# Analyzing Security Vulnerabilities in Mobile Browsers: Risks and Exploits

Jayaraj Patil
Department of BCA, Seshadripuram College
Seshadripuram College Karnataka, India
jayarajvp2005@gmail.com

*Abstract*— With the increasing use of mobile devices, mobile browsers have become prime targets for cyber threats such as phishing, clickjacking, hidden address bar manipulation via JavaScript injection, and browser fingerprinting. In this study, I analyse vulnerabilities in popular mobile browsers through penetration testing and case studies. The findings reveal significant security weaknesses, exposing users to various cyber risks. To mitigate these threats, I propose a security-enhancing solution designed to improve protection against emerging attacks, ensuring a safer mobile browsing experience.

## I. INTRODUCTION

The explosive development of mobile technology has revolutionized the way users access the internet. Mobile browsers are a portal to web sites, and thus they are a target for cyber-attacks. In contrast to the traditional desktop environment, mobile devices have special security challenges because of its limited resources, constant network switching, and user behavior patterns.

One of the key issues in mobile browser security is susceptibility to cyber threats like phishing, clickjacking, address bar manipulation through JavaScript injection, and browser fingerprinting Cyber attackers take advantage of these vulnerabilities to steal sensitive data, inject malicious code, or manipulate user data. Although contemporary browsers have security measures like sandboxing, HTTPS enforcement, and phishing protection, attackers constantly evolve new methods to circumvent these security measures.

This research paper investigates the security weaknesses of mobile browsers, with a focus on real-world attack demonstrations and countermeasures. The research seeks to examine actual threats, evaluate browser security features, and suggest improvements to counter possible risks. Through these issues, this research adds to the continuous effort to enhance cybersecurity awareness and create more secure mobile browsing environments.

## II. LITERATURE REVIEW

### Web Browser Forensics and Cyber Investigations

Web browsers are vital to contemporary digital activities, but they are also possible avenues of cybercrime entry. Chanda et al. (2024) highlight that browser forensic examinations need to investigate multiple artifacts such as browsing history, cache files, cookies, and autofill data to decipher user activity and possible security threats
. Their work assessed forensic programs on various browser modes—default, private, and portable—showing how, even in cleared browsing, remnants of users' activities tended to be obtainable using forensics.

The research also found inconsistencies in privacy promises on mainstream browsers like Google Chrome, Mozilla Firefox, Brave, Tor, and Microsoft Edge. Private and incognito modes purport to restrict data retention, yet forensic tools were still capable of retrieving pieces of visited URLs, cache information, and session information. This highlights the inadequacies of current privacy features, leaving users exposed to tracking and forensic recovery.

### Challenges in Private Browsing and Evidence Recovery

Alotibi et al. (2024) investigated the efficacy of forensic tools in recovering browser artifacts from regular and private browsing modes. Their study examined Mozilla Firefox, Safari, and Microsoft Edge, with an emphasis on the technical challenges private browsing presents to forensic investigations
. The research identified that although private browsing stops data from being recorded in traditional storage areas, forensic tools like Autopsy, AXIOM, and XRY managed to recover deleted search history, login information, and session artifacts in specific situations.

In addition, the research analyzed forensic acquisition methods for mobile browsers and desktop browsers and discovered that jailbroken or rooted devices yield far more forensic information compared to non-rooted devices
. This points to the danger of compromised mobile devices and the necessity for more robust security measures to avoid unauthorized forensic data extraction.

### Relevance to This Study

These researches present important information regarding browser security limitations and forensic analysis methods, strengthening the necessity of more robust privacy protection mechanisms. Based on this, this study performs penetration testing on mobile browsers to evaluate actual-world vulnerabilities, with emphasis on phishing attacks, JavaScript injections, and fingerprinting methods. By determining major security vulnerabilities, this research seeks to suggest countermeasures that will improve user security and privacy within mobile browsing scenarios.

## III. METHODOLOGY

This study employs penetration testing to examine the security vulnerabilities of mobile web browsers. The methodology is structured into four key phases: browser selection, attack scenario development and penetration testing.

### A. *Browser Selection*

To ensure comprehensive analysis, this study focuses on popular mobile browsers with significant market share. The selected browsers include:

- Chrome.
- Brave.
- Firefox.
- Edge.

### B. *Attack Scenario Development*

Three common attack vectors were selected to evaluate browser vulnerabilities:

- Phishing Attacks – Testing browsers ability to detect and block fraudulent websites.
- JavaScript Injection – Injecting scripts to manipulate page elements (hiding the address bar and XSS attacks)
- Fingerprinting Techniques – Analysing how much identifiable information browsers expose to tracking scripts.
- Clickjacking – Embedding hidden elements to trick users into interacting with malicious content.
- Use a zero before decimal points: “0.25”, not “.25”. Use “cm3”, not “cc”. (*bullet list*)

### C. *Penetration Testing Approach*

Penetration testing was conducted in a controlled environment such as virtual android.

- Phishing attack was performed by help of known phishing URL from PhishTank dataset.
- Clickjacking: CSS overlays were used to capture unauthorized clicks.
- XSS Attacks: Malicious JavaScript payloads were injected into web pages and tested on different browsers.
- Fingerprint analysis: Java script navigation is used.

### D. *Ethical Considerations*

All penetration tests were conducted in a secure, controlled environment without targeting real users.

## IV. RESULTS AND ANALYSIS

This section presents the findings from the penetration testing conducted on mobile web browsers. The results highlight vulnerabilities related to phishing, JavaScript-based attacks (Clickjacking, XSS, Address Bar Manipulation), and fingerprinting techniques. Each browser’s security performance is evaluated based on detection capabilities, susceptibility to attacks, and residual forensic artifacts.

### A. *Phishing Attack Detection*

What is a phishing attack. Phishing is a type of social engineering attack often used to steal user data, including login credentials and credit card numbers. It occurs when an attacker, masquerading as a trusted entity, dupes a victim into opening an email, instant message, or text message.

Experimental Setup

- Dataset Used: PhishTank (a public database of phishing websites).
- Test Environment: Virtual Android device.
- Browsers Tested: Google Chrome, Brave, Mozilla Firefox, Microsoft Edge.
- Procedure:
  - The same phishing URLs were accessed on all browsers.
  - Observations were made on whether the browser blocked the site, displayed a warning, or allowed access.

Results

| | **Browser name** | **Phishing Detection Rate (%)** |
|---|---|---|
| **1.** | Chrome | 40% |
| **2.** | Brave | 20% |
| **3.** | Edge | 50% |
| **4.** | Firefox | 40% |

Analysis and Key Findings

- Edge had highest detection rate of 50%.
- Brave performance was slight low with 20%.
- Chrome and Firefox both 40% detection rate.
- All browsers blocked http phishing links but when it came to https some missed.

### B. *Clickjacking*

Clickjacking is a web-based attack where an attacker tricks a user into clicking something different from what they perceive. This is done by overlaying transparent or disguised elements over a legitimate webpage, leading the victim to unknowingly perform unintended actions.

Experimental Setup

- Test Environment: Virtual Android device.
- Attack Method: A custom Clickjacking webpage was developed and deployed in a controlled environment.
- Browsers Tested: Google Chrome, Brave, Mozilla Firefox, Microsoft Edge.
- Procedure:
  - The webpage contained a hidden iFrame with an actual button underneath a fake UI element.
  - The page was accessed on all browsers, and observations were recorded on whether the browser prevented the attack or allowed the hidden click action.

Results

| | **Browser name** | **Clickjacking worked** |
|---|---|---|
| **1.** | Chrome | YES |
| **2.** | Brave | YES |
| **3.** | Edge | YES |
| **4.** | Firefox | YES |

Analysis and Key Findings

- All browsers were affected by clickjacking.

### C. *Cross-Site Scripting (XSS) Attack Experiment*

Cross-Site Scripting (XSS) is a type of web vulnerability where an attacker injects malicious JavaScript into a website. When executed in a victim's browser, this script can steal cookies, session tokens, or other sensitive data, manipulate web pages, or perform unauthorized actions on behalf of the user.

Experimental Setup

- Test Environment: Virtual Android device.
- Browsers Tested: Google Chrome, Mozilla Firefox, Microsoft Edge, Brave.
- Procedure:
  - A webpage was designed with an embedded malicious JavaScript payload that executes when loaded.
  - The script does the following:
    - Displays an alert box (alert("XSS Successful!")).
    - Attempts to steal cookies (document.cookie).
    - Modifies page content (document.body.innerHTML = "Hacked!").

Results

| | **Browser name** | **Executed Preloaded XSS?** | **Displayed Security Warning?** | **Prevented Cookie Theft?** |
|---|---|---|---|---|
| **1.** | Chrome | YES | NO | Same Site & HttpOnly Protected |
| **2.** | Brave | YES | Partial Warning | Blocked some, but DOM scripts still ran |
| **3.** | Edge | YES | NO | Blocked by SmartScreen |
| **4.** | Firefox | YES | NO | Protected by Cookie Policy |

Analysis and Key Findings

- All browsers executed the preloaded XSS script since the malicious JavaScript was part of the page's original code.
- Chrome, Firefox, and Edge blocked cookie theft attempt due to HttpOnly and SameSite cookie policies.
- Brave provided some additional security by blocking certain requests but still allowed DOM manipulations.

### D. *Browser Fingerprinting Experiment*

Browser fingerprinting is a tracking technique where websites collect unique attributes from a user's browser to identify them without using cookies. This includes:

- Device & OS details (Android, iOS, Windows, etc.)
- Browser type & version (Chrome, Firefox, Edge, Brave)
- Screen resolution & colour depth
- Installed fonts & plugins
- User-Agent string

Experimental Setup

- Test Method: JavaScript's navigator object was used to collect browser fingerprints.
- Test Environment: Virtual Android device.
- Browsers Tested: Google Chrome, Mozilla Firefox, Microsoft Edge, Brave.
- Data Collected:
  - User-Agent String
  - Platform & OS
  - Screen Resolution
  - Do Not Track (DNT) Status
  - Installed Plugins Count

Results

| | **Browser name** | **Fingerprint Score** | **Blocks JavaScript Fingerprinting?** |
|---|---|---|---|
| **1.** | Chrome | 92% | NO |
| **2.** | Brave | 45% | YES |
| **3.** | Edge | 88% | NO |
| **4.** | Firefox | 80% | PARTIAL |

Analysis and Key Findings

- Google Chrome and Edge were the most fingerprint able browsers, exposing nearly all data points.
- Firefox partially blocked fingerprinting, reducing uniqueness but still exposing screen resolution and platform data.
- Brave was the most privacy-focused, blocking most fingerprinting techniques and reducing trackability.

## V. PROPOSED SOLUTION

Based on the findings from the phishing, clickjacking, XSS, and fingerprinting experiments, a security-enhancing solution is proposed to improve mobile browser security by introducing:

- Per-Tab VPN – Each tab operates with its own VPN connection to enhance privacy and security.
- Fake Device Information – Virtual tabs provide randomized device details to prevent fingerprinting.
- AI-Powered Firewall – Analyzes network packets to detect and block malicious activity in real time.
- AI-Based Web Scanning – Scans a website before loading to detect phishing, malware, and other threats.

### A. *Per-Tab VPN*

Problem Identified:

- Traditional mobile browsers route all traffic through a single network connection, making tracking easier.
- VPNs improve privacy, but using the same VPN across all tabs still allows cross-tab tracking.

Proposed Solution:

- Each tab operates with its own independent VPN tunnel, preventing tracking across tabs.
- Dynamic VPN Switching: VPN connections rotate randomly to prevent network fingerprinting.

### B. *Fake Device Information*

Problem Identified:

- Websites track users by collecting device details (screen resolution, OS, browser type, hardware specs).
- Fingerprinting works even without cookies and remains a major privacy risk.

Proposed Solution:

- Virtual Tabs: Each new tab generates fake randomized device information to break tracking mechanisms.
- Dynamic User-Agent Spoofing to make every session appear as a different device.

### C. *AI-Powered Firewall*

Problem Identified:

- Traditional firewalls work only at the OS level and do not analyse individual browser traffic.
- Packet-based attacks (MITM, DNS hijacking, injection attacks) are undetectable by standard browsers.

Proposed Solution:

- AI-Powered Firewall at the browser level to analyze network packets in real time.
- Uses machine learning models to detect:
  - Anomalous traffic patterns
  - Unusual data requests (e.g., phishing data exfiltration)
  - Known malware signatures.

### D. *AI-Based Web Scanning*

Problem Identified:

- Current browsers only detect phishing/malware *after* a page has loaded.
- Users still interact with malicious content before warnings appear.

Proposed Solution:

- AI scans every website before allowing it to load, analyzing:
  - Visual layout for phishing attempts (overlapping login forms, fake URLs).
  - JavaScript behavior analysis (hidden redirects, XSS payloads).
  - Network requests to detect malware-hosting domains.

## VI. Conclusion and Future Work

### A. *Conclusion*

This research examined security vulnerabilities in mobile browsers, focusing on phishing, clickjacking, XSS attacks, and browser fingerprinting. Experimental results highlighted that:

- Phishing detection varies across browsers, with some failing to block known malicious sites.
- Clickjacking vulnerabilities exist due to weak enforcement of X-Frame-Options and Content Security Policy (CSP).
- XSS attacks remain a risk when browsers do not sanitize user input properly.
- Browser fingerprinting techniques effectively track users, even in private browsing mode.

To mitigate these threats, we proposed a security-enhancing solution consisting of:

- Per-Tab VPN, ensuring independent VPN tunnels for each tab to enhance privacy.
- Fake Device Information (Virtual Tabs), which randomizes device attributes to prevent fingerprinting.
- AI-Powered Firewall, capable of analyzing network packets in real time to detect threats.
- AI-Based Web Scanning, which scans websites before loading to prevent phishing and malware attacks.

### B. *Limitations of the Proposed Solution*

While the proposed system enhances browser security, it comes with certain downsides and challenges:

- High Resource Consumption
  - The AI-powered firewall and web scanner require significant processing power, potentially impacting battery life and device performance.
  - Running a per-tab VPN for each browsing session increases CPU and memory usage, making it difficult for low-end mobile devices to handle efficiently.
- Increased Network Overhead
  - Per-tab VPN connections generate multiple encrypted tunnels, consuming more bandwidth than a standard VPN.
  - This could lead to slower browsing speeds, especially on networks with limited bandwidth.
- Compatibility Challenges
  - Implementing fake device information requires overriding browser APIs, which may be restricted by some mobile platforms.
  - Some web applications may break if they rely on fingerprinting for security checks.
- Potential False Positives in AI Detection
  - The AI-based firewall and web scanner rely on machine learning models, which may occasionally misclassify legitimate websites as threats.
  - Overly strict fingerprinting defenses may block sites that require device identification for authentication.

### C. *Future Work*

To address these limitations, future research can explore:

- Optimization of AI Models for Efficiency
    - Developing lightweight AI models that consume fewer resources while maintaining high accuracy.
    - Implementing on-device AI processing to reduce cloud dependency and improve real-time threat detection.
- Improving Performance of Per-Tab VPN
    - Optimizing VPN protocols to reduce CPU and memory load.
    - Implementing a smart VPN switching system that enables/disables VPN tunnels based on security needs.
- Balancing Security and Usability
    - Allowing users to customize security levels based on their device capability and browsing habits.
    - Enhancing AI explainability to minimize false positives and ensure better decision-making.
- Expanding Platform Compatibility
    - Extending support for iOS browsers and custom Android ROMs.
    - Exploring desktop implementations for Windows, macOS, and Linux.

Future improvements should aim to strike a balance between security, performance, and usability, ensuring that the enhanced protections do not significantly impact browsing experience.

Final Thoughts

The proposed solution provides a robust defense against phishing, clickjacking, XSS, and fingerprinting but requires further optimizations to enhance efficiency and cross-platform compatibility. Future work should focus on reducing resource consumption while maintaining security effectiveness to make this system practical for real-world use.

## Acknowledgment

The author would like to express gratitude to the cybersecurity research community for providing open-source datasets and resources that contributed to this study.

## References

[1] R. R. Chand, N. A. Sharma, and M. A. Kabir, *Web Browser Forensics and Cyber Investigations*, 1st ed. Springer, 2024.

[2] A. Alotibi, M. Al-Dhaqm, and S. Sumithra, "Challenges in Private Browsing and Evidence Recovery," *Journal of Digital Forensics*, vol. 15, no. 2, pp. 45-60, May 2024.

[3] "PhishTank: Join the fight against phishing," OpenDNS, 2025. [Online]. Available: https://www.phishtank.com/. [Accessed: Feb. 11, 2025].

[4] "Bliss OS: Open-source Android for PC," BlissLabs, 2025. [Online]. Available: https://blissos.org/. [Accessed: Feb. 11, 2025].

[5] "Brave Browser: Privacy and Security," Brave Software, 2025. [Online]. Available: https://brave.com/privacy/. [Accessed: Feb. 11, 2025].

[6] "Google Chrome Privacy Whitepaper," Google, 2025. [Online]. Available: https://www.google.com/chrome/privacy/whitepaper.html. [Accessed: Feb. 11, 2025].

[7] "Mozilla Firefox Privacy Notice," Mozilla, 2025. [Online]. Available: https://www.mozilla.org/en-US/privacy/firefox/. [Accessed: Feb. 11, 2025].

[8] "Microsoft Edge Privacy Statement," Microsoft, 2025. [Online]. Available: https://privacy.microsoft.com/en-us/privacystatement. [Accessed: Feb. 11, 2025].