Optional Walled Gardens Through Progressive Web Apps

Shengdun Wang
cppfastio

ABSTRACT

Traditionally, desktop operating systems such as Windows, macOS, and Linux have allowed sideloading of applications and lacked app sandboxing, resulting in numerous security vulnerabilities, especially for non-tech-savvy users. In contrast, mobile platforms like Android and iOS employ app sandboxing and typically require apps to be installed through official app stores. However, these mobile platforms are often criticized for being overly restrictive, creating *walled garden* ecosystems. Despite these restrictions, many security threats, such as phishing websites, still persist. Furthermore, power users find these environments frustrating due to their inability to customize or operate outside the constraints imposed by the ecosystem.

With the emergence of Progressive Web Apps, this paper explores a novel security model called the S/P Login Toggle. This approach aims to segregate the use cases for power users and average users by leveraging the flexibility and security of PWAs. The paper provides a historical overview, compares the S/P Login Toggle with the security models of existing operating systems, and demonstrates its superiority in balancing usability and security for a diverse range of users.

1 INTRODUCTION

For much of computing history, end users have enjoyed the freedom to install whatever software they desire. This is a fundamental right for end users as they own their devices and should be able to use them as they see fit. However, with the massive adoption of the Internet and web browsers in the 1980s and 1990s, this freedom gradually became a significant security issue. The Web is inherently unsafe, particularly for those who are not tech-savvy. For example, users might open their browsers such as as Google Chrome, use Google Search, visit a malicious website, download malwares, and end up infecting their entire Windows operating system[29]. This can even affect other machines on the Internet. Even without downloading malwares, non-tech-savvy people can still fall victim to phishing[28], scareware, cryptojacking, and many other security threats. This issue persists today, so much so that nearly every Windows user relies on antivirus software, including Microsoft's own preinstalled Windows Defender. The demand for antivirus software led to incidents like the CrowdStrike outage in July 2024, which caused widespread blue screens of death across the globe. This disruption affected critical services, including airlines and medical equipment, forcing many to close temporarily[9].

In 2007, when Steve Jobs introduced the iPhone, the mobile phone ecosystem took a different direction. Users were no longer allowed to execute executables directly, as they could with traditional Windows, Linux, and MacOS systems. Software needed to be packaged as apps and run in app sandboxes. These types of operating systems are often referred to as *Walled Garden* operating systems. It also became increasingly difficult for users to install alternative operating systems on their phones, as phone vendors locked down the bootloader[26]. On iPhones, users could not even install third-party software[2]. This trend led to a monopoly, with Google and Apple effectively becoming the duopoly of apps. Both Apple and Google charge a fee 30% on app store sales, creating significant tension between developers and these tech giants[17, 21]. In Richard Stallman's and the Free Software Foundation's words: `tivoization`[39]. This situation has escalated to the point that governments worldwide are filing antitrust lawsuits against them[7, 15, 32]. For example, Apple has been declared a gatekeeper and is forced to allow sideloading in the European Union[7]. Despite this duopoly, users still face scams and other security threats[24, 56].

The monopolies of Google and Apple have had serious long-term consequences. With the failure of Microsoft's Windows Phone due to app gaps, Windows, as the most important platform for running critical business software, cannot implement similar restrictions. Developers do not create apps for the Microsoft Store as they do for the Google Play Store or Apple App Store. Microsoft offers Windows in S mode[43], but it lacks apps, making it less attractive compared to Android tablets or Chromebooks, which have a wider range of apps, better security, and are often much cheaper. In addition to the shortage of apps, Windows users dislike the restrictive nature of walled gardens[49, 52]. This preference stems from Windows' long-standing tradition of distributing applications through third-party sources, which has proven to be highly effective. This situation also does not help other operating systems such as Linux. It also leads to planned obsolescence, contributing to environmental pollution, as many Android phones from five years ago no longer receive updates from vendors[22]. The outdated Linux kernel on these devices makes them vulnerable to attacks[37].

Clearly, this is a textbook example of the contradictions between security and usability. If you prioritize absolute freedom, you end up with numerous security issues. On the other hand, if you prioritize absolute security, the device becomes practically useless. What is the point of using electronic devices like phones, PCs, or VR headsets when they are inherently less secure than not using them? This often becomes an excuse for big tech companies like Apple[2, 11, 13, 34] and Google to justify their monopolistic practices. Over time, users have been steadily losing control over their own hardware and software rights[6]. The question then arises: why can't we have both security and usability[20]?

Conference'17, July 2017, Washington, DC, USA
© 2025 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper, we will explore a new security model applicable to Windows and other operating systems. Windows serves as an excellent example because it remains a significant target for malware, although its market share is gradually being replaced by Android as more people use phones over PCs. Despite this shift, we still rely heavily on our computers, so improving Windows security is crucial. Unlike phones, which are heavily locked down as previously mentioned, computers offer more flexibility for tasks such as programming.

With the rise of Progressive Web Apps (PWAs)[16, 55], many websites now support this technology. PWAs can perform a variety of core Apps, including music player apps like Snae Player[36] and EPUB reader apps like Flow[14]. Many core on-line services also support PWAs, such as Microsoft Office, Onedrive, YouTube, TikTok, Spotify, Apple Music, Starbucks, Tinder, Instagram, Twitch, Overleaf, Battle.net, Facebook, X (Twitter), Bestbuy, Discord, Wikipedia, Reddit, LinkedIn, Uber, TradingView, Yahoo Finance, Chase, VSCode, Microsoft Copilot, ChatGPT, DeepSeek, Murlok.io, Pornhub, Tencent News, Taobao, RT, PressTV, and GitHub. Even virtual machine apps, such as virtual machines in Windows 95 using v86, support PWAs. Although many social media platforms often promote their own mobile applications and limit the functionality of their mobile PWAs, these PWAs still serve users effectively on a daily basis.

The Web browser naturally acts as a sandbox, greatly reducing security risks compared to native apps on Windows, as Windows Win32 binaries typically do not run in app sandboxes. Microsoft itself can provide a wide range of core services, including Microsoft Office, Teams, OneDrive, and Copilot, to fill that gap. Therefore, we can create an optional walled garden login toggle for non-tech-savvy people to use these devices through PWAs on Windows while preserving the versatility and openness of Windows as a platform.

1.1 S/P Toggle

In this paper, we propose a new login screen toggle for future Windows 12, known as the `S/P toggle`. Inspired by Microsoft's Windows S mode, this feature is no longer a separate Windows version but a login screen toggle for standard user mode on Windows Home and Professional editions. We refer to the traditional unrestricted Windows user mode as P mode, where P represents Power user mode. This toggle allows users to easily switch between simple user mode and Power user mode. With this new toggle, separate Windows S mode versions will be removed and will no longer exist. Existing devices still using Windows S mode versions will automatically be unlocked.

In simple user mode (`S mode`), users are restricted to a locked-down, Walled Garden Mode where everything is an App. In this mode, users are only allowed to install progressive web apps from the Microsoft Store and are NOT allowed to directly use any web browsers or search engines, including Google Chrome and Google Search. Even Microsoft's own Microsoft Edge browser and Bing is not directly accessible and can only be used for loading Progressive Web Apps. The Microsoft Store should provide installations for Progressive Web Apps and Chromium Web Extensions. Since Progressive Web Apps are SEO-friendly, a search engine still exists in this mode, but it will only search content within the installed Progressive Web Apps. Additionally, users can use generative AI chatbots like Microsoft Copilot, ChatGPT and DeepSeek PWAs for searching web content. The user interface should resemble those of Android and iOS devices, making it more touch-friendly.

In power user mode (`P mode`), Windows operates as it does today, allowing users to run `.exe` PE files, install third-party software or web browsers, use the Windows Subsystem for Linux, and access UEFI settings to install alternative operating systems like Linux or FreeBSD. P mode also includes features for managing S mode, such as sideloading Progressive Web Apps and Chromium Web Extensions, as well as configuring app permissions. While users can enable Microsoft Edge or install another sandboxed browser in S mode, the operating system provides warnings and encourages users to sideload websites as Progressive Web Apps instead of using web browsers directly, or remain in P mode for unrestricted functionality.

The S/P mode toggle is exclusively a login screen UI feature and does not affect other remote functionalities, such as SSH or remote desktop.

This `S/P` toggle can easily be adopted by other operating systems like Linux or macOS to enhance security, and by Android and iOS to improve usability. However, in this paper, we will primarily focus on Windows since it is well understood by most readers and its adoption is more practical in the near term.

2 BACKGROUND

2.1 `unsafe` keyword in C#/Rust

In programming languages like C# and Rust, pointer operations are explicitly marked as `unsafe`. This establishes a language-wide boundary to distinguish memory-unsafe code. It is a brilliant concept because it allows users to demonstrate that they can handle APIs correctly, reducing the likelihood of errors. This is especially valuable when working with third-party libraries. In contrast, while C++ employs RAII (Resource Acquisition Is Initialization) for resource management, it does not inherently ensure memory safety. This is because C++ libraries cannot guarantee that users will utilize them correctly 100% of the time.

The S/P toggle concept discussed in this paper draws inspiration from the `unsafe` keyword in C# and Rust. It aims to create a similar boundary, enabling users to safely and correctly operate their devices. At the same time, it ensures that advanced users retain the flexibility they need to perform tasks when necessary.

2.2 Google's Misplaced Blame on Memory-Unsafe Languages

Google Chromium has consistently identified memory-unsafe languages, such as C++, as a primary source of its security issues[38]. However, this paper contends that, based on this same logic, Google Chrome itself should be deemed fundamentally unsafe. The root cause of these issues extends beyond programming languages, stemming instead from the inherent design of web browsers. As of February 2025, Google Chrome held a dominant 66.3% share of the global market[1], with Chromium-based browsers collectively exceeding 80%. This extensive exposure to arbitrary and untrusted input highlights the fundamental weaknesses in browser architecture.

Every website accessed via a browser is an untrusted remote code execution. While memory-unsafe languages do contribute to some security problems, the larger issue lies in the browser's unavoidable handling of untrusted input, which makes it significantly more susceptible to threats compared to other applications. Following this logic, banning Google Chrome for security concerns for majority of users would align with the rationale used for criticizing C++. If web browsers never existed in the world, the majority of security issues would also disappear, as most devices would no longer face the risks associated with exploiting security vulnerabilities.

2.3 App Sandboxing

App sandboxing confines an application's access to system resources and user data within a restricted environment. Apple mandates that all apps in the App Store utilize app sandboxing[3]. In the Linux ecosystem, app sandboxing has also gained significant traction with technologies like Flatpak, Snap, and AppImage. Similarly, all Android apps are sandboxed by default. The primary advantage of app sandboxing is its ability to enforce app-specific permissions, limiting what each application can do.

However, the drawbacks are substantial. Applications often need to be rewritten to function within sandboxed environments, and this requirement extends to their third-party library dependencies. Many open-source libraries and software lack the necessary updates to adapt to these environments. Additionally, performance can suffer due to the sandbox's isolated nature, which restricts resource sharing. A notorious example is Firefox on Ubuntu, distributed as a Snap package. The Snap version of Firefox has been criticized for slow boot time—sometimes taking up to 20 seconds on an average computer—because it bundles its own version of glibc and other libraries instead of using libraries the distribution itself provides[8, 48].

These sandboxed apps can also create conflicts. Drivers bundled within Snap or Flatpak applications may not align with those provided by the operating system, leading to potential compatibility issues. Furthermore, while the app itself is sandboxed, it still operates within the same memory address space as the kernel. If the kernel has vulnerabilities, app sandboxing cannot prevent kernel-level exploits. This limitation is evident on Android, where malware remains a persistent issue despite widespread app sandboxing. The problem is particularly relevant for mainstream OS kernels like Windows NT, Linux, and Darwin, all of which employ monolithic kernels, not micro kernels.

2.4 Progressive Web Apps

Progressive Web Apps (PWAs) are essentially websites that function as applications. Since every platform includes browser support and mainstream web browsers embrace this technology, PWAs are highly versatile. For example, websites like Snaeplayer Music Player or Flow EPUB Reader can be accessed, added to your home screen on any device—be it a phone or PC—and effectively installed as apps. These apps operate using the browser engine already available on the device.

Unlike WebView apps or Electron apps, PWAs do not bundle their own Chromium installation. WebView apps, such as the pre-installed Microsoft Outlook on Windows 11, and Electron apps often duplicate Chromium, which can be resource-intensive. In contrast, PWAs share the same browser engine that the user is already utilizing.

Progressive Web Apps support offline functionality once installed. For instance, Snaeplayer works seamlessly without an internet connection. Furthermore, PWAs can access local files and support features like push notifications, provided the platform-specific apps allow it. They are truly cross-platform, adhering to open web standards, rather than being proprietary technology, making them a robust and inclusive solution for modern app experiences.

Since Progressive Web Apps operate directly within the browser, they inherently function within a sandboxed environment. This design prevents many direct exploits targeting OS APIs and the kernel, providing an additional layer of security compared to other technologies like app sandboxing.

2.5 WebAssembly

WebAssembly[18, 54] enables native languages like C and C++ to run on the web, achieving performance levels close to that of native applications. When combined with Progressive Web Apps, it simplifies the development of web-based applications using native languages, making the process more efficient and accessible.

2.6 Universal Windows Platform

Microsoft's earlier attempt to establish a walled garden ecosystem relied on its proprietary Universal Windows Platform (UWP) technology. However, Microsoft eventually abandoned UWP in favor of WinUI 3 and other frameworks. One of the key issues with UWP was its severe limitations, particularly due to the lack of a proper permission management system. For example, the developer of Rufus highlighted that UWP could not meet the requirements for accessing USB drives[51]. Even Microsoft's own documentation for Windows Terminal revealed challenges with UWP, indicating that it was unsuitable for implementation. Moreover, Microsoft's decision to restrict the use of C++ in UWP created additional hurdles, preventing apps like Spotify from porting their cross-platform core engines, which are built in C++ and Assembly, to the UWP ecosystem[12].

Although UWP was marketed as a cross-platform solution, developers seeking true cross-platform compatibility require a "universal platform" rather than a "universal Windows platform." They aim to develop for all major operating systems—such as Android, iOS, Linux, and macOS—beyond the Microsoft ecosystem. Progressive Web Apps have proven to be a superior alternative, offering all the capabilities UWP was meant to provide, while adhering to open web standards and supporting seamless functionality across platforms.

2.7 App Store and Sideloading

App Stores are now a standard feature across all major operating systems, including Linux GNOME, which offers Flatpak packages. As software distribution channels, App Stores, when properly managed, enhance security by having reviewers evaluate apps before their release—making them more secure than downloading random apps from the internet. However, the significant 30% revenue cut imposed by platforms like the Apple App Store and Google Play Store remains a widely criticized drawback. Additionally, some apps are either unavailable on these stores or are heavily modified to meet store requirements.

Microsoft is notably more accommodating toward Progressive Web Apps compared to Apple and Google, as they allow PWAs to be distributed through the Microsoft Store. Interestingly, when Steve Jobs unveiled the first-generation iPhone, there was no App Store; instead, he advocated for web apps as the primary application model. Within the framework of this paper, PWA apps should ideally be distributed via a public channel, such as a GitHub repository, while ensuring that a dedicated PWA App Store focuses exclusively on installing PWAs.

2.8 Antimalware Solutions, Including Microsoft Defender SmartScreen

Microsoft offers its own anti-malware solution integrated into the Microsoft Edge browser, such as Microsoft Defender SmartScreen, to guard against phishing attacks, malware, and downloading malicious files from the internet. However, much like C++ address sanitizers, these solutions cannot provide absolute guarantees. While they can help detect malicious actors, they cannot completely eliminate security threats. The persistence of malware issues on Windows demonstrates this limitation, despite Microsoft's deployment of multiple layers of security measures, including SmartScreen, Windows Defender, and Windows Firewall.

Third-party anti-malware solutions also attempt to address these challenges, but they too have their own shortcomings. Notable incidents, such as the CrowdStrike outage, underscore the limitations of antivirus solutions across platforms, including Windows, Android, and macOS. These solutions often fail to completely prevent security threats and can sometimes introduce additional problems, as seen in cases like the aforementioned CrowdStrike outage. This highlights the inherent weaknesses in traditional antivirus approaches, especially in complex and evolving threat environments.

3 THREAT MODEL

We consider a scenario where users lack technical expertise and may unknowingly engage in risky online behaviors. A typical situation involves users browsing the web with Google Chrome, performing Google Searches, or mistyping URLs, which could inadvertently lead them to malicious or untrusted websites. Upon accessing these sites, users face multiple threats from malicious actors. These attackers may exploit vulnerabilities through techniques like side-channel attacks using JavaScript or WebAssembly, deliver cryptojacking, scareware[47], or leverage flaws in the browser's security.

Additionally, users can be tricked into downloading malicious applications designed to extract sensitive data, such as credit card information. Once this occurs, the attackers gain significant control, enabling them to execute harmful actions without restriction. Figure 1 illustrates the threat model, capturing the various attack vectors and their potential impacts.

4 DESIGN

The purpose of the S/P mode is to create a distinct separation between simple users and power users. This section details the design of the S/P mode toggle. Each Windows user account would offer the S/P mode toggle directly on the login screen. This toggle does not alter the existing permission management systems of Windows or Unix-based operating systems.

Figure 2 presents a prototype of the proposed login screen UI incorporating the S/P mode toggle. The key modification compared to the standard Windows login screen is the addition of a new button located on the right side of the existing login options. This button allows users to directly select either S Mode or P Mode from the login interface. The system remembers the user's previous choice: if S Mode was selected during the last session, it will default to S Mode; likewise, if P Mode was chosen, it will default to P Mode unless the user opts to switch. Notably, this enhancement is strictly a UI-level adjustment and does not alter the underlying functionality for user accounts, whether they are administrators, sudo users, or regular users. Instead, it introduces a secure and intuitive interface to complement the existing account structure without changing the way users interact with the system. When the user selects S Mode, the operating system does not launch the standard desktop UI. Instead, it uses Microsoft Edge as the engine for running PWAs and presents a UI similar to iPhone or Android, where users can access and launch PWAs in an intuitive, app-like interface. This design enhances accessibility while maintaining the simplicity of the PWA-focused environment.

4.1 S Mode

Figure 3 illustrates the architecture diagram for the S Mode toggle, while Figure 4 depicts the scope of the PWA Walled Garden. S Mode is specifically designed for users who focus on essential tasks such as working with Word, Excel, or engaging with social media apps like YouTube and Instagram, supported through progressive web apps. Advanced functionalities intended for power users are omitted in this environment. The primary goal of S Mode is to implement stringent controls, with restrictions surpassing those enforced on iPhones in numerous settings.

4.1.1 Disallow Direct Browser Use. Browsers are considered unsafe and are strictly prohibited for direct use. They are only permitted for hosting PWAs. No application, including Microsoft Edge or third-party browsers like Google Chrome and Firefox, should provide standalone browser functionalities.

4.1.2 Disallow Direct Search Engine Use. Search engines are also regarded as unsafe and are disallowed in S Mode. This restriction helps prevent phishing attempts, blocks access to fake websites, and mitigates the growing issue of SEO pollution[5]. Users are limited to searching only within the websites of installed PWAs, and are encouraged to rely on AI-powered chatbots, such as Microsoft Copilot, ChatGPT, or Deepseek, for accessing web content securely.

4.1.3 Browser Running Within the App Sandbox in S Mode. In S Mode, the browser application itself should operate within the App Sandbox[4] to add an additional layer of defense. This setup ensures that PWAs are not only sandboxed by the browser but also benefit from the security of the App Sandbox encapsulating the browser. This dual-layer protection strengthens security by combining the isolated environments of both the browser and the App Sandbox.

4.1.4 All Progressive Web Apps Must Be Installed via the App Store. Progressive web apps are installed in a manner similar to

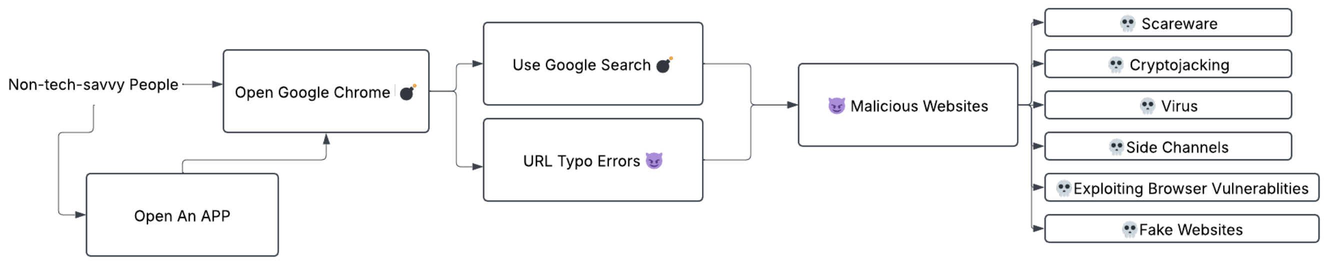

Figure 1: Threat Model: Browsers and Search Engines Considered Unsafe

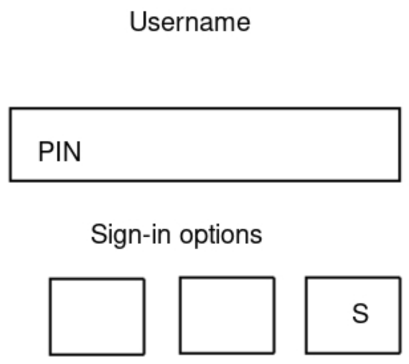

Figure 2: Demo of the S/P Mode Toggle Login Screen

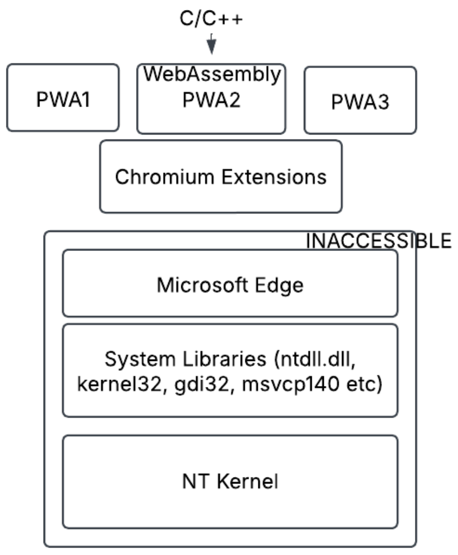

Figure 3: S Mode Toggle Architecture (NOT Windows S mode)

traditional apps on platforms like Google Play Store or Apple App Store. However, instead of native apps, users install PWAs. These apps run internally using Microsoft Edge, but direct access to the browser itself is restricted.

4.1.5 Browser Extensions are in the App Store. Since PWAs operate within browsers, they often leverage browser extensions. For example, users might install ad blockers for YouTube PWAs or use global speed/pitch adjusters to modify playback speeds for apps like Spotify, transforming them into nightcore players. To maintain control, such extensions should be made available only through the App Store.

4.1.6 Restrict Direct URL Access. Allowing users to type URLs poses risks, as typographical errors can lead to security issues. Hence, direct URL access is prohibited in all forms within S Mode.

4.1.7 PWA App Store Must Be Open. Progressive Web Apps are web resources and should remain open to ensure accessibility and transparency. Developers should have the ability to submit their PWAs to a publicly available GitHub repository, akin to how Microsoft vcpkg operates, or alternatively, rely on web standards to oversee PWA App Stores. Furthermore, no commission fees should be allowed for app submissions. This openness enables other operating systems, such as Linux and macOS, to adopt and implement this security model effectively.

4.1.8 Limit External Links. Web pages often include external links that could redirect users to malicious websites. To address this, PWAs must include a manifest file specifying a whitelist of allowed external links. By default, login pages for services like Google, Apple, or Microsoft accounts are permitted unless explicitly restricted by the app.

4.1.9 Restrict Permission Changes. Modern apps frequently request excessive permissions to exploit user data. In S Mode, users are prohibited from modifying app permissions to maintain data security.

4.1.10 Disallow Sideloading. Sideloading of both PWAs and browser extensions is strictly prohibited in S Mode.

4.1.11 Restrict Basic Settings Changes. Even seemingly simple settings, like date and time or timezone configurations, can be problematic for users. Incorrect settings often disrupt app functionality, such as the Google Play Store. To reduce complexity and maintenance needs, these settings cannot be altered in S Mode.

4.1.12 Restrict File Access to `$HOME/.smode` Directory. Complete restriction of file access is impractical, yet unrestricted access poses significant risks, as demonstrated by historical issues with Windows Phone. In S Mode, file access is limited to the `$HOME/.smode` directory (%USERPROFILE%/.smode on Windows), ensuring essential functionality while minimizing potential vulnerabilities. Additionally, users can synchronize their data seamlessly with Microsoft

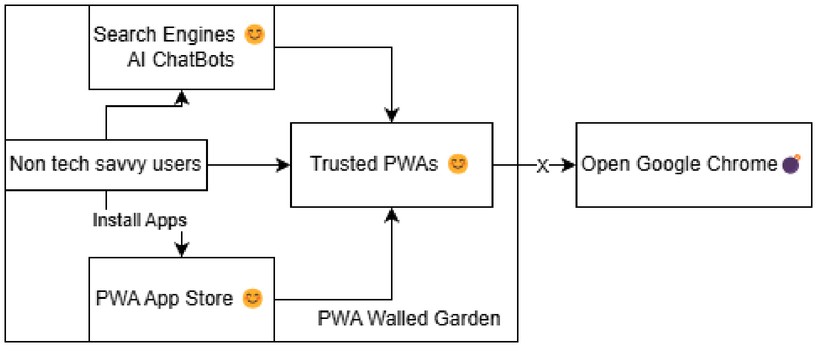

Figure 4: PWA Walled Garden in S Mode Toggle (NOT Windows S mode)

OneDrive within S Mode. However, apps should NEVER be allowed to access Microsoft OneDrive directly to prevent ransomware.

4.1.13 App Transitions Restricted to Installed PWAs. If one PWA attempts to open another PWA, the transition must be limited to installed PWAs. A browser popup should appear to handle the transition, but if the target PWA is not installed, the app transition should be blocked.

4.1.14 Gradually Implement Enhanced Security Measures for PWAs. Progressive Web Apps should gradually adopt stronger security measures to improve their robustness. For instance, WebAssembly binaries—often compiled from memory-unsafe languages like C/C++—require additional mitigations, such as WebAssembly Memory Tagging[53], to ensure safer execution. S Mode offers an opportunity to enforce stricter security measures for PWAs. However, as most WebAssembly applications currently lack adequate mitigations, this process must be introduced incrementally to allow for widespread adoption and compatibility.

4.1.15 S Mode UI Should Closely Resemble iPhone's UI. The iPhone's UI has become the de facto standard for mobile phone interfaces and walled garden operating systems. As mobile UIs are the most familiar to a majority of end users, adopting a similar design for S Mode ensures ease of use and eliminates any learning curve for users.

4.1.16 Automatic PWA Synchronization. Currently, PWAs are synchronized through Microsoft Edge, making them easily accessible for end users. To enhance usability in S Mode, these PWAs should be automatically synced, ensuring a seamless experience for users.

4.1.17 Integration with Trusted Platform Module for Sensitive Apps. Sensitive PWAs, such as the Chase Bank App, can enhance security by leveraging the Trusted Platform Module[57] to verify the integrity of both the Edge browser engine and the banking application's code. This ensures that sensitive operations are executed within the app's sandbox, with their integrity fully guaranteed.

4.2 P Mode

P Mode represents the Windows environment in its standard form. It allows users to perform tasks such as running `.exe` files, utilizing browsers, accessing the Windows Subsystem for Linux, using `cmd` and PowerShell, editing the registry (`regedit`), and rebooting into UEFI to install other operating systems. Logging into P Mode signifies that the user is a power user, comparable to utilizing `unsafe` blocks in programming languages like C# or Rust. It conveys to the operating system: "I understand the risks and am fully capable of managing them. Please grant me unrestricted access to perform my tasks."

With the introduction of the S/P mode toggle, P Mode also takes on the responsibility of managing settings for S Mode. Below, we present the design principles for P Mode.

The Windows Settings app in P Mode should include a dedicated section for managing S Mode settings.

4.2.1 Power User Mode Must Grant Full Control, Including Installation of Other Operating Systems. P Mode is designed for power users and is not intended to be a locked-down environment. Power users should not face the restrictions common to ecosystems like Android or iOS. Instead, P Mode must allow users complete freedom, including the ability to install alternative operating systems.

4.2.2 Enable Comprehensive Settings Management. While S Mode restricts changes to date, time, and timezone settings, P Mode must provide full access to these configurations. This highlights the versatility of the S/P mode toggle compared to fully locked platforms like iPhones, where errors in settings can arise due to the absence of boundaries for unsafe operations. In the pursuit of complete safety, unintended vulnerabilities may emerge—this toggle offers a balance.

4.2.3 Allow Sideloading of PWAs and Browser Extensions into S Mode. Since PWAs are essentially websites, sideloading them into S Mode should be straightforward—users can simply input URLs to initiate the process. Additionally, the Edge browser in P Mode can feature an option that allows users to install PWAs and Chromium extensions directly into S Mode.

4.2.4 Centralized Permission Management for PWAs in S Mode. Permissions for PWAs in S Mode should be managed exclusively through P Mode. Android apps often manipulate users into granting unnecessary permissions. By centralizing permission management in P Mode, this approach protects non-tech-savvy users from being tricked by applications while retaining full functionality for power users.

4.2.5 Adjust PWA Synchronization Settings. P Mode should offer users the flexibility to customize how PWAs are synchronized for S Mode. This ensures personalized and efficient use of PWAs.

4.2.6 *Modify File Access Settings for S Mode.*

By default, file access in S Mode is restricted to `$HOME/.smode` (On Windows it is `%USERPROFILE%/.smode`) directory. In P Mode, users should be able to expand access by adding more directories or modifying the default directory that S Mode can access.

4.2.7 *Dedicated File Storage for Each PWA.*

In addition to shared storage for Progressive Web Apps, each PWA can maintain its own dedicated file storage. This approach enhances sandboxing capabilities in S Mode, ensuring greater isolation and security for individual app.

4.2.8 *Popup Suggestions for Entering S Mode in Sensitive Apps.*

Sensitive applications, such as banking PWAs like Chase, require heightened security measures. In this context, browsers operating in power user mode should provide a discreet suggestion prompting users to switch to S Mode for enhanced protection. To ensure a seamless user experience, these prompts should be limited in frequency and appear only for sensitive applications. Unless mandated by a bank's policy to enforce the use of S Mode, these suggestions should remain optional, allowing users to make the choice without feeling overwhelmed by intrusive notifications.

4.2.9 *Gradually Expand Options for Modifying S Mode Settings.*

The aforementioned list outlines initial options for modifying S Mode settings. Over time, users may require additional flexibility, such as changing the browser engine used for PWAs instead of relying solely on Microsoft Edge. However, this introduces potential security risks—if the browser engine originates from a malicious source, it could compromise the entire system, as illustrated by scenarios like using a Chase Bank PWA. To mitigate these risks, modifications should only be allowed through the App Store in P Mode or by requiring cryptographic signatures provided by Microsoft to verify the integrity of the browser engine. These changes must be implemented with caution and gradually.

Additional options, such as enabling the sideloading of Win32 sandbox applications into S Mode, could pose significant security risks. Unfortunately, many apps vendors—such as WeChat—choose to monetize user data rather than offering PWAs. To ensure the security and integrity of S Mode, these features must be implemented thoughtfully and incrementally. Nonetheless, Progressive Web Apps should remain the preferred method for app delivery over Win32 sandbox applications, as PWAs inherently add an extra layer of sandboxing through the browser engine.

5 EVALUATION

This is a position paper, so direct evaluation through the implementation of this feature and subsequent data collection on its impact on the Windows security ecosystem is not feasible.

However, extensive evaluations can still be conducted indirectly. For example, comparisons can be made regarding security, performance, and usability relative to existing operating system modes. Other related studies provide valuable data that support and demonstrate the effectiveness.

5.1 App Size

Progressive Web Apps offer a significant advantage in terms of their smaller binary size compared to Android and iOS applications, a fact supported by various studies and statistics. This advantage stems from two primary factors. First, the browser itself provides numerous built-in functionalities, eliminating the need for individual apps to duplicate code to implement those features. Second, and more importantly, PWAs are essentially websites or web pages. As a result, browsers only download the specific use-case a user requires, unlike Android or iOS apps, which often include features that the user never uses[50].

The Web Is Dead! Long Live The Web! [50] by Sam Thorogood highlighted a notable comparison in 2018: the Twitter app on iOS required 186MB, on Android 70MB, while the Twitter Progressive Web App was under 2MB, demonstrating the significant size advantage of PWAs.

The benefits of Progressive Web Apps extend beyond Twitter. According to Microsoft's documentation[45], Tinder experienced remarkable improvements with its PWA, reducing load times from 11.91 seconds to 4.68 seconds. Additionally, the PWA's size is 90% smaller compared to the compiled Android app, showcasing its efficiency.

PWAs are consistently smaller in size compared to Android and iOS apps. While certain applications, such as games, may have smaller native versions, developers still have the option to create Win32 sandbox apps for S Mode. Additionally, P Mode remains available, allowing users to access traditional Win32 applications as needed.

5.2 Security Evaluation

Table 1 highlights the security measures of the S/P Mode Toggle compared to other operating systems. The S/P Mode Toggle demonstrates notable security advantages over Android and iOS. Under S Mode, the system is significantly more restricted, with features such as the prohibition of direct browser use and the unavailability of numerous settings, ensuring a more locked-down environment. While P Mode is less secure, S Mode offers a reliable solution for non-tech-savvy users or professionals in corporate settings who primarily perform lightweight tasks such as working with Word, Excel, or PowerPoint. For these users, adhering to S Mode provides a strong assurance of a trouble-free experience without the need to transition to P Mode.

S mode toggle uses Progressive Web Apps which are inherently sandboxed by the browser—a system designed as a natural sandbox. This structure offers a higher level of security compared to iOS apps. Furthermore, the browser operates as a WebAssembly virtual machine, allowing memory-unsafe C/C++ applications to run securely within the browser environment. Advanced security mitigations, such as *WebAssembly Memory Tagging*[53], enhance protection against vulnerabilities. Implementing such mitigations is significantly more challenging with native applications on Android and iOS.

Another significant advantage is the ease of upgrading every component of the system. This includes the app itself, the browser, and the operating system. When any of these components are patched, the system becomes inherently safer. Additionally, since Progressive Web Apps are fundamentally websites, addressing vulnerabilities or implementing feature updates is significantly faster compared to native apps. Moreover, updates for PWAs do not require app store approval, streamlining the process even further.

The S Mode toggle also minimizes the risk of SEO pollution by restricting search engines to only retrieve information from trusted apps. This feature offers a distinct advantage over Android and iOS, as their apps are not inherently SEO-friendly, making such a controlled search experience unattainable. As a result, Android and iOS systems expose non-tech-savvy users to a higher risk of encountering fraudulent content, whereas S Mode provides enhanced protection.

5.3 Usability Evaluation

Table 2 provides a usability comparison table. When evaluated alongside Android and iOS, the S/P Mode toggle showcases significant usability advantages. As a simple UI feature integrated into the login screen, the S/P Mode toggle does not alter the way the operating system functions. Users retain the ability to replace their operating systems seamlessly, as they currently can, which is not easily achievable on Android or iOS. Moreover, in P Mode, users can run Win32 .exe PE executables—a feature that is unavailable on iOS. While Android permits some degree of this functionality through Termux, the operating system itself imposes limitations, and running graphical binaries is considerably more challenging compared to Windows.

Additionally, the S/P Mode toggle enhances usability for non-tech-savvy users by providing a secure and simplified experience in S Mode. This eliminates concerns about malware or complex system details, giving these users the opportunity to focus on learning to use a computer safely before transitioning to P Mode.

Chromium extensions significantly enhance usability compared to Android and iOS apps in our S mode toggle. Chromium-based extensions enable extensive customization options for apps, far surpassing the capabilities of Android and iOS. Additionally, browser functionalities on Android are more limited than those on Windows; for instance, Google Chrome on Android lacks support for extensions. While Microsoft Edge now offers extension support on Android, this feature remains unavailable on iOS, further highlighting the advantages of browser-based customization in S Mode.

The developer experience in our S/P Mode far surpasses that of proprietary software development on platforms like Android and iOS. The web operates as an open standard, enabling developers to compile C/C++ code into WebAssembly and run it seamlessly across all platforms. In contrast, Android and iOS impose restrictions on the programming languages developers can easily use and require separate development efforts for each platform. With S/P Mode, developers benefit from using PWAs in S Mode or Win32 APIs in P Mode—both of which are well-established, widely understood, and fully supported by mainstream platforms (Win32 APIs are compatible through wine).

From a usability perspective, comparing our S/P Mode toggle to Windows S Mode reveals significant limitations in the latter. Windows S Mode lacks apps, and once users switch out of it, there is no option to revert back. Furthermore, it fails to protect users from scams originating from Microsoft Edge and Bing. Additionally, it offers no safeguards for power users, as they cannot utilize Windows S Mode effectively. It also does not prevent users from accidentally altering critical settings like date and time. In contrast, our S/P Mode toggle stands out with its high level of customization, making it versatile and suitable for both simple users and power users, addressing a broad range of use cases with tailored solutions.

5.4 Performance evaluation

Table 3 presents the performance evaluation of the S/P Mode. Compared to traditional Windows setups, one notable advantage is that PWAs, being web pages, benefit from Microsoft Edge's smart sleep functionality, which is the most power efficient compared to other browsers on Windows[44]. Microsoft Edge puts background web pages into sleep mode[27, 46]—a feature not feasible for Win32 applications, as standard C/C++ binaries lack built-in support for app suspension. Consequently, the OS kernel must save the entire state when the user closes the lid. When users logins with the S Mode instead of P Mode, the system becomes significantly lightweight as it exclusively runs Progressive Web Apps. This design enhances Windows' power efficiency for lightweight tasks, surpassing the power efficiency of traditional Win32 configurations.

In contrast to Android and iOS, PWAs in S Mode utilize the shared Edge browser engine, while P Mode takes advantage of shared Windows DLLs across various applications, as it functions like standard Windows. This architectural design ensures performance benefits optimized for each mode. Additionally, S/P Mode offers superior resource sharing in both S and P modes, unlike Android and iOS apps, where each app operates within a separate app sandbox.

5.5 Deployment Evaluation

Table 4 presents a comparison between the S/P Mode toggle and Windows Group policies. Unlike traditional tools such as Windows administrative group policies, the S/P Mode toggle is designed with simplicity in mind, making it highly accessible to average users. By merely switching to S Mode, users can enable a secure, locked-down walled garden environment without requiring additional configurations. While group policies can achieve comparable results, their complexity necessitates an IT administrator, posing challenges for non-tech-savvy individuals who purchase computers from retail stores like Walmart or Best Buy. The S/P Mode toggle eliminates this barrier, offering an intuitive solution that ensures system security with minimal effort.

This model offers additional deployment advantages, as it is compatible with all existing operating systems, including Linux and macOS. Users of Linux, macOS, or even future operating systems can seamlessly benefit from the S/P Mode toggle, enjoying both enhanced security and improved usability without sacrificing any current functionalities. Progressive Web Apps seamlessly operate on Linux today without any complications. This is because the web serves as a universal platform that is inherently platform-agnostic, ensuring compatibility across various operating systems. Moreover, the simplicity of this model allows individual users to deploy it independently, eliminating the need to rely on an IT professional for setup.

Table 1: Comparison of Security Across S/P Mode, Android, iOS, and Windows.

Security	S/P Mode	Android	iOS	Windows
User Mode Separation	✓ (S Mode for basic users, P Mode for power users)	✗ (No separation)	✗ (No separation)	✓ (Admin/User accounts)
App Restrictions	✓ (Only via official store, no sideloading)	✗ (Sideloading supported)	✓ (Only via App Store)	✗ (Manual installation allowed)
Browser Restrictions	✓ (Browser and search disabled, PWA-only access)	✗ (Full browser access)	✗ (Full browser access)	✗ (Full browser access)
File Access Restrictions	✓ (Restricted to `$HOME/.smode` directory)	✗ (Varies by device)	✓ (Strict sandboxing)	✗ (Full file access)
Permission Management	✓ (Centralized in P Mode)	✗ (Managed per app)	✓ (Strict prompts)	✗ (Scattered permissions)
Update Flexibility	✓ (Independent OS and browser updates)	✗ (Highly fragmented)	✓ (Unified updates)	✓ (Separate OS/browser updates)
Operating System Updates	✓ (Frequent updates with patches and features)	✗ (Delayed, depends on vendors)	✓ (Regular and unified)	✓ (User-controlled update strategies)
Sideloading Restrictions	✓ (Prohibited in S Mode, allowed in P Mode)	✗ (Widely supports sideloading)	✓ (Strictly prohibited)	✗ (Fully supports sideloading)
Search Engine Restrictions	✓ (Search limited to installed PWA content)	✗ (Unlimited search)	✗ (Unlimited search)	✗ (Unlimited search)
Browser Security Updates	✓ (Frequent updates for enhanced protection)	✓ (Depends on the user)	✓ (Unified control)	✓ (User-controlled updates)
Setting Restrictions	✓ (Time and timezone changes disabled)	✗ (Fully adjustable)	✓ (Partial restrictions)	✗ (No unified restrictions)
App Sandbox Restrictions	✓ (Each app runs in its own sandbox)	✗ (Inconsistent sandboxing)	✓ (Strict sandboxing)	✗ (Some apps lack sandbox)
Prohibit Executable Files	✓ (Blocked in S Mode, allowed in P Mode)	✓ (Executables not supported)	✓ (Strictly prohibited)	✗ (Executable files allowed)
Ban on Untrusted External Links	✓ (Only allows links to installed PWAs; user confirmation required)	✗ (Unrestricted external links)	✗ (Unrestricted external links)	✗ (Unrestricted external links)
Side-Channel Mitigations	✓ (PWAs only from trusted sources; no side-channel risks)	✗ (Susceptible to side-channel risks)	✗ (Partial mitigations)	✗ (Varies by app/system)
Cryptojacking Protection	✓ (No mining due to PWA restrictions)	✗ (Potentially vulnerable)	✗ (Open via web browsers)	✗ (Depends on installed apps)
Scareware Protection	✓ (PWAs vetted through trusted sources)	✗ (Vulnerable due to open ecosystem)	✗ (Open via web browsers)	✗ (Users must verify software)
Memory Safety Mitigations	✓ (Incremental measures like WebAssembly Memory Tagging)	✗ (Highly dependent on OS/app implementation)	✗ (Potential kernel exploits from buggy apps)	✗ (Memory safety depends on developer practices)

6 RELATED WORK

Dynamic Malware Analysis in the Modern Era–A State of the Art Survey[33], authored by Or-Meir, Ori, Nissim, Nir, Elovici, Yuval, and Rokach, Lior, provides a definition of malware and categorizes it into various types, including viruses, trojans, spyware, worms, adware, scareware, bots, ransomware, and cryptominers. The paper also classifies malicious behaviors associated with malware. It offers a comprehensive survey evaluating the strengths and weaknesses of different analysis methods and their resilience against malware evasion techniques. In addition, the authors highlight how mainstream approaches predominantly rely on machine learning for malware detection. The paper *Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection* by Firdausi, Ivan, Lim, Charles, Erwin, Alva, and Nugroho, Anto Satriyo employed the J48 decision tree algorithm, achieving a recall of 95.9%, a false positive rate of 2.4%, a precision of 97.3%, and an accuracy of 96.8%.

Table 2: Usability Comparison: S/P Mode, Android, iOS, Windows S Mode, and Standard Windows

Usability	**S/P Mode**	**Android**	**iOS**	**Windows S Mode**	**Windows**
Operating System Replacement Support	✓ (Full control, OS replacement possible)	✗ (Hardware restrictions, nearly impossible)	✗ (Completely closed, no OS replacement)	✗ (Requires exiting S Mode)	✓ (Open but limited by hardware compatibility)
Mode Switching	✓ (Seamless switching between S and P Mode)	✗ (No mode switching)	✗ (No mode switching)	✗ (Cannot switch without exiting S Mode)	✓ (Supports admin and user switching)
Application Installation Options	✓ (Supports official store, PWA, and sideload in P Mode)	✓ (Supports store, APK, and sideloading)	✗ (Restricted to App Store apps)	✗ (Restricted to Microsoft Store apps)	✓ (Supports store and sideloading)
File Access	✓ (S Mode restricts directories, P Mode allows full access)	✓ (Supports sandbox and external access)	✗ (Strict sandbox restrictions)	✗ (Limited access to system directories)	✓ (Full file access)
Web Browsing Support	✓ (S Mode disables browser, P Mode supports full browser)	✓ (Powerful, open browser support)	✓ (Browser supported with limitations)	✗ (Restricted to Microsoft Edge)	✓ (Full browser functionality supported)
Settings Flexibility	✓ (S Mode restricts critical settings, P Mode allows customization)	✓ (Full user control)	✗ (Partially restricted)	✗ (Strict restrictions on changing settings)	✓ (Complete customization freedom)
Application Exclusivity	✓ (PWA in S Mode, compatible across platforms)	✗ (Applications limited to Android devices)	✗ (Applications limited to iOS devices)	✗ (Applications limited to Windows environment)	✓ (Supports applications across environments)
Application Update Method	✓ (Developers update directly, no app store required)	✗ (Dependent on app store updates)	✗ (Must go through App Store)	✗ (Requires Microsoft Store for updates)	✓ (Supports direct updates by developers)
Programming Language Support	✓ (Supports WebAssembly, compatible with C++)	✗ (Primarily Java/Kotlin)	✗ (Primarily Objective-C/Swift)	✗ (Limited frameworks and language support)	✓ (Supports diverse languages including C++)

Malware Classification with Recurrent Networks[35] by Pascanu, Razvan, Stokes, Jack W., Sanossian, Hermineh, Marinescu, Mady and Thomas, Anil explored malware detection using a recurrent bidirectional neural network to analyze system calls, with their best model yielding a precision of 82% and a false positive rate of 0.5% based on 114 distinct system calls from 250,000 malware samples. *A Machine-Learning Approach for Classifying and Categorizing Android Sources and Sinks*[41] by Rasthofer, Siegfried, Arzt, Steven, and Bodden, Eric proposed the SUSI framework, evaluated on 11,000 malware samples, with their top-performing model achieving an accuracy of 92.3%. *Adaptive Detection of Covert Communication in HTTP Requests*[42] by Schwenk, Guido and Rieck, Konrad analyzed 695 malware samples but did not report metrics such as accuracy, precision, or false positive rate. *Trusted System-Calls Analysis Methodology Aimed at Detection of Compromised Virtual Machines Using Sequential Mining*[31] by Nissim, Nir, Lapidot, Yuval, Cohen, Aviad and Elovici, Yuval utilized memory dumps from virtual machines and reconstructed systems using WinDbg; their analysis of ten malware and six benign samples with random forest achieved a precision rate of 98% with no false positives. *Machine Learning in Side-Channel Analysis: A First Study*[23] by Hospodar, Gabriel, Gierlichs, Benedikt, De Mulder, Elke, Verbauwhede, Ingrid and Vandewalle, Joos demonstrated that machine learning could effectively extract cryptographic keys using power consumption traces as features, with their LS-SVM linear classifier reaching a success rate of 75%. *On the Feasibility of Online Malware Detection with Performance Counters*[10] by Demme, John, Maycock, Matthew, Schmitz, Jared, Tang, Adrian, Waksman, Adam, Sethumadhavan, Simha, and Stolfo, Salvatore tested their decision tree-based implementation on Android and Linux, using 201 benign samples and 503 malware

Table 3: Performance Comparison: S/P Mode, Android, iOS, Windows S Mode, and Standard Windows

Performance	**S/P Mode**	**Android**	**iOS**	**Windows S Mode**	**Windows**
Resource Usage	✓ (PWA is lightweight, downloads only necessary content)	✗ (Applications depend on virtual machine, leading to inefficiency)	✗ (Applications tend to be larger due to high-resolution assets)	✗ (Applications have higher local storage requirements)	✓ (Shared system resources reduce redundancy)
Performance Optimization	✓ (WebAssembly provides near-native performance)	✗ (Java virtual machine introduces performance bottlenecks)	✓ (Native code delivers stable performance)	✗ (Optimizations depend on app development quality)	✓ (Optimization depends on developer and hardware)
Background Sleep Support	✓ (PWA pages naturally sleep when inactive)	✗ (Some apps have frequent background activity)	✗ (Apps must be tailored for sleep modes)	✗ (Most apps cause significant background energy drain)	✗ (Apps must be specifically modified to fully sleep)
Energy Consumption	✓ (Unified browser manages resources efficiently, reducing power draw)	✗ (Energy usage depends on app and device quality)	✗ (Sandbox environment adds resource overhead)	✗ (Background resources are not fully optimized)	✗ (Energy draw varies without unified management)
Shared Resource Model	✓ (Browser provides shared pool for all PWA applications)	✗ (Sandbox applications rarely share resources)	✗ (Sandbox isolates application resources)	✗ (Applications are isolated, with no shared mechanism)	✓ (Applications share system resources effectively)
Update Impact	✓ (Only relevant content is updated, reducing resource strain)	✗ (Full updates require downloading large packages)	✗ (System updates tend to bundle unnecessary data)	✗ (System updates cause significant overhead)	✗ (Update impact depends on user management)

samples, achieving a detection rate of 83% and a false positive rate of 10%. Finally, *EDDIE: EM-Based Detection of Deviations in Program Execution*[30] by Nazari, Alireza, Sehatbakhsh, Nader, Alam, Monjur, Zajic, Alenka and Prvulovic, Milos monitored electromagnetic emissions from embedded and IoT devices, detecting statistical anomalies caused by code injections into application loops.

Understanding and Detecting Real-World Safety Issues in Rust[40] by Qin, Boqin, Chen, Yilun, Liu, Haopeng, Zhang, Hua, Wen, Qiaoyan, Song, Linhai, and Zhang, Yiying highlights that programming languages like Rust can encounter memory safety issues if the `unsafe` keyword is misused. The authors conducted an extensive study, analyzing five widely-used Rust libraries and two online security databases. Their manual inspection uncovered 70 memory bugs, 100 concurrency bugs, and 110 programming errors leading to panics. This research emphasizes that excessive reliance on `unsafe` significantly diminishes its intended effectiveness.

Is Google Getting Worse? A Longitudinal Investigation of SEO Spam in Search Engines[5] by Bevendorff, Janek, Wiegmann, Matti, Potthast, Martin, and Stein, Benno examines the decline in Google Search quality due to the rise of SEO spam. The study highlights how spam and link farms are becoming increasingly indistinguishable, a trend that is expected to worsen with the advent of generative AI.

Bringing the Web Up to Speed with WebAssembly[19] by Barzolevskaia, Anna, Branca, Enrico, and Stakhanova, Natalia serves as a foundational work introducing WebAssembly by detailing its motivation, design, and formal semantics. *Not so Fast: Analyzing the Performance of WebAssembly vs. Native Code* by Jangda, Abhinav, Powers, Bobby, Berger, Emery D., and Guha, Arjun developed BROWSIX-WASM to evaluate the performance gap between WebAssembly and native code, finding an average slowdown of 45% on Firefox and 55% on Chromium, with peak slowdowns of 2.08x (Firefox) and 2.5x (Chrome). The performance issues were attributed to both platform-specific factors, such as bounds checking and function call verification, and missing optimizations. *Everything Old is New Again: Binary Security of WebAssembly*[25] highlights how WebAssembly binaries, often compiled from memory-unsafe languages like C/C++, remain vulnerable to memory safety exploits, enabling the construction of malicious cross-site scripting attacks. *WebAssembly Memory Tagging*[53] by Shengdun W. and Aravind P. offers a solution to WebAssembly's memory safety concerns using memory tagging, inspired by the ARM Memory Tagging Extension. Their research demonstrates that software-based memory tagging introduces overheads of 48.91% for Wasm64 and 72.38% for Wasm32, but with ARM MTE-supported CPUs, these overheads drop significantly to 5.71% for Wasm64 and 18.05% for Wasm32, while effectively addressing real-world CVEs.

7 CONCLUSION

S/P mode, akin to the `unsafe` keyword in programming languages like C# and Rust, is a straightforward yet highly effective concept.

Table 4: Comparison: S/P Mode Toggle vs. Windows Group Policies

Feature	S/P Mode Toggle	Windows Group Policies
Configuration Method	Simple toggle between S Mode and P Mode via user interface, designed for all users	Configured through administrative tools like gpedit.msc or Active Directory, requiring technical expertise
Target Audience	Everyday users (S Mode) and advanced users (P Mode)	IT professionals and system administrators managing organizational environments
Implementation Complexity	✓ (User-friendly, minimal setup required; no technical knowledge needed)	✗ (Highly complex, configuration requires expertise, not suitable for non-tech-savvy users)
Dynamic Adjustment	✓ (Users can instantly switch between S and P Mode based on needs)	✗ (Requires admin approval and pre-configuration; adjustments are not user-initiated)
Scope of Control	Individual user-level settings (toggle per device)	Organization-wide enforcement across multiple devices and users
Feature Granularity	Limited to mode-based controls (e.g., app installation, resource access)	Highly granular (controls registry, software restrictions, network policies, etc.)
Ease of Rollback	✓ (Instantly revert back to S Mode)	✗ (Rollback requires manual reconfiguration or scripts, not easy for average users)
Security Management	✓ (Switch to P Mode for flexibility; back to S Mode for strict controls)	✓ (Powerful organizational security enforcement, but complex to implement)
Admin Rights Requirement	✗ (No admin privileges needed for toggling modes)	✓ (Admin rights are mandatory for setup, changes, and enforcement)
Usability for Everyday Users	✓ (Ideal for non-technical users who rely on basic apps like Word/Excel)	✗ (Overwhelming and inaccessible for users who only need basic functionality)
Usage Scalability	Best suited for single-user or device-level control	Designed for enterprise-level deployments, not casual individual usage
Integration with External Systems	✓ (Integrates with cloud services, PWAs support APIs for external communication)	✓ (Extensive integration with Windows Server, Azure AD, enterprise tools)

By leveraging progressive web applications and browser sandboxing, implementing an S/P mode toggle has the potential to mitigate most security risks associated with untrusted inputs. For non-technical users, a simple `walled garden` approach is essential to shield them from the inherent dangers of the internet.

8 FUTURE WORK

While S/P mode provides a protective `walled garden` for web browsers—currently the primary source of untrusted inputs—other vulnerabilities, such as malicious emails or SMS messages, may still bypass this safeguard. Extending the principles of S/P mode to address these challenges represents a promising direction for future research.

REFERENCES

[1] 2025. StatCounter Global Stats. (2025). https://gs.statcounter.com/ Accessed: 2025-03-23.

[2] Apple. 2021. Building a Trusted Ecosystem for Millions of Apps: A Threat Analysis of Sideloading. (2021). https://www.apple.com/privacy/docs/Building_a_Trusted_Ecosystem_for_Millions_of_Apps.pdf

[3] Apple. n.d.. App Sandbox - Apple Developer Documentation. (n.d.). https://developer.apple.com/documentation/security/app-sandbox Accessed: 2025-03-21.

[4] Alvin Ashcraft and cchavez. Win32 App Isolation Overview. https://learn.microsoft.com/en-us/windows/win32/secauthz/app-isolation-overview. (????). Accessed: 2025-03-24.

[5] Janek Bevendorff, Matti Wiegmann, Martin Potthast, and Benno Stein. 2024. Is Google Getting Worse? A Longitudinal Investigation of SEO Spam in Search Engines. In *Advances in Information Retrieval: 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24–28, 2024, Proceedings, Part III*. Springer-Verlag, Berlin, Heidelberg, 56–71. https://doi.org/10.1007/978-3-031-56063-7_4

[6] Jody Bruchon. n.d.. Windows 11 Must Be Stopped - A Veteran PC Repair Shop Owner's Dire Warning - Jody Bruchon. (n.d.). https://youtu.be/LcafzHL8iBQ Accessed: 2025-03-21.

[7] European Commission. 2024. Commission Sends Preliminary Findings to Apple and Opens Additional Non-Compliance Investigation Against Apple Under the Digital Markets Act. (2024). https://ec.europa.eu/commission/presscorner/detail/en/ip_24_3433

[8] Reddit User Community. n.d.. What's wrong with snaps, why so many people hate it? (n.d.). https://www.reddit.com/r/linux/comments/j3ajnf/whats_wrong_with_snaps_why_so_many_people_hate_it/ Accessed: 2025-03-23.

[9] Wikipedia contributors. 2024. 2024 CrowdStrike-related IT outages. (2024). https://en.wikipedia.org/wiki/2024_CrowdStrike-related_IT_outages Accessed: 2025-03-21.

[10] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the feasibility of online malware detection with performance counters. *SIGARCH Comput. Archit. News* 41, 3 (June 2013), 559–570. https://doi.org/10.1145/2508148.2485970

[11] Alex Dobie. 2022. Why Apple is wrong about sideloading. Android Central. (2022). https://www.androidcentral.com/why-apple-wrong-about-sideloading Accessed: 2025-03-24.

[12] EDGECELSIOR. 2024. Windows Phone WTF: EPISODE 7 - THE DREADED APP GAP, Microsoft banned C++ in UWP. (2024). https://youtu.be/-eF4Fdgi3KA Accessed: 2025-03-21.

[13] Craig Federighi. 2021. Apple Keynote: Privacy and Security. YouTube video. (2021). https://youtu.be/f0Gum8UkyoI Web Summit, 79.6K subscribers, Published on Nov 10, 2021.

[14] FlowOSS. n.d.. FlowOSS App. (n.d.). https://app.flowoss.com Accessed: 2025-03-21.

[15] State Administration for Market Regulation. 2025. China's National Market Supervision Administration Decides to Investigate Google for Suspected Monopoly Law Violations. (2025). https://www.samr.gov.cn/xw/zj/art/2025/art_396a9ab3aa6d4c4bbd40833815afd245.html

[16] David Fortunato and Jorge Bernardino. 2018. Progressive web apps: An alternative to the native mobile Apps. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. 1–6. https://doi.org/10.23919/CISTI.2018.8399228

[17] FreeBSDfan. 2024. We need a real GNU/Linux (not Android) smartphone ecosystem. (2024). https://www.reddit.com/r/linux/comments/1fx5fq0/we_need_a_real_gnulinux_not_android_smartphone/ Accessed: 27 March 2025.

[18] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the web up to speed with WebAssembly. *SIGPLAN Not.* 52, 6 (June 2017), 185–200. https://doi.org/10.1145/3140587.3062363

[19] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*. Association for Computing Machinery, New York, NY, USA, 185–200. https://doi.org/10.1145/3062341.3062363

[20] Julie Haney. 2022. Users are not stupid: Six cyber security pitfalls overturned. (2022). https://csrc.nist.gov/csrc/media/Projects/usable-cybersecurity/documents/Final_Proof_Users_are_not_stupid.pdf Accessed: 2025-03-21.

[21] Julie Haney. 2023. Users Are Not Stupid: Six Cyber Security Pitfalls Overturned. (2023). https://csrc.nist.gov/csrc/media/Projects/usable-cybersecurity/documents/Final_Proof_Users_are_not_stupid.pdf

[22] Jerry Hildenbrand. 2021. Android will never be supported by the 'regular' Linux kernel, but that won't stop Google from trying. (2021). https://www.androidcentral.com/android-will-never-be-supported-regular-linux-kernel-thats-not-stopping-google-trying Accessed: 2025-03-21.

[23] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. 2011. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering* 1, 4 (2011), 293–302.

[24] Taeri Kim, Noseong Park, Jiwon Hong, and Sang-Wook Kim. 2022. Phishing URL Detection: A Network-based Approach Robust to Evasion. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1769–1782. https://doi.org/10.1145/3548606.3560615

[25] Daniel Lehmann, Johannes Kinder, and Michael Pradel. 2020. Everything Old is New Again: Binary Security of WebAssembly. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 217–234. https://www.usenix.org/conference/usenixsecurity20/presentation/lehmann

[26] melontini. 2025. Bootloader Unlock Wall of Shame. (2025). https://github.com/melontini/bootloader-unlock-wall-of-shame

[27] Microsoft. Learn About Performance Features in Microsoft Edge. https://support.microsoft.com/en-us/topic/learn-about-performance-features-in-microsoft-edge-7b36f363-2119-448a-8de6-375cfd88ab25 (????). Accessed: 2025-03-24.

[28] Microsoft. 2024. Phishing trends and techniques - Microsoft Defender for Endpoint. (2024). https://learn.microsoft.com/en-us/defender-endpoint/malware/phishing Accessed: 2025-03-21.

[29] Microsoft. 2024. Prevent malware infection - Microsoft Defender for Endpoint. (2024). https://learn.microsoft.com/en-us/defender-endpoint/malware/prevent-malware-infection Accessed: 2025-03-21.

[30] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. 2017. EDDIE: EM-Based Detection of Deviations in Program Execution. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 333–346. https://doi.org/10.1145/3079856.3080223

[31] Nir Nissim, Yuval Lapidot, Aviad Cohen, and Yuval Elovici. 2018. Trusted system-calls analysis methodology aimed at detection of compromised virtual machines using sequential mining. *Know.-Based Syst.* 153, C (Aug. 2018), 147–175. https://doi.org/10.1016/j.knosys.2018.04.033

[32] United States Department of Justice. 2024. Justice Department Sues Apple for Monopolizing Smartphone Markets. (2024). https://www.justice.gov/archives/opa/pr/justice-department-sues-apple-monopolizing-smartphone-markets

[33] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. 2019. Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Comput. Surv.* 52, 5, Article 88 (Sept. 2019), 48 pages. https://doi.org/10.1145/3329786

[34] Mental Outlaw. 2021. Disproving Apple's Lies About Sideloading Apps. YouTube video. (2021). https://youtu.be/pUBe8VM6BFo Published on Nov 8, 2021, 144,997 views.

[35] Razvan Pascanu, Jack W. Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. 2015. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1916–1920. https://doi.org/10.1109/ICASSP.2015.7178304

[36] Snae Player. 2025. Snae Player. (2025). https://snaeplayer.com Accessed: 2025-03-21.

[37] Android Vulnerabilities Project. n.d.. Android Vulnerabilities by Year. (n.d.). https://androidvulnerabilities.org/by/year/ Accessed: 2025-03-21.

[38] Chromium Project. n.d.. The Rule of 2 - Chromium Security Documentation. (n.d.). https://chromium.googlesource.com/chromium/src/+/master/docs/security/rule-of-2.md Accessed: 2025-03-23.

[39] GNU Project. 2025. Tivoization. (2025). https://www.gnu.org/philosophy/tivoization.html Accessed: 27 March 2025.

[40] Boqin Qin, Yilun Chen, Zeming Yu, Linhai Song, and Yiying Zhang. 2020. Understanding memory and thread safety practices and issues in real-world Rust programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 763–779. https://doi.org/10.1145/3385412.3386036

[41] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. 2014. A machine-learning approach for classifying and categorizing android sources and sinks. In *NDSS*, Vol. 14.

[42] Guido Schwenk and Konrad Rieck. 2011. Adaptive Detection of Covert Communication in HTTP Requests. In *2011 Seventh European Conference on Computer Network Defense*. 25–32. https://doi.org/10.1109/EC2ND.2011.12

[43] Microsoft Support. Access Year. Windows 10 and Windows 11 in S mode FAQ. (Access Year). https://support.microsoft.com/en-us/windows/windows-10-and-windows-11-in-s-mode-faq-851057d6-1ee9-b9e5-c30b-93baebeebc85 Accessed: 27 March 2025.

[44] Digital Citizen Team. Test and Comparison: Which Browser Will Make Your Laptop Battery Last Longer? https://www.digitalcitizen.life/test-comparison-which-browser-will-make-your-laptop-battery-last-longer/. (????). Accessed: 2025-03-24.

[45] Microsoft Edge Team. Overview of Progressive Web Apps (PWAs). https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/. (????). Accessed: 2025-03-24.

[46] Microsoft Edge Team. 2022. Sleeping Tabs: Delivering up to 99More with Microsoft Edge 100. https://blogs.windows.com/msedgedev/2022/04/07/sleeping-tabs-edge-100-improvements/. (2022). Accessed: 2025-03-24.

[47] Microsoft Edge Team. 2025. Stand up to scareware with scareware blocker. (2025). https://blogs.windows.com/msedgedev/2025/01/27/stand-up-to-scareware-with-scareware-blocker/ Accessed: 2025-03-21.

[48] Chris Titus Tech. 2022. Ubuntu's Decline. (2022). https://www.youtube.com/watch?v=pMfqCzbSmQU Accessed: 2025-03-23.

[49] Peter "Durante" Thoman. 2016. Why PC games should never become universal 'apps'. (2016). https://www.pcgamer.com/why-pc-games-should-never-become-universal-apps/ Accessed: 2025-03-21.

[50] Sam Thorogood. 2018. The Web Is Dead! Long Live The Web! YouTube video. (2018). https://youtu.be/ogMjMN1G6V8 LinuxConfAu 2018 - Sydney, Australia, Published on Jan 31, 2018.

[51] GitHub user. 2020. Why do you hate UWP? Seems promising thing. (2020). https://github.com/pbatard/rufus/issues/1617 Accessed: 2025-03-21.

[52] Reddit user. n.d.. Creator of Rufus outlines the problems with Microsoft's UWP. (n.d.). https://www.reddit.com/r/programming/comments/noagrx/creator_of_rufus_outlines_the_problems_with/ Accessed: 2025-03-21.

[53] Shengdun W. 2024. Public Presentation for WebAssembly Memory Tagging. (25 November 2024). https://www.youtube.com/live/8W9faYGooqA Streamed live, 64 views as of access. No description provided. Accessed: 2025-03-23.

[54] WebAssembly. 2025. WebAssembly. https://webassembly.org/. (2025).

[55] web.dev. n.d.. Progressive Web Apps - Learn PWA. (n.d.). https://web.dev/learn/pwa/progressive-web-apps?hl=fr Accessed: 2025-03-21.

[56] Tim Werthmann, Ralf Hund, Lucas Davi, Ahmad-Reza Sadeghi, and Thorsten Holz. 2013. PSiOS: bring your own privacy & security to iOS devices. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '13)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/2484313.2484316

[57] Stephan Wesemeyer, Christopher J.P. Newton, Helen Treharne, Liqun Chen, Ralf Sasse, and Jorden Whitefield. 2020. Formal Analysis and Implementation of a TPM 2.0-based Direct Anonymous Attestation Scheme. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20)*. Association for Computing Machinery, New York, NY, USA, 784–798. https://doi.org/10.1145/3320269.3372197