Pank, AS LHV Finance, AS LHV Kindlustus ja AS LHV



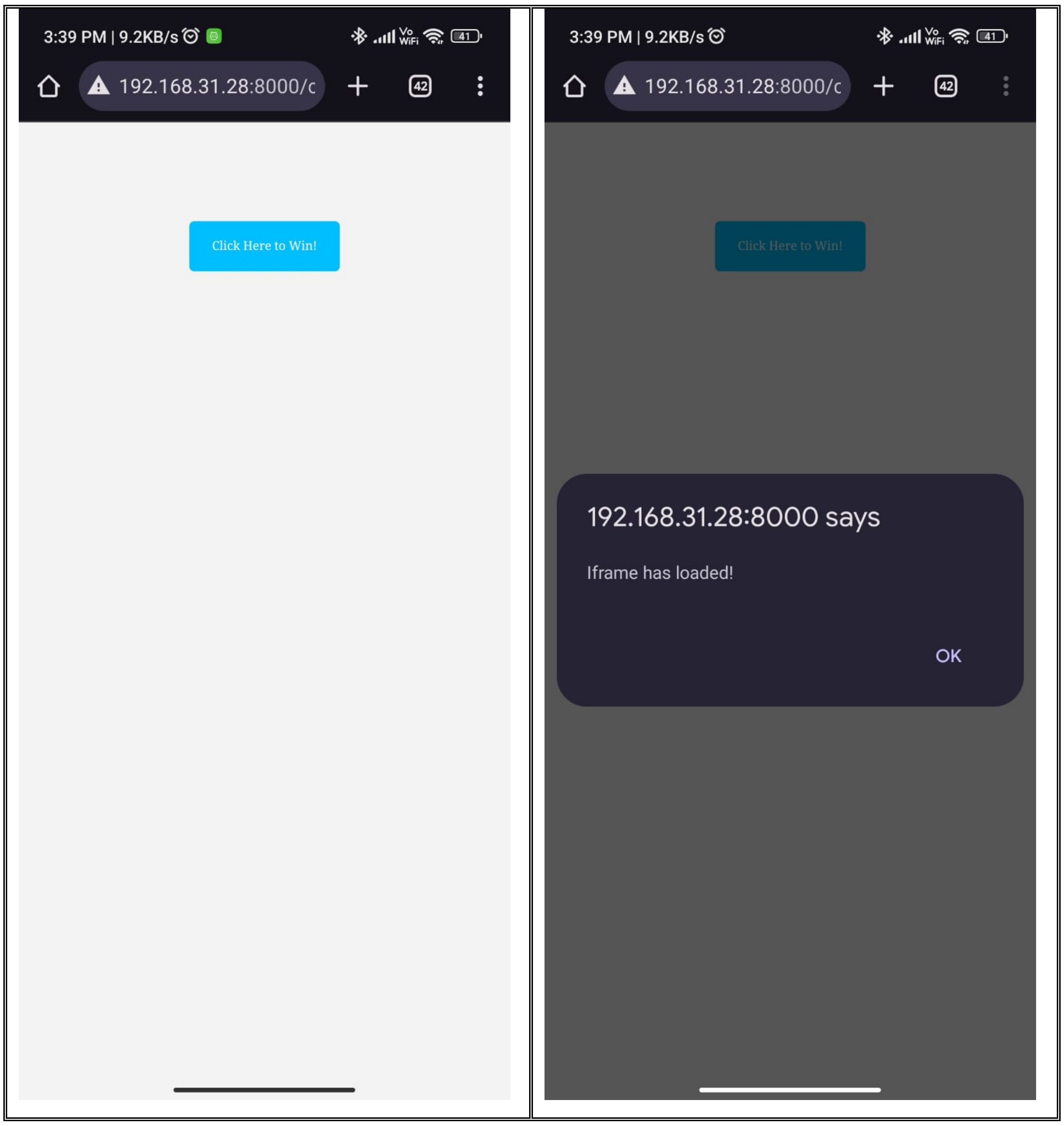
## b) Clickjacking Test

To demonstrate clickjacking attacks, custom-designed web pages were created as part of a controlled environment. These pages visually mimic legitimate interfaces while embedding invisible or disguised clickable elements that redirect user actions without their knowledge.

The primary objective was to simulate how attackers can trick users into interacting with hidden elements, such as “Like” buttons or download links, layered beneath seemingly harmless content. This demonstration was carried out in a secure, offline setting solely for research and educational purposes.

No real-world websites or external services were used or affected during this simulation.

- Google Chrome

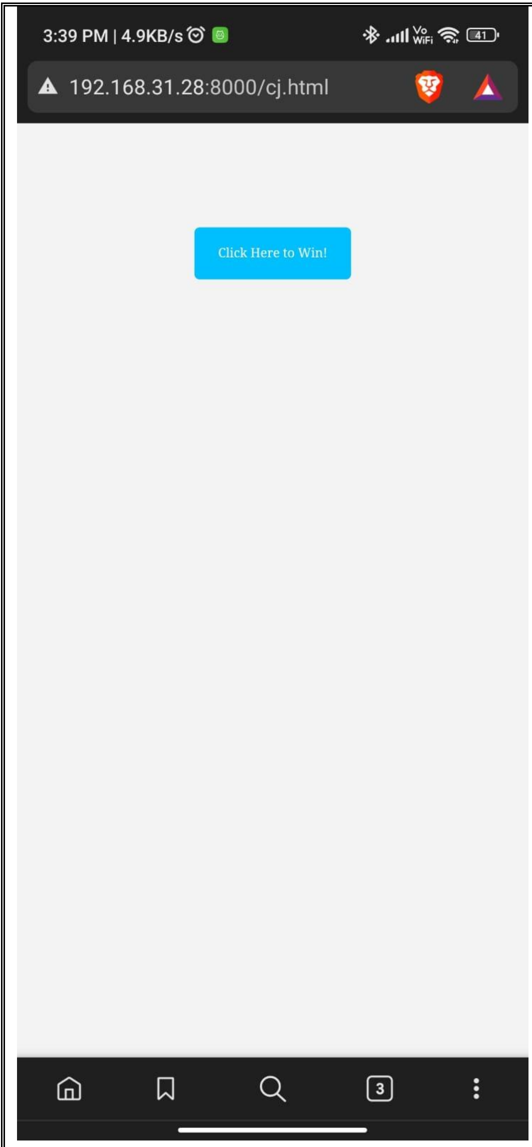


192.168.31.28:8000 says

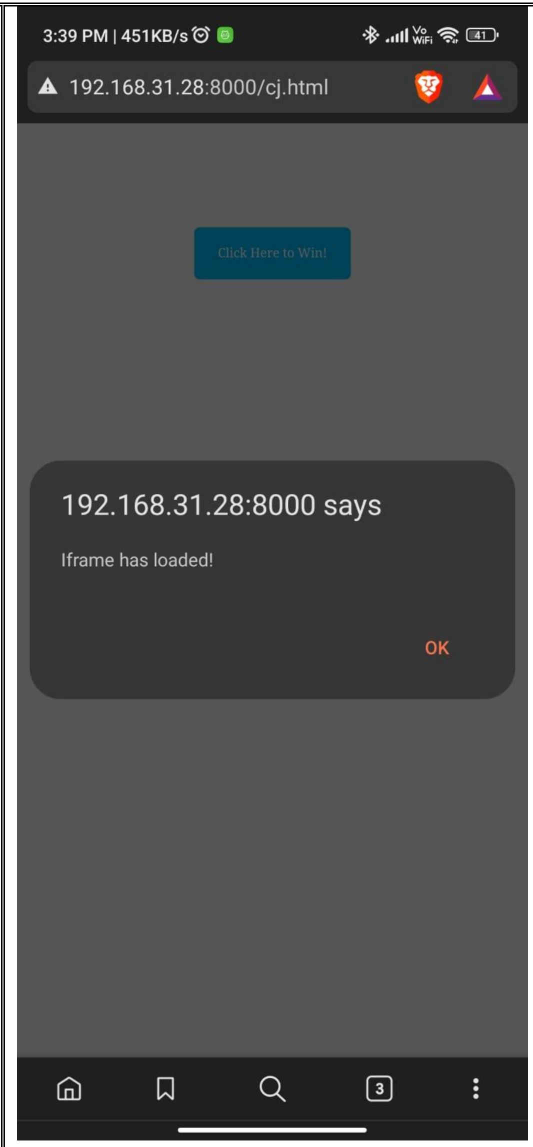
Iframe has loaded!

OK

➢ Brave



192.168.31.28:8000/cj.html

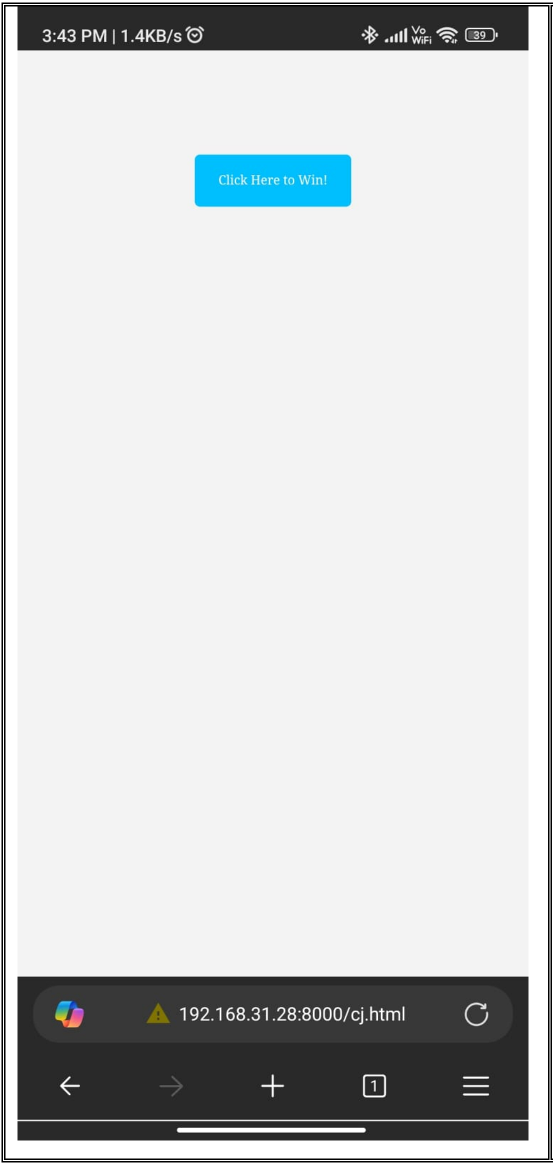


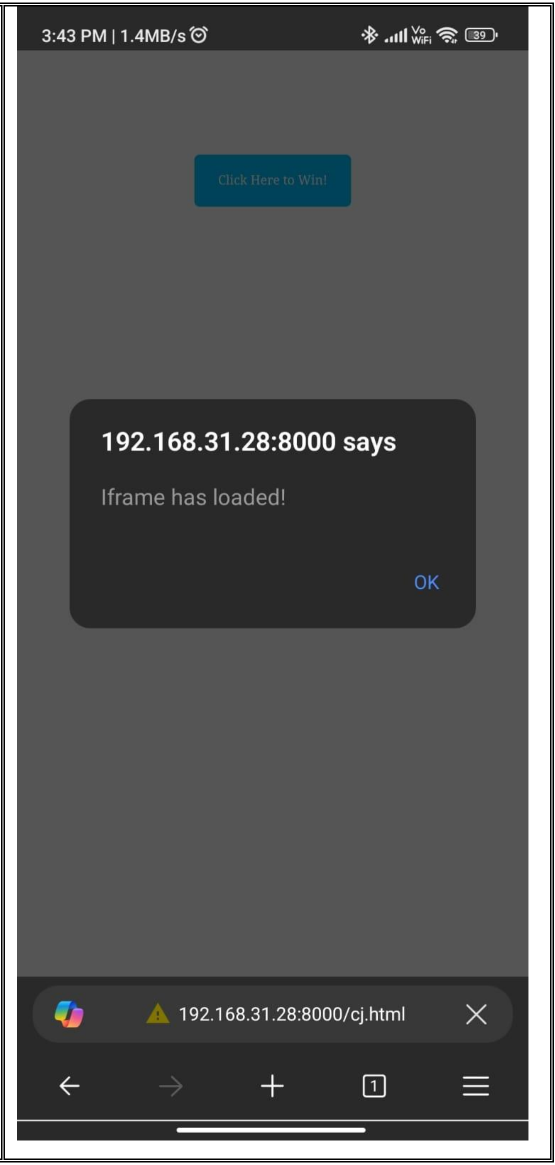
192.168.31.28:8000/cj.html

192.168.31.28:8000 says

Iframe has loaded!

- Edge



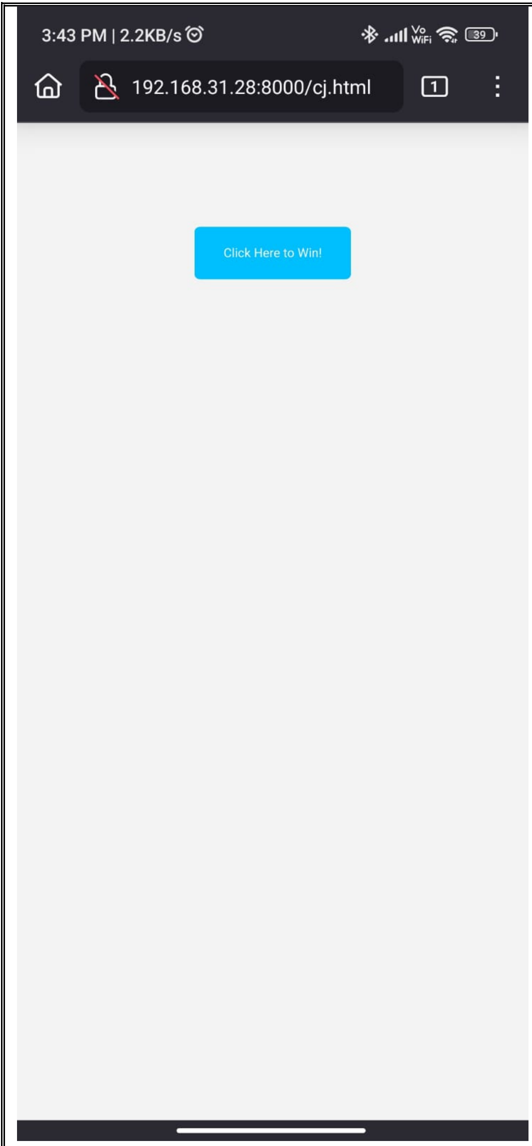


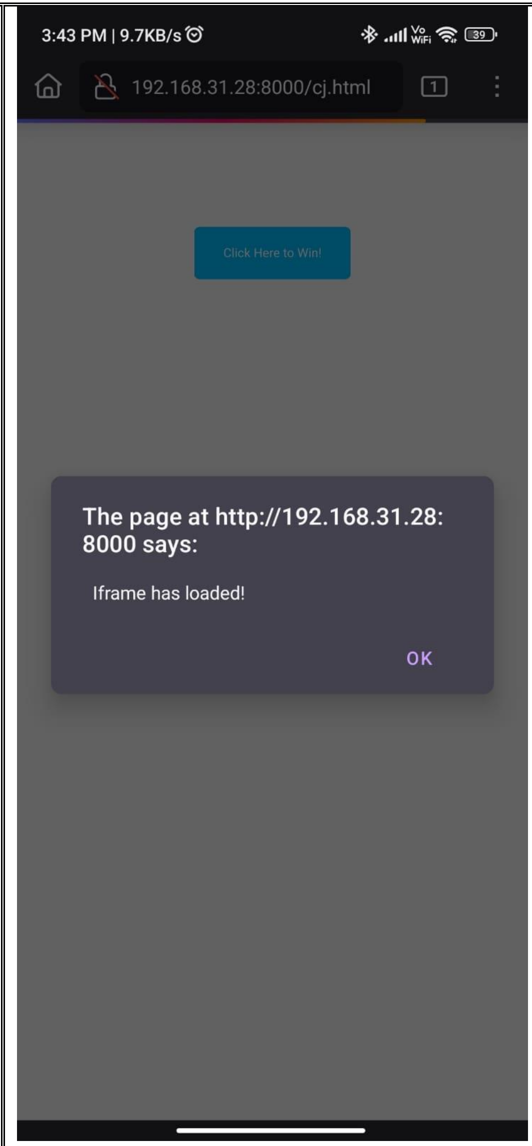
192.168.31.28:8000 says

Iframe has loaded!

OK

➢ Fire Fox





The page at http://192.168.31.28:8000 says:

Iframe has loaded!

OK

### c) Fingerprinting Test

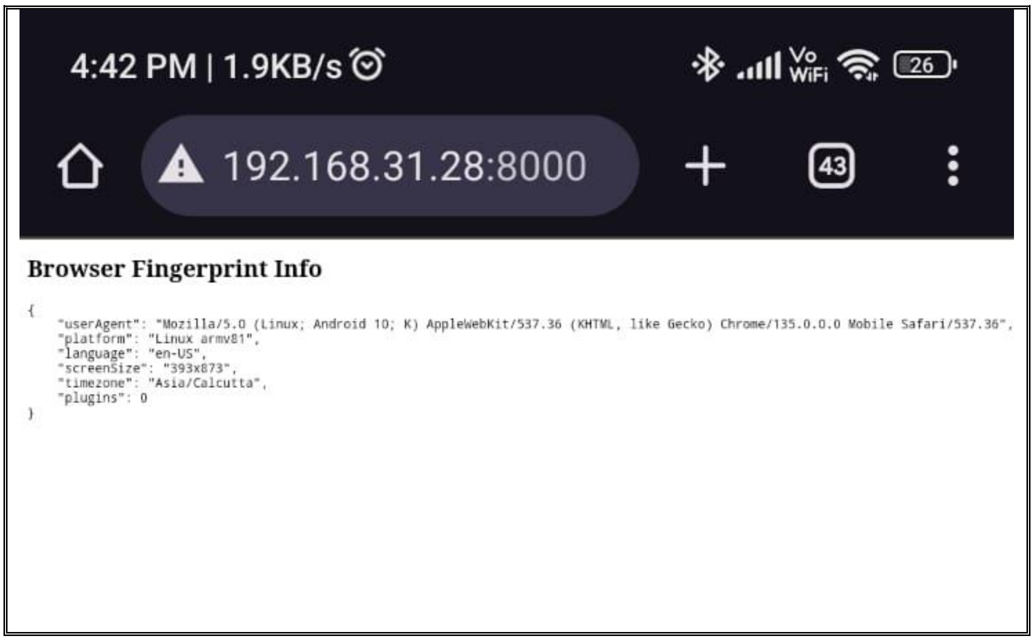
To evaluate **fingerprinting resistance**, custom scripts and test pages were developed to simulate common browser fingerprinting techniques. These techniques aim to uniquely identify users based on a combination of device-specific and browser-specific attributes such as:

- User-Agent string
- Screen resolution and colour depth
- Timezone and language settings
- Installed fonts and plugins
- Canvas and WebGL rendering data

The test environment was designed to assess how much identifiable information could be collected from a user's browser without their explicit consent. The findings highlight the importance of implementing countermeasures such as anti-fingerprinting browser extensions, privacy-focused browsers, and minimizing script permissions.

All testing was conducted in a controlled, offline setup strictly for academic and research purposes, without accessing or affecting any real user data.

- Goggle Chrome



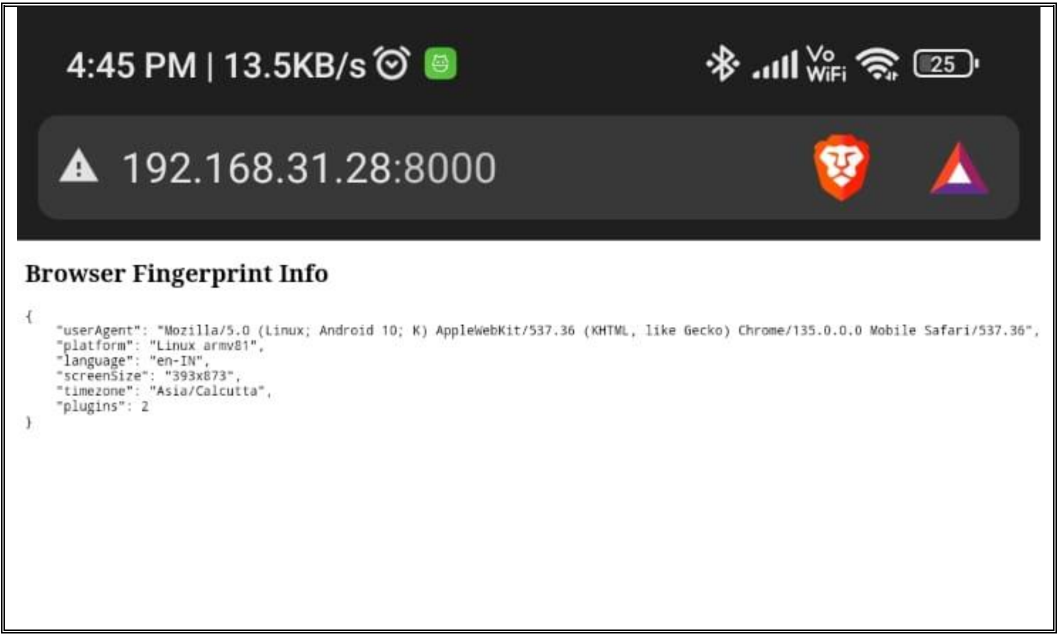
Browser Fingerprint Info

```
{
    "userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36",
    "platform": "Linux armv81",
    "language": "en-US",
    "screenSize": "393x873",
    "timezone": "Asia/Calcutta",
    "plugins": 0
}
```

Log_file

```
{"timestamp": "2025-04-13 16:42:27", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36", "platform": "Linux armv81", "language": "en-US", "screenSize": "393x873", "timezone": "Asia/Calcutta", "plugins": 0}}
```

- Brave



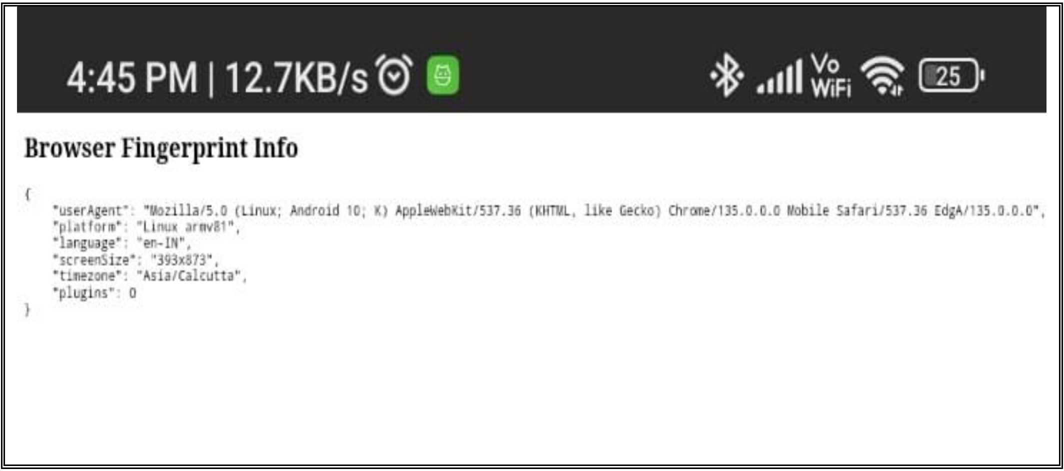
**Browser Fingerprint Info**

```
{
    "userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36",
    "platform": "Linux armv81",
    "language": "en-IN",
    "screenSize": "393x873",
    "timezone": "Asia/Calcutta",
    "plugins": 2
}
```

Log_file

```
{"timestamp": "2025-04-13 16:43:11", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36", "platform": "Linux armv81", "language": "en-IN", "screenSize": "393x873", "timezone": "Asia/Calcutta", "plugins": 2}}
```

- Edge



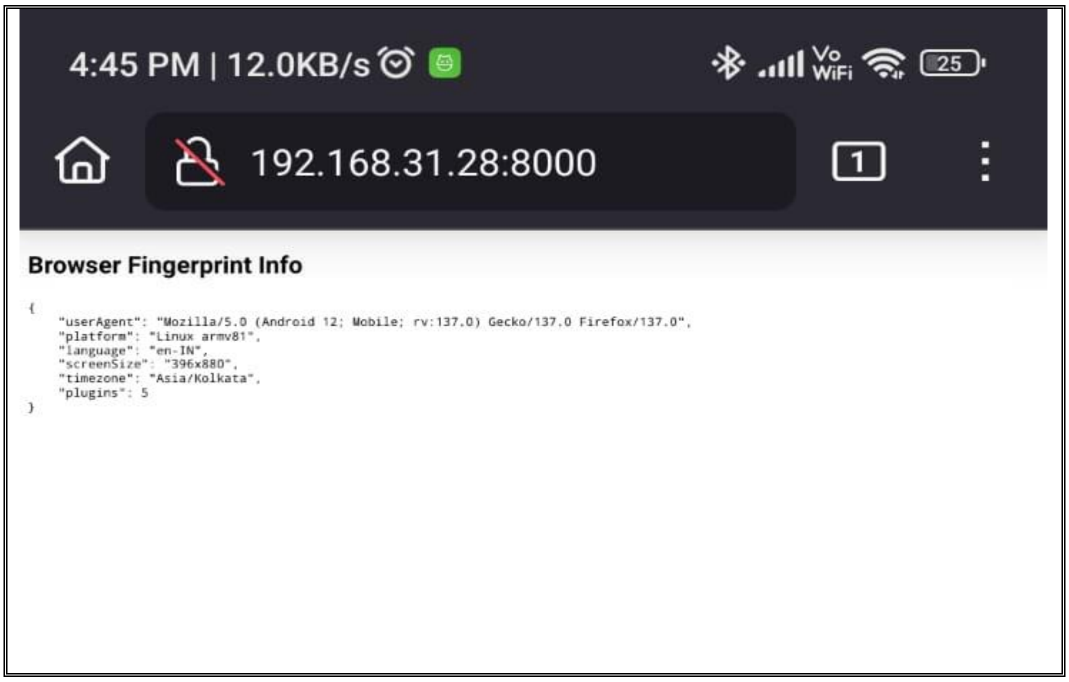
Browser Fingerprint Info

```
{
    "userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36 EdgA/135.0.0.0",
    "platform": "Linux armv81",
    "language": "en-IN",
    "screenSize": "393x873",
    "timezone": "Asia/Calcutta",
    "plugins": 0
}
```

Log_file

```
{"timestamp": "2025-04-13 16:44:30", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Mobile Safari/537.36 EdgA/135.0.0.0", "platform": "Linux armv81", "language": "en-IN", "screenSize": "393x873", "timezone": "Asia/Calcutta", "plugins": 0}}
```

➢ Fire Fox



Browser Fingerprint Info

```
{
    "userAgent": "Mozilla/5.0 (Android 12; Mobile; rv:137.0) Gecko/137.0 Firefox/137.0",
    "platform": "Linux armv81",
    "language": "en-IN",
    "screenSize": "396x880",
    "timezone": "Asia/Kolkata",
    "plugins": 5
}
```

Log_file

```
{"timestamp": "2025-04-13 16:43:54", "ip": "192.168.31.106", "fingerprint": {"userAgent": "Mozilla/5.0 (Android 12; Mobile; rv:137.0) Gecko/137.0 Firefox/137.0", "platform": "Linux armv81", "language": "en-IN", "screenSize": "396x880", "timezone": "Asia/Kolkata", "plugins": 5}}
```

# X. Conclusion

## a) Overview

In the modern digital era, mobile browsers serve as the primary gateway through which billions of users access the web. While desktop browser security has matured considerably over the years, mobile browser security remains a relatively underexplored and highly vulnerable space. Through this research project, an in-depth analysis of mobile browser security was conducted to identify weaknesses, test modern attack vectors, and evaluate how popular browsers respond to real-world threats.

The study focused on four common and impactful forms of web-based attacks:

- Phishing attacks (tested using real URLs from PhishTank)
- Clickjacking attacks (via CSS overlays and iframe injections)
- JavaScript-based XSS attacks (using direct script injection)
- Browser fingerprinting (via the navigator and screen JavaScript APIs)

Popular mobile browsers — Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave — were tested for their response and resistance to these threats in a controlled, emulator-based lab environment.

## b) Key Learnings and Outcomes

- **Real-World Phishing URLs Remain Effective**

  Despite advances in phishing detection technology, many phishing sites listed in the PhishTank database successfully bypassed browser defenses — especially when served over HTTPS. This highlights a critical gap in trust signals; users are misled by the presence of SSL certificates and padlock icons. Even browsers with robust blacklisting (like Edge) failed to detect all URLs.

  **Lesson**: Phishing protection in browsers is still reactive and cannot keep up with the dynamic nature of phishing campaigns.

- **Clickjacking Defences Are Practically Nonexistent**

All tested browsers failed the clickjacking tests. Transparent iframes placed above legitimate buttons were never flagged, nor blocked, and no warnings were shown. This is particularly alarming because mobile devices do not support hover effects or advanced visual cues that could help users detect such manipulation.

**Lesson:** Clickjacking is a serious and overlooked threat on mobile platforms — both browsers and websites must share responsibility for its prevention.

- **XSS Remains Partially Unfiltered at Client-Side**

Embedded JavaScript payloads executed without restriction across all browsers, confirming that modern browsers rarely implement client-side XSS filtering. While cookie protections like HttpOnly and SameSite blocked direct theft, scripts could still hijack the page DOM, alter content, or launch deceptive modals.

**Lesson:** Browser-level XSS prevention is weak. Security largely depends on developer-side hygiene, which cannot be guaranteed across the internet.

- **Browser Fingerprinting Is a Major Privacy Concern**

Most browsers revealed highly identifying fingerprint data through JavaScript, even in private browsing modes. Brave was the only browser that actively obfuscated or limited data points such as plugin lists and screen dimensions. Chrome and Edge were found to be the most fingerprint able.

**Lesson**: Users are unknowingly tracked across sessions through fingerprinting, and browser vendors are not doing enough to prevent it.

## c) Research Challenges Faced

This project was conducted using open-source tools and manual scripting methods instead of commercial testing platforms. A few challenges encountered include:

- Manual logging and observation: All results had to be recorded and verified through real-time user observation.
- PhishTank URL expiry: Many URLs became inactive quickly and had to be refreshed or replaced regularly.
- No forensic tools used: While some studies use tools like Autopsy, this project focused purely on active behavioural testing.
- Emulator limitations: Certain browser features behave slightly differently in emulated environments compared to physical devices.

Despite these limitations, the project successfully simulated and analysed all four attack vectors and delivered actionable findings.

## d) Security Implications of the Study

The findings of this research carry serious implications for browser developers, mobile operating system vendors, and end-users:

For Browser Developers:

- Implement client-side XSS filters and DOM sanitizer modules.
- Strengthen clickjacking prevention mechanisms (e.g., visual overlays, iframe detection).
- Include pre-render scanning for phishing or deceptive content using AI or behavioral pattern analysis.
- Develop anti-fingerprinting layers that rotate or spoof identifying information by default in private mode.

For Web Developers:

- Enforce headers such as X-Frame-Options, Content-Security-Policy, and Strict-Transport-Security.
- Use JavaScript responsibly to avoid contributing to fingerprinting vectors.
- Regularly test pages against common exploit scripts and overlays.

For Users:

- Understand that private mode does not guarantee anonymity.
- Treat all websites with caution — including those with HTTPS and padlocks.
- Use privacy-focused browsers or extensions that block fingerprinting and malicious scripts.

## e) Future Enhancements and Scope

This project opens up several avenues for future research, development, and security design:

- **AI-Powered Browser Défense**

Future work can focus on integrating machine learning models to analyze page behavior before loading — detecting phishing pages even if they've never been seen before.

- **Per-Tab VPN & Isolation**

Building a browser that opens each tab in a separate virtual network tunnel (per-tab VPN) could prevent cross-site tracking and enhance privacy.

- **Fingerprint Spoofing Engine**

A JavaScript-based engine that returns randomized browser data per session/tab could reduce trackability by design.

- **Security-Focused Browser Build**

With the core scripts already developed (clickjacking, fingerprinting, XSS simulation), a future project could aim to fork Chromium or Firefox to create a lightweight privacy-first browser with built-in protections.

- **Community Dataset on Browser Behaviour**

A collaborative dataset could be developed where researchers worldwide log how browsers respond to security tests. This could improve browser competition and awareness.

## f) Final Words

This project set out to answer a critical question: How secure are modern mobile browsers against basic, real-world web attacks?

The answer, as demonstrated through multiple structured tests, is not secure enough.

While browsers have made great strides in visual design, speed, and rendering performance, their defences against social engineering and JavaScript-based attacks remain insufficient. Users often assume safety due to HTTPS padlocks or incognito mode—but in reality, threats like clickjacking and fingerprinting are thriving silently.

This research calls for a paradigm shift in how mobile browser security is approached. Instead of passive safety tools, browsers must adopt active, intelligent, and adaptive defines mechanisms. Only then can the internet become a safer space for mobile users in an increasingly connected world.

## XI. Bibliography

- Chanda, R., Patel, H., & Saxena, A. (2024). *Web Browser Forensics: A Comparative Analysis of Private Browsing Modes in Desktop and Mobile Browsers*. Journal of Cybersecurity Research.
- Alotibi, R. M., Li, Z., & Luo, X. (2024). *Examining the Effectiveness of Incognito Modes in Android Browsers: A Forensic Perspective*. International Journal of Information Security.
- Hansen, J., & Grossman, R. (2022). *Clickjacking: Attacks and Defences*. WhiteHat Security Labs.
- Nikiforakis, N., Joosen, W., & Kapravelos, A. (2019). *Cookie less Monster: Exploring the Ecosystem of Web-based Browser Fingerprinting*. IEEE Symposium on Security and Privacy.
- Eckersley, P. (2010). *How Unique Is Your Web Browser?* Electronic Frontier Foundation (EFF) & Panoptic lick Project. Retrieved from https://panopticlick.eff.org
- Mozilla Developer Network. (2023). *HTTP Headers: X-Frame-Options, CSP, Same Site, HttpOnly*. Retrieved from https://developer.mozilla.org/
- Brave Software Inc. (2024). *Privacy Features in Brave Browser*. Retrieved from https://brave.com/privacy/
- PhishTank. (2025). *Community-based Phishing Database*. Retrieved from https://www.phishtank.com/
- Python Software Foundation. (2024). *http.server — Basic web server for testing*. Retrieved from https://docs.python.org/3/library/http.server.html
- W3Schools. (2024). *HTML, CSS, and JavaScript Tutorials*. Retrieved from https://www.w3schools.com
- BrowserStack. (2023). *Mobile Browser Testing Guide*. Retrieved from https://www.browserstack.com/

## XII. Daily Log

| Week | Date Range | Work-Tasks Done | Tools, Technologies Used | Remarks, Progress |
|---|---|---|---|---|
| Week 1 | (2024) Dec 2-8 | -Selection of Topic | -Google, Peers | Decided to Select Mobile Browser Vulnerability |
| Week 2 | Dec 9-15 | -Background Research | -Google, Chat-GPT, YouTube | Understood demand and dependency of mobile Browsers |
| Week 3 | Dec 16-22 | -Understanding Working of Browser | - YouTube | Got clear picture of how Browsers works |
| Week 4 | Dec 23-29 | -Studying Existing Research Papers & Other Resources | - Google scholar | Studied Research paper written by experts |
| Week 5 | (2025) Jan 5-11 | - Planning and Selecting Tests | -Google, Chat-GPT | Selected Four test measures |
| Week 6 | Jan 12-18 | - Understanding phishing & XSS & Clickjacking & Fingerprinting | -YouTube, Chat-GPT, GeeksForGeeks, W3Schools | Understood phishing & XSS & Clickjacking & Fingerprinting |

| | | | | |
|---|---|---|---|---|
| Week 7 | Jan 19-25 | - Selecting Resources | -Phishtank, VM-ware, VS code, Python, Server-clint, Android studio | made list of resources needed |
| Week 8 | Jan 26- Feb1 | - Developing Scripts and testing | -VM Ware, VS Code | Created and tested scripts |
| Week 9 | Feb 9-15 | - Conducting Main Tests on Browsers | -ALL Tools Combined | Documented results |
| Week 10 | Feb 16-22 | - Understanding Format of research Paper and Selecting Format | -YouTube, Google, sample Research paper | Selected Format |
| Week 11 | Feb 23- Mar 1 | - Writing Research paper | -MS Word, Grammarly | Completed |

## 1. Week 1 Dec 2-8, 2024

**Objective:**
To finalize a research topic aligned with cybersecurity, focused on an area with real-world relevance and technical depth.

**Tasks Performed:**

- Held informal discussions with peers and mentors to explore trending research topics in web and mobile security.
- Searched online using Google and educational platforms for recent issues in browser-based attacks.
- Explored various domains including phishing, password security, Android malware, and web application firewalls.
- Evaluated each domain for feasibility, interest, and potential for implementation.

**Tools and Resources Used:**

- Google (research and idea discovery)
- Peer consultation
- ChatGPT for topic ideation
- College library notes on cybersecurity

**Challenges:**

- Narrowing down to a topic that's both academically rich and practically implementable.

**Outcome:**

- Finalized the research topic: **"Analyzing Security Vulnerabilities in Mobile Browsers"**
- Chose to explore phishing, clickjacking, XSS, and fingerprinting due to their relevance and demonstrability.

### 2. Week 2 Dec 9-15, 2024

**Objective:**
To understand the background, demand, and dependency on mobile browsers in daily digital life.

**Tasks Performed:**

- Conducted secondary research to evaluate how widely mobile browsers are used worldwide.
- Collected statistics on mobile vs desktop internet usage trends from recent surveys.
- Read articles and watched videos explaining user behaviour, browser habits, and mobile privacy concerns.
- Compared multiple browsers and their features in Android environments.

**Tools and Resources Used:**

- Google Search (browser usage data)
- ChatGPT (summary of browser behaviour patterns)
- YouTube (browser walkthroughs, mobile OS comparison)

**Challenges:**

- Distinguishing between marketing claims and real user-centric privacy features.

**Outcome:**

- Gained a solid understanding of why mobile browser security is more critical today than ever.
- Understood the popularity and attack surface of mobile browsers compared to desktop ones.

## 3. Week 3 Dec 16-22, 2024

**Objective:**

To understand the internal working of mobile and desktop web browsers, their architecture, and user interaction flow.

**Tasks Performed:**

- Studied browser architecture including rendering engines, JavaScript engines.
- Learned about the function of components like network layer, UI thread, sandbox, and caching mechanisms.
- Explored browser developer tools (DevTools) and learned how to inspect scripts, cookies, and DOM.
- Compared behaviour of Chrome, Brave, Edge, and Firefox on Android for common security tasks.

**Tools and Resources Used:**

- YouTube (browser engine videos and DevTools tutorials)
- Google (for comparing mobile browser architecture)

**Challenges:**

- Understanding browser internals without direct source code access (due to complexity of Chromium-based systems)

**Outcome:**

- Gained technical clarity on how browsers render pages, execute scripts, and handle security features.
- This knowledge directly supported further testing in upcoming weeks, particularly for XSS and fingerprinting experiments.

## 4. Week 4 Dec 23-29, 2024

**Objective:**

To study existing literature, technical papers, and security research related to mobile browser vulnerabilities.

**Tasks Performed:**

- Searched for and downloaded research papers from Google Scholar.
- Analysed past studies focusing on browser privacy claims versus forensic realities.
- Reviewed how researchers used tools like forensic software.
- Compared traditional security testing (black-box, white-box) with behavioural browser-based approaches.

**Tools and Resources Used:**

- Google Scholar
- ChatGPT (to summarize and explain technical jargon in papers)

**Challenges:**

- Some papers had paywalls or limited access — resolved by searching for free preprints or summaries.
- Terminologies in formal academic papers were highly complex and required interpretation.

**Outcome:**

- Built a foundational understanding of what has already been done in the field.
- Identified gaps that my research can address, particularly in live attack behaviour testing without using forensic tools.

## 5. Week 5 Jan 5-11, 2025

**Objective:**

To finalize the testing scope by selecting key attacks to simulate and prepare a structured testing plan.

**Tasks Performed:**

- Drafted a list of potential browser vulnerabilities that can be ethically tested in a controlled lab.

- Selected four core attacks for implementation: Phishing, Clickjacking, XSS, Fingerprinting.
- Planned browser testing sessions with clearly defined metrics (e.g., warning displayed, cookie access, iframe rendering, script execution).
- Mapped out which components would require custom code and which could use datasets (like PhishTank).
- Organized a testing timeline and divided the project into implementation, observation, and analysis phases.

**Tools and Resources Used:**

- ChatGPT
- Notes from previous research papers

**Challenges:**

- Filtering attack types that are testable without breaching ethical or legal boundaries.
- Ensuring that no real-world harm is caused through simulations.

**Outcome:**

- Finalized testing structure and selected attack types.
- Created a high-level timeline for implementation and documentation.

## 6. Week 6 Jan 12-18, 2025

**Objective:**

To gain a deep understanding of the technical workings of each selected attack vector: phishing, XSS, clickjacking, and fingerprinting.

**Tasks Performed:**

- Watched tutorials on YouTube about phishing simulations, JavaScript injection, and clickjacking exploits.
- Studied how iframe overlays can be used for clickjacking and how to craft a transparent layer on buttons.
- Learned about navigator, screen, and document APIs used in browser fingerprinting.
- Explored how XSS payloads can alter the DOM, trigger alerts, or steal cookies using scripts.
- Documented implementation strategies for each attack, deciding which would use live URLs (PhishTank) and which would require coding.

**Tools and Resources Used:**

- YouTube (XSS/Phishing/Clickjacking tutorials)
- W3Schools and GeeksForGeeks (for HTML/CSS/JS syntax)
- ChatGPT (to generate and explain small scripts for testing)

**Challenges:**

- Ensuring test scripts don’t go beyond the ethical scope.
- Confirming that test pages behave similarly across different mobile browsers.

**Outcome:**

- Gained strong technical clarity on all four attacks.
- Prepared to move forward with script writing and test environment setup.

## 7. Week 7 Jan 19-25, 2025

**Objective:**

To gather all required tools and resources, set up the testing environment, and begin preparing the system for attack simulation.

**Tasks Performed:**

- Installed and configured **VMware** and set up **Kali Linux** as the testing OS.
- Installed **Bliss OS** (Android emulator) to simulate mobile browser behaviour on a virtual Android device.
- Identified and downloaded the required mobile browsers: Chrome, Brave, Firefox, and Edge (Android versions).
- Set up **VS Code** for writing scripts and structured folders for each attack vector (clickjacking, XSS, fingerprinting).
- Verified internet connectivity within the emulator to test real phishing URLs from PhishTank.

**Tools and Technologies Used:**

- VMware Workstation
- Kali Linux (Rolling release)
- Bliss OS (Android emulator)
- VS Code
- PhishTank URLs
- Browsers (Chrome, Brave, Edge, Firefox for Android)

**Challenges:**

- Initial network bridge issues between VMware and Android emulator (resolved via NAT settings).
- Ensuring each browser installed cleanly without crashing in the emulator.

**Outcome:**

- Full testing environment ready: Host (Kali) + Emulator (Android) + Python server + browsers installed.
- Created directories to organize test scripts and outputs for each attack type.

## 8. Week 8 Jan 26- Feb1, 2025

**Objective:**

To write, test, and debug custom scripts for fingerprinting, clickjacking, and XSS attacks.

**Tasks Performed:**

- Developed fingerprinting script using JavaScript to collect userAgent, platform, screen resolution, timezone, and language.
- Created a clickjacking simulation page using HTML and CSS with a transparent iframe layered over a visible button.
- Built a basic XSS injection page using embedded <script> tags that triggered alerts and attempted DOM modification.
- Set up Python's HTTP server to host all test files locally and ensured browsers could access them via emulator browser.
- Performed initial trial runs in Chrome and Firefox to verify that scripts loaded correctly and simulated attacks behaved as expected.

**Tools and Technologies Used:**

- HTML, CSS, JavaScript
- Python 3.x http.server
- VS Code

- Localhost environment via Kali VM
- Mobile browsers in emulator

**Challenges:**

- Adjusting CSS positions to align the iframe exactly over the fake button.
- Ensuring JS code didn’t get blocked due to browser restrictions or server misconfiguration.

**Outcome:**

- All three test pages (fingerprinting, XSS, clickjacking) worked reliably.
- Verified access to test URLs across all mobile browsers.

## 9. Week 9 Feb 9-15, 2025

**Objective:**

To conduct the core testing phase, running simulations on all selected mobile browsers and logging responses.

**Tasks Performed:**

- Loaded PhishTank URLs in each browser and recorded whether access was allowed, blocked, or warned.
- Opened the clickjacking test page and clicked the overlaid button to see if interaction was redirected.
- Tested the XSS page in all four browsers — observed alerts, DOM changes, and access to document.cookie.
- Executed the fingerprinting script in each browser — collected outputs and compared data points.
- Created a tracking spreadsheet to log results: success/failure, response type, warnings shown, and screenshots taken.

**Tools and Technologies Used:**

- All four Android browsers
- Local Python server

- Screenshots & browser dev tools
- Manual observation sheets and data table.

**Challenges:**

- PhishTank URLs expired quickly, had to keep refreshing new active links.
- Some browsers showed inconsistencies (e.g., Edge blocked one phishing page but allowed another).

**Outcome:**

- Successfully tested all four attacks across all browsers.
- Compiled raw data to be analysed in the upcoming weeks for the final report.

## 10. Week 10 Feb 16-22, 2025

**Objective:**

To study research paper formatting guidelines, review published samples, and finalize the structure for writing the full report.

**Tasks Performed:**

- Watched videos on YouTube explaining the standard structure of a research paper: Abstract, Introduction, Literature Survey, Methodology, etc.
- Downloaded sample research papers from Google Scholar and college repositories for comparison.
- Compared APA, IEEE, and basic academic report formats to choose one suitable for college submission.
- Outlined section titles, word limits, and where to insert diagrams, tables, and code.
- Segregated screenshots, test data, and results into folders for easy inclusion.

**Tools and Resources Used:**

- YouTube tutorials on research paper writing
- Google Scholar and ResearchGate
- MS Word for outlining the document

- Grammarly (for grammar, tone, and style checking)

**Challenges:**

- Choosing between concise technical writing and a more detailed academic explanation style.
- Adapting formal language while still including code-based content and browser test outcomes.

**Outcome:**

- Finalized the complete structure of the research report.
- Made a checklist for each section to be completed in Week 11.

## 11. Week 9 Feb 23- March 1, 2025

**Objective:**

To write, review, and finalize the complete research paper along with all supportive materials like outputs, and formatting.

**Tasks Performed:**

- Wrote all core sections: Abstract, Introduction, Literature Survey, etc.
- Inserted code snippets for fingerprinting, XSS, and clickjacking directly into the Implementation section.
- Created result tables showing detection rates, attack behaviour, and browser responses.
- Reviewed logs, daily activities, and formatted this weekly work log for the final report.
- Did a final proofread and formatting check (font, margins, citations, page numbers, table of contents).

**Tools and Resources Used:**

- MS Word (final write-up)
- Grammarly (proofreading)
- College report template (if provided)
- Screenshots and formatted tables

**Challenges:**

- Balancing technical detail with readable flow.
- Ensuring plagiarism-free writing and original analysis of findings.

**Outcome:**

- Final report completed, polished, and ready for submission.
- Weekly log compiled, bibliography completed, code inserted, and all sections reviewed.