Open
Web Advocacy

Digital Markets Act - Interventions
In-App Browsers

VERSION 1.2

Open Web Advocacy
contactus@open-web-advocacy.org

1. Table of Contents

2. Introduction

Imagine a world where every desktop app you use has its own built-in browser, in many cases built on a tiny budget, missing features, and, worst of all, is able to spy on and manipulate third-party websites.

This is the reality users face on their phones today. It doesn't have to be this way. Mandating that apps respect browser choice isn't just a matter of convenience; it's about empowering both consumers and companies to thrive in a more open, stable, feature rich, secure and private digital ecosystem.

Disrespect of users' choice of browser also stifles innovation. The Web thrives when browsers compete on functionality and user experience. In-app browsers, however, don't compete as browsers. Many, perhaps even most, users are unaware they are being foisted upon them. They're also unaware of the quality, security and privacy risks associated with them. In-app browsers operate in a vacuum, immune from competitive pressure to improve.

This stagnation hurts everyone. Users are stuck with forgetful, subpar in-app browsers. Businesses struggle to reach and engage with audiences due to missing features and bugs in browsers that no one made an affirmative choice to use.

The solution is clear – *respect the user's choice of browser*.

Mandating that non-browser apps utilize a users' default browser for third-party websites, will unlock a more vibrant and equitable digital landscape. Users will enjoy familiar tools they trust, experiencing websites as they were designed. Businesses and publishers will gain access to earned audiences, free from interference by native app developers.

The intrusion of in-app browsers, along with the significant bugs, missing features and critical privacy and data protection concerns they introduce should be addressed. The Digital Markets Act (DMA) contains all of the necessary enforcement powers to right this wrong. This document outlines how it can be done, both technically and legally.

We propose 6 remedies:

1. Designated Core Platform Services should respect the users choice of default browser.

2. App Store rules must mandate non-browser apps use the user's chosen default browser for http/https links to third-party websites/Web Apps.

3. Apple must update SFSafariViewController (Apple's system provided in-app browser for iOS) to respect the user's choice of default browser.

4. Third-Party businesses must be provided an explicit and effective technical opt-out from non-default In-App Browsers.

5. OSes must provide a global user opt-out. Apps must also request explicit permission from users.

6. Google must remove the ability to override a user's choice of default browser via Android Custom Tabs (Google's system provided in-app browser for Android) .

These remedies overlap, addressing different aspects of the problem. When effected, these remedies will prevent gatekeepers from subverting the user choice of browser, provide users enhanced control over their privacy and security, and make technical fixes to the underpinnings of how web pages are loaded. These fixes will project a user's choice of browser in non-browser apps they use, providing users the security, privacy, stability and features on which browser wars are legitimately fought. No longer will apps be allowed to siphon web traffic for their own enrichment at the detriment of the community at large.

3. Definitions

This is a confusing subject for experts and non-experts alike. Discussion of in-app browsers is full of difficult, incomplete terminology, creating stumbling blocks even for those who work in the industry.

Therefore, we first define terms before discussing problems or remedies. Where existing terminology is vague or non-descriptive, we create new terms and endeavor to use them consistently within our submissions.

3.1. Browser

Most users are familiar with browsers. They are applications that open and display web content when users click links, whether from web pages or in external programs that handle email, sms, or documents. Examples include Safari, Firefox, Chrome, Edge, Opera, Brave and Vivaldi.

The definition of “a browser” is a recurring feature of DMA concern, and OWA believes:

> *A browser is an application that can register with an OS to handle `http` and `https` navigations by default.*

Put another way, browsers are applications that open when a link is clicked or tapped. By this definition, apps like Gmail are not browsers, even though they can display HTML-formatted emails.

True browsers are products that users choose to handle their internet browsing. These products, and user choices about them, have enjoyed regulatory protection around the world since the late 90s. This is natural, as browsers act as both a gateway to the internet, and to a standards-based application platform in its own right.

Browser choice is meaningful to users as browsers enforce user's wishes in relation to privacy and security. It is essential that the user retains control over these choices so that they can choose a browser that best fits their utility, privacy and security needs. User choice is also the engine that ensures competition between vendors and continued investment in the standards-based commons.

An application is a browser if:

1. Its **Primary Purpose** is to browse third-party content on the Web.
   - Third party content is content not owned or controlled by the Browser Vendor. The definition of first, second and third-party content is discussed later in this document.

2. It serves as a **handler of http/https links**.
   - The browser has the ability to register with the operating system and when a user taps/clicks an http/https link in other non browser applications, it can open in the browser.

3. The primary interface displayed when the application is launched, facilitates browsing third-party content, with non-browsing functionality being visually secondary.

3.2. Default Browser

The **default browser** is the browser registered with the system to open http/https navigations from non-browser apps.

How a default browser is set differs between operating systems and is sometimes subject to anti-competitive behavior and dark patterns. Depending on the OS, the default browser can sometimes be changed during browser installation, through browser settings, system settings, or via a prompt. Many users stay on the browser provided by the operating system, and never attempt to change the default.

The Digital Markets Act contains specific provisions to make it easy for users to change their default browser, including prompting them with a choice screen. These provisions are an important rule to allow effective browser choice but only to the extent that that choice is respected when users browse the Web.

3.3. WebView

A webview is a system component for rendering web content. It typically does not provide browser user interface, including address bar, forward/back buttons, settings, etc.

Both iOS and Android provide system webviews for use by native apps. iOS provides the WKWebView, based on the version of Webkit bundled with iOS. Android provides Android WebView, based on the version of Blink bundled with Android, but updated independently of the OS as part of the Chrome update process.

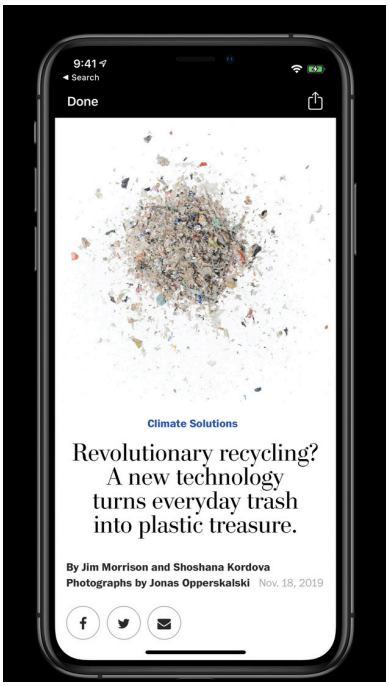
These webviews do not rely on apps to include updated versions, instead they are updated separately by the OS.

3.3.1. What WebViews are useful for?

Webviews can be useful for:

1. **Rendering web content inside a native app (first-party content).**

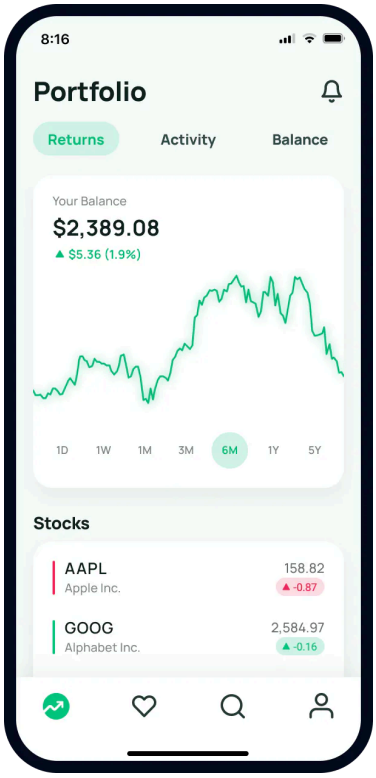
   Apps can use webviews to display first-party web content (content created by the app developer) inside apps. This allows native app developers to re-use their web content where it makes sense. A common use case is help documentation within apps, or other complex text layout tasks like displaying email.



An example of the Washington Post using the iOS WKWebView to display content within their app

2. **Rendering a Web App inside a Native App container (first-party content).**

   There's a long history in native apps using webviews as part of platforms that allow developers to primarily develop using web technology. Important examples include PhoneGap (now Cordova) and Ionic. Apps built this way behave like native apps with a single codebase that works on multiple platforms. These apps are often referred to as "hybrid apps". Essentially they are wrapper native apps that use the webview to host the web app.



An example app from Ionic that uses the web technology to produce native apps that work on both Android and iOS.

3. **Displaying Ads (second-party content).**

   Native Apps can use a webview to display advertisements such as Mobile Rich Media Ads (MRAID). This allows advertisers and app developers to display adverts within native apps using web technology, which then is interoperable between devices. Importantly, the content from advertisers expects to run in a limited

environment, without the full power of the web platform provided by complete browsers.

4. **Developing a WebView In-App Browser (third-party content).**

   Using a webview to display third-party web content inside of a native app when a user clicks a link. Native app developers can surround such webview with a custom user interface. These browsers are limited and problematic, as discussed here.

5. **Developing a WebView Browser (third-party content).**

   Browser developers can use webviews to build browsers without having to ship their own engine. This is discussed further below.

3.4. WebView Browsers

Browsers can be built using the OS-provided webview component for rendering web pages, in lieu of bundling their own engines.

These apps qualify as browsers because their primary purpose is to browse the web and they can register with the operating system to handle http/https links. Their primary interface is also designed for browsing third-party content.

Owing to Apple policy forbidding the bundling of alternative engines, all iOS browsers except iOS Safari are webview browsers. Apple has announced a proposal to comply with the DMA but we are concerned it is insufficient and ineffective.

While we take strong exception to this rule and believe it severely damages the Web, producing a webview browser is a legitimate choice that vendors can make in an open market.

Apple's official documentation aligns with our concern that webviews are not generally capable enough to provide competitive browsing experiences:

> *"**Avoid using a web view to build a web browser.** Using a web view to let people briefly access a website without leaving the context of your app is fine, but **Safari is the primary way people browse the web. Attempting to replicate the functionality of Safari in your app is unnecessary and discouraged.**"*
>
> Apple Official Documentation
> (emphasis added)

Relying on the system webview allows browser developers to reduce costs on engine development, but removes any practical ability for them to differentiate on platform features. The poor tools provided by Apple and Google for webview "flag" customization, a mechanism used to toggle browser features on or off, mean that the relative simplicity of webview browser construction is outweighed by the difficulty and cost in providing equivalent features across platforms in all but the simplest cases.

While webview browsers may be a good option for companies with smaller budgets, they are not effective replacements for true engine choice for large-scale, competitive browser developers.

3.5. In-App Browser

An in-app browser can open when users tap on links in a non-browser app.

Rather than switching out of the app and to the user's default browser, in-app browsers open a pane within the host native app to render web pages. Hence the name, "in-app browser".

There are three implementation strategies for in-app browsers::

- **Remote Tab** In-App Browser
- **WebView** In-App Browser
- **Bundled Engine** In-App Browser

  Note: *We have invented this terminology to more clearly distinguish in-app browser types.*

3.5.1. Remote Tab In-App Browser

Remote tab in-app browsers are a type of system component provided by the OS which rely on a real browser to handle navigations without removing users from the app. Remote tab in-app browsers are the newest type of in-app browser, and OWA's preferred variant.

When apps invoke this component upon a user clicking a http/https link, a pane with simplified browser user interface opens to render the website. This pane uses either the OS's browser or the default browser to render web content.

Both Android and iOS provide components for applications that want to invoke browsers for loading content in the in-app browser style. iOS's SFSafariViewController is

hard-coded to load content in Safari (it does not respect browser choice), while Android Custom Tabs will respect choice by default, but this can be overridden.

Android Custom Tabs will invoke and render with the user's default browser in most cases (i.e. Chrome, Firefox, Edge, etc) but can be locked by the hosting app to a specific browser. OWA believes this happens most frequently via the Android Google Search App ("AGSA" or "AGA") that loads all web pages linked from Google Search results in Google Chrome, regardless of the user's default browser choice. **In our opinion, Android Custom Tabs should be required to invoke the default browser in all cases.**

iOS's SFSafariViewController, on the other hand, always invokes and renders web pages in Safari, subverting browser choice entirely. It is our opinion **SFSafariViewController should be required to be modified to always invoke the user's default browser.**

Remote tab in-app browsers have a number of advantages for users:

- Hosting apps cannot inject JavaScript or modify the page or track users, as the browser is sandboxed to clearly separate the host-app from the browser content.

  There is no way for the app developer to access, read or modify the content of the rendered website, nor to track any follow-up navigation (e.g. fill out a form, navigate to a sub-page, etc) without the explicit cooperation of the webpage.

- Logins and settings are not forgotten. Session information such as cookies and form auto-completion are not forgotten by well-behaved remote tab in-app browsers, ensuring that user privacy, security, and permissions settings are applied consistently.

  It is worth noting this is **not** the case in SFSafariViewController since iOS 11, we cover this in this section.

- Extensions and content blockers work consistently, inheriting their settings from the user's default browser.

- Accessibility, privacy, and security settings from the invoked browser are also shared.

3.5.2. WebView In-App Browser

Webview in-app browsers are custom-built in-app browsers that rely on the OS provided system webview to render web content. They do not ship their own browser engine, and do not share any settings, state, preferences, or customization across apps. They are

surrounded by a custom user interface that may or may not contain a control allowing users to jump into their default browser.

Webview in-app browsers are incredibly common, and are perhaps most often encountered in Meta's suite of iOS applications, including in Facebook, Messenger, and Instagram.

3.5.3. Bundled Engine In-App Browser

Bundled engine in-app browsers are custom-built in-app browsers created by the hosting app and using a browser engine bundled with the app to render web content. They do not make use of the OS provided system webview. They are surrounded by a custom user interface that may or may not contain a control allowing users to jump into their default browser.

iOS does not allow any app (including actual browsers) to use their own engine, so these bundled engine in-app browsers are typically found on Android. Meta's Facebook app for Android features such an in-app browser for all links tapped from the Facebook for Android timeline.

3.6. First-Party and Affiliated Content vs Third-Party Content

3.6.1. First-Party Content

First-party content is content for a developer's app that the developer made, owns, licenses and/or hosts.

3.6.2. Second-Party (Affiliated) Content

Second-party (affiliated) content is content from a second-party that the first-party has a strong business relationship. The most common use case is adverts and analytics. Second-party content can be considered as consenting to be displayed in the developers app.

3.6.3. Third-Party Content

Third-party content can be thought of as non-cooperating content. Third-party content is all non first- or second-party content. This is content from third parties that the app developer has no relationship with.

3.6.4. How do we work out first, second vs third-party content?

In the context of links to websites and Web Apps classifying content into first, second or third-party content is relatively straightforward.

1. Does the app run/host/control/own the website?
   Yes: First-party content

2. Does the app that wants to render the content in their in-app browser have a strong business relationship and the explicit consent of the party that hosts the content at the hyperlink?
   Yes: Second-party content

3. All other websites are **non-cooperating third-party content.**

4. Problems

Each style of in-app browser can present unique challenges, and all have the potential to subvert browser choice and reduce the effectiveness of the Web for users and businesses if not reined in.

4.1. Problems with WebView In-App Browsers

Webview in-app browsers introduce a wide range of problems for users and businesses. In our view, there is little to no benefit to any party aside from for the hosting app that has subverted the user's choice of default browser.

4.1.1. Overrides the Users Choice of Default Browser

In-app browsers frequently divert web content away from default browsers without explicit notification or consent. This covert operation leaves users unaware that their browsing experience has been altered, undermining their ability to make informed choices about their online interactions. The lack of transparency surrounding in-app browser usage hinders user agency.

App makers may argue their webview in-app browsers offer users significant benefits, and that users would voluntarily and actively choose to use them to browse third-party websites. However, the behavior of apps today does not present an informed choice to users. OWA believes that choice in browsers should be prominent and explicit.

For example, Meta claims their Instagram in-app browser protects its users, but at the same time doesn't affirmatively present users with prominent choice in browser when opening links, even after the press publicity of referenced articles were published.

Meta's behavior shows a pattern and practice of disrespect for browser choice via in-app browsers. Each Meta-produced app automatically opts users into their in-app browser for navigations from their apps. The controls for disabling this behavior are incredibly difficult to find. The setting for this behavior is not synced across devices. No option is available for disabling the in-app browsers across all Meta apps from a single account.

This means that settings for disabling Meta's choice-subverting in-app browsers must be configured in every app, on every device, from incredibly difficult to find menus that are different between each app. The predictable result is that user choice is not credibly provided or respected.

This subversion of choice has important implications regarding competition, privacy, security, stability, feature set and web experience. Indeed, degraded security and privacy properties are an inherent issue of in-app browsers.

4.1.2. Privacy and security

Aside from degraded features, webview and bundled engine in-app browsers present significant privacy and security concerns.

OWA believes the vast majority of users are not aware that they are not using their default browser, or that settings to prevent this subversion may be available.

With both webview and bundled engine in-app browsers, the app developers can observe and manipulate web pages that the users visit. Apps can monitor what users view and do through their in-app browsers and even modify the page's contents. It is not credible, as some defenders of in-app browsers assert, that users understand they are granting these Native Apps the ability to surveil their browsing or defeat their extensions (e.g., to block trackers and ads).

Whatever the design intent of operating systems vendors in providing webviews, the increasing level of concern by policy makers and advocates regarding digital surveillance and cybersecurity is out of step with a status quo that facilitates wide-scale removal of effective privacy and security control from users by powerful app makers.

This chart shows a number of major apps on iOS that have their own webview in-app browser and whether or not they inject JavaScript into third-party web pages:

App	Option to open in default browser	Modify page	Fetch metadata	JS	Updated
TikTok	⛔	Yes	Yes	.js	2022-08-18
Instagram	✅	Yes	Yes	.js	2022-08-18
FB Messenger	✅	Yes	Yes	.js	2022-08-18
Facebook	✅	Yes	Yes	.js	2022-08-18
Amazon	✅	None	Yes	.js	2022-08-18
Snapchat	✅	None	None		2022-08-18
Robinhood	✅	None	None		2022-08-18

Instagram was shown for example to inject JavaScript into every webpage that the users visits using their webview in-app browser.

> *"**Instagram iOS subscribes to every tap on any button, link, image or other component on external websites** rendered inside the Instagram app.*
> *Instagram iOS subscribes to every time the user selects a UI element (like a text field) on third-party websites rendered inside the Instagram app.*
>
> *Note on subscribing: When I talk about 'App subscribes to', I mean that the app subscribes to the JavaScript events of that type (e.g. all taps). **There is no way to verify what happens with the data.**"*
>
> Felix Krause - Security & Privacy Researcher
> (emphasis added)

In response to an earlier article by Felix Krause, a Meta spokesperson said:

> *"The code in question allows us to respect people's privacy choices by helping aggregate events (such as making a purchase online) from pixels already on websites, before those events are used for advertising or measurement purposes"*
>
> Andy Stone - Meta Spokesperson

While we believe that Meta's statement is technically correct regarding the purpose of the script being injected, this appears to be an acknowledgment that Meta is using this privileged position and access to private user interactions with third-party websites for the purposes of better targeted advertising. Meta to our knowledge has not publicly outlined the purpose of their injected JavaScript in detail.

The Register asked Meta's spokesperson to elaborate on how injecting code in a custom in-app browser to assess user tracking preferences can be said to "*honor people's [ATT] choices*" when opening web pages in users' preferred browser, or with the help of Apple's SFSafariViewController, would do so more efficiently. They received no response. 16 months after the initial statement by Meta, there were no changes made to address the privacy and security concerns, neither by Apple, nor by Meta.

While Meta's apps (Instagram, Messenger, Facebook) in-app browser offers a button called "Open in External Browser". That feature is rather hidden, and for non-tech savvy users it will be hard to find. With TikTok, the situation is worse, as they opted to not include any button to open the currently shown website in the user's default browser, nor allowing them to access or copy the full URL.

> *"This should not come as a surprise, given that the FCC already called TikTok an unacceptable security risk. **When you open any link on the TikTok iOS app, it's***

> ***opened inside their in-app browser.*** *There is no alternative. While you are interacting with the website,* ***TikTok subscribes to all keyboard inputs (including passwords, credit card information, etc.) and every tap on the screen****, like which buttons and links you click. There is no way to know what TikTok uses the subscription for, but from a technical perspective,* ***this is the equivalent of installing a keylogger on third-party websites****. TikTok confirmed that those features exist in the code, but said that it is not using them."*
>
> Pieter Arntz - MalwareBytes
> (emphasis added)

It is not clear what benefit users receive from using either webview or bundled engine in-app browsers. Many major native apps choose not to provide them and simply use the users default browser (see chart below). Opening in the user's default browser either directly or by the system provided remote tab in-app browser (SFSafariViewController on iOS, Android Custom Tabs on Android) preserves both the user's choice of browser and their privacy.

This chart is a list of major apps on iOS that, instead of using their own webview in-app browsers, use either the default browser directly or use the system provided remote tab in-app browser (SFSafariViewController).

App	Technology	Updated
Twitter	SFSafariViewController	2022-08-15
Reddit	SFSafariViewController	2022-08-15
WhatsApp	Default Browser	2022-08-15
Slack	Default Browser	2022-08-16
Google Maps	SFSafariViewController	2022-08-15
YouTube	Default Browser	2022-08-15
Gmail	Default Browser	2022-08-15
Telegram	SFSafariViewController	2022-08-15
Signal	SFSafariViewController	2022-08-15
Tweetbot	SFSafariViewController	2022-08-15
Spotify	Default Browser	2022-08-15
Venmo	SFSafariViewController	2022-08-15
Microsoft Teams	Default Browser	2022-08-16
Microsoft Outlook	Default Browser or Edge	2022-08-16
Microsoft OneNote	Default Browser	2022-08-16
Twitch	Default Browser	2022-08-16

Apps that directly open the default browser or use system-provided remote tab in-app browsers can not inject Javascript, read, modify or track interactions on third-party websites.

These examples highlight that, via these webview and bundled engine in-app browsers, **users are unwittingly granting apps significant access to their private interactions with third-party websites**. Whether or not these apps use or even misuse this data is not known, and we have no evidence to prove that they have, but the fact that they can (and indeed some do, at least locally) collect this data is concerning when paired with the lack of affirmative notice and control. Each of these apps could choose to preserve the users privacy simply by using the users default browser when clicking on links to non-cooperating third-party websites.

4.1.2.1. Apple's App Tracking Transparency

The terms of ATT suggest that apps are out of compliance if they track users via IABs.

Apple's guidance states that:

> *"Examples of tracking include, but are not limited to:*
>
> *- Displaying targeted advertisements in your app based on user data collected from apps and websites owned by other companies."*

However, OWA is not aware of any effective enforcement of these provisions with regards to IABs. In fact, as outlined in our document there is strong evidence that this is not being effectively enforced even for standard native apps usage.

Even when users do not consent to Apple uniquely identifying them to Native Apps, the permissive privacy and security model of platform-specific apps (relative to the Web) Apps facilitates this fingerprintable collection through myriad APIs not available to Web Apps or only available to Web Apps behind permission prompts:

> *"When it comes to stopping third-party trackers, App Tracking Transparency is a dud. Worse, giving users the option to tap an 'Ask App Not To Track' button may even give users a false sense of privacy,"*
>
> Johnny Lin - Lockdown co-founder, Former Apple iCloud engineer

For example when users did not consent to be tracked via ATT on iOS, platform-specific games such as Subway Surfers — listed as one of the App Store's "must-play" games — collected and shared with advertisers the following data in late 2021, years after began advertising under the slogan "*Privacy. That's iPhone.*":

- Device Name (e.g., "John's iPhone X")
- Accessibility Setting: Bold Text
- Accessibility Setting: Custom Text Size
- Display Setting: Dark Mode
- Screen Resolution
- Time Zone
- Total Storage Space (bytes precision)
- Free Storage Space (bytes precision)
- Currency (e.g., "USD")

- iOS Version
- Audio Output (e.g., “Speakerphone”/"Bluetooth")
- Audio Input (e.g., “iPhone Microphone”)
- Accessibility Setting: Closed Captioning
- Country
- Cellular Carrier Name (E.g., “AT&T")
- Cellular Carrier Country
- Last Restart Time (Exact Timestamp, Second Precision)
- Calendar Type (E.g., “Gregorian”)
- Enabled Keyboards (E.g., “English, Emoji, Arabic”)
- Current Battery Level (15 decimals precision)
- Current Volume Level (3 decimals precision)
- Accessibility Setting: Increase Contrast
- Current Screen Brightness (15 decimals precision)
- Portrait/Landscape Mode
- Battery Charging State (E.g., “Plugged In”)
- iPhone Model (E.g., “iPhone X")
- Language
- User Agent (Browser Agent)
- IP address

> *"Our investigation found the iPhone’s tracking protections are nowhere nearly as comprehensive as Apple’s advertising might suggest. We found at least three popular iPhone games share a substantial amount of identifying information with ad companies, even after being asked not to track.*
>
> *When we flagged our findings to Apple, it said it was reaching out to these companies to understand what information they are collecting and how they are sharing it. After several weeks, nothing appears to have changed."*
>
> Geoffrey Fowler And Tatum Hunter - Washington Post

“an analysis of a number of popular iPhone apps found that they were sending a crazy amount of data about your device to an ad company. It seems pretty obvious that the specificity of this data is designed to fingerprint your device.”

Ben Lovejoy - 9to5mac

"Fingerprinting" being the ability to uniquely re-identify users silently based on information available without any consent prompt.

While Apple has nominally forbidden fingerprinting in the App Store, it has not enforced these terms. Instead, it made noise since '17 about removing sources of entropy from Safari, while only just recently begun a tepid clamp-down of native app fingerprinting abuse. This effort does not meaningfully limit runtime use of APIs or impose data use policies on App Store publishers. Instead, it only requires developers to attest to a "reason" for their data request.

Apple has begun to belatedly introduce unenforced "nutrition labels" that shift the burden of understanding tracking by native apps onto the user.

> *"But this is only the tip of the iceberg. Now the app stores should take the next step: ban SDKs from any data brokers that collect and sell our location information."*

> *"There is no good reason for apps to collect and sell location data, especially when users have no way of knowing how that data will be used. We implore Apple and Google to end this seedy industry, and make it clear that location data brokers are not welcome on their app stores"*

Bennett Cyphers - Electronic Frontier Foundation

4.1.2.2. Google Privacy Sandbox

Google's Privacy Sandbox effort largely impacts Google's own first-party behaviour. In this way, we believe that it will have minimal impact outside of Chrome use, which is only relevant for apps that use ACT.

Regarding Android native app restrictions from Privacy Sandbox, Google's documentation currently suggests that the current status is "*Design proposals publicly available for review and feedback*" and that "*The Privacy Sandbox on Android is a multi-year effort.*"

We are not aware of Privacy Sandbox policies that have gone into effect, nor any proposals to reduce collection through IABs as a result of Privacy Sandbox initiatives. This is doubly concerning, as it would appear to give Google broad leeway to delay improving privacy by clamping down on IABs while continuing to facilitate the worst behaviours of native apps while putting ads offered through web browsers at a targeting disadvantage to the same sites loaded in IABs.

4.1.3. Extensions (such as ad-blockers and tracker blockers) are not shared

When using a webview in-app browser, the user's default browser extensions are not shared. That means ad-blockers and tracker blockers are functionally disabled.

4.1.4. Privacy and Security Settings not respected

Importantly the users privacy and security settings set in their default browser are not respected, possibly against user wishes or awareness.

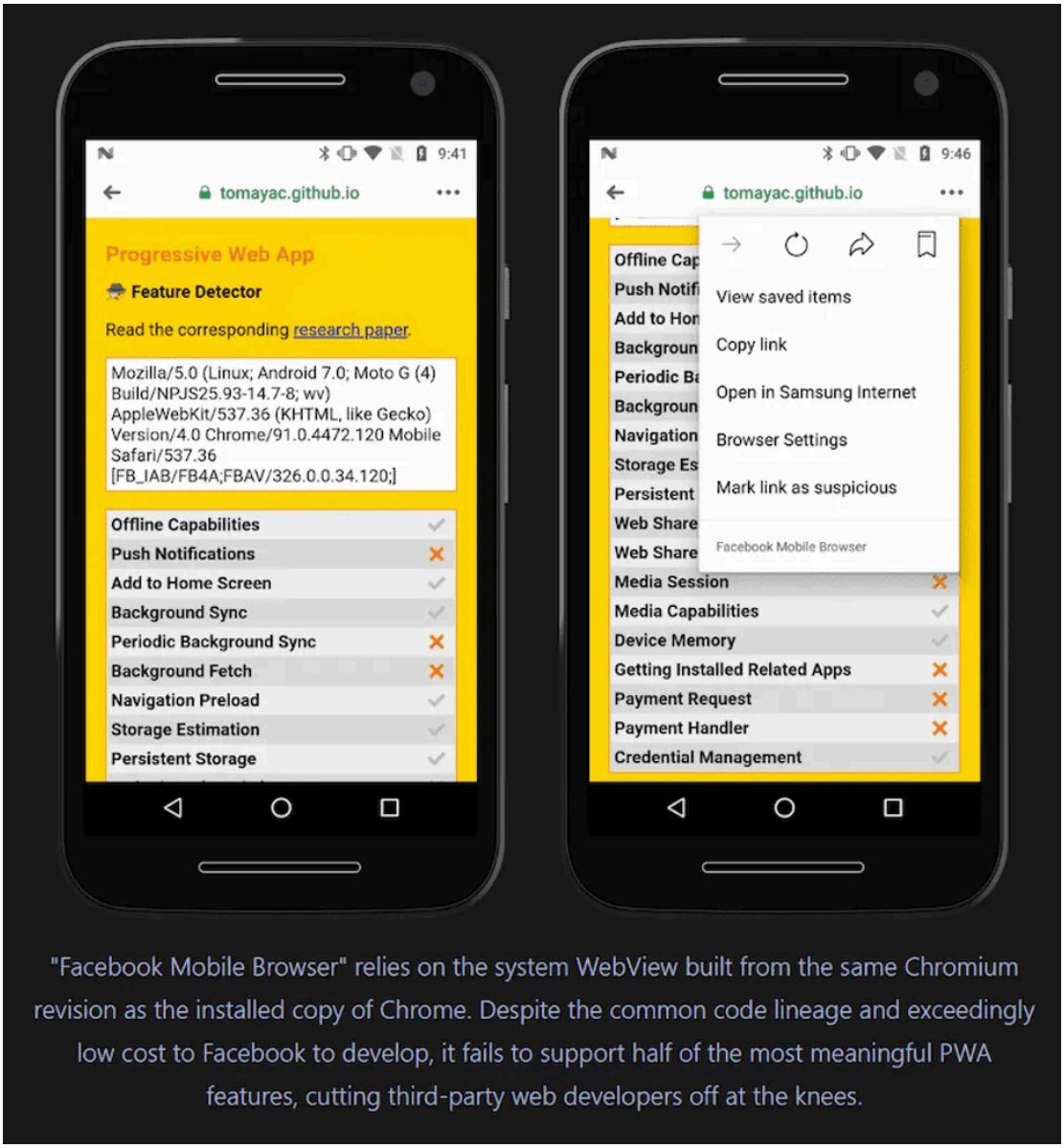
The lack of transparency inherent of many in-app browsers raises concerns regarding data sharing and utilization. Unlike established browsers, which offer granular control over data sharing and clearing, webview and bundled engine in-app browsers operate under the privacy policies of their host apps, subjecting all browsing activity within them to the terms and conditions of the app, not their browser.

Many browsers have tracking blocking built into the browser. While webview in-app browsers can implement this functionality, they also have the ability to not implement it, or to circumvent protections that are added to the webview. It is entirely the choice of the hosting app that builds the in-app browser.

There’s additional cause for concern regarding the state of security indicators and permissions controls. In-app browsers have tended to feature threadbare controls for powerful permissions like geolocation. They have also tended to fail to respect strong policies common to real browsers regarding security indicators like the “lock” icon. Providing competent and complete controls over these important concerns is a full-time job for sizable teams, and in-app browser vendors tend not to fund this work to a level that inspires confidence.

4.1.5. Missing Features

As discussed in this article, creating an in-app browser on top of a webview requires wiring up significant amounts of user interface and code, a non-trivial task.



Features that need extra care to support include:

- Re-engagement features including:
  - Web App installation and home screen shortcuts for sites
  - Push Notifications

- Privacy and site quality support, including "acceptable ads" enforcement and browser history clearing.

- Security indicators (TLS padlock, interstitial warnings, etc.)

- Basic navigation and window management features; for example the ability to open new tabs using `window.open()` and `<a target="_blank">`. These features are critical to some site monetization flows.

- Friction reducing OS integrations such as:
  - Web Payments (streamlined e-commerce checkout)
  - Web OTP (for easier/faster sign-in)
  - Web Share

- Hardware access APIs, notably:
  - Geolocation
  - Camera/mic (getUserMedia())
  - Web Bluetooth
  - WebUSB
  - Web Serial
  - WebHID
  - WebMIDI
  - Web NFC
  - Filesystem Access

> *"Few (if any) WebView browsers implement all of these features, even when underlying system WebViews provide the right hooks.*
>
> *The situation is even more acute in WebView IABs, where features **are often broken even when they appear to be available to developers**. Debugging content in IAB franken-browsers is challenging, and web developers are often blind to the volume of traffic they generate, meaning they may not even understand how broken their experiences are."*
>
> Alex Russell - Program Manager Microsoft Edge

This is likely due to the fact this is simply not a priority for these companies, they only have very small teams running these in-app browsers and as they are not competing as a browser (many users are unaware they are not using their default browser) they have no fear of losing market share by breaking third-party websites.

These missing features limit the capabilities of Web Apps in the context of in-app browsers and hurt their reputation in the eyes of users who are not aware of the reasons for these limitations. They naturally end up blaming the Web Apps rather than the in-app browser. They are then led to believe these Web Apps cannot fulfill their needs and discard them, hurting the companies developing them.

4.1.6. Unique bugs/issues

These missing features can also lead to bugs.

Because an underlying webview may technically support an API if a developer does the work to hook it up correctly, webview in-app browsers may incorrectly signal to developers that features which do not work are available. Feature detection is a method web developers use to work out whether a browser supports a feature by programmatically asking it. When webview in-app browsers fail to implement features correctly, these APIs will return "true", but attempts to try and use the API silently fail.

Webview and bundled engine in-app browsers can also introduce new bugs. For example, Facebook Messenger's webview in-app browser for iOS closes the in-app browser if users swipe-down anywhere in a web page.

This breaks the user interface of websites that use swipe down gestures. This is not an issue in either the system's remote tab in-app browser or directly in the user's default browser.

When google.com detects it is in Facebook Messenger in-app browser it **disables** the gesture using custom code to work around this unique bug. Other sites such as

pinterest.com do not, causing users to unintentionally close the whole in-app browser when interacting with the app. This is doubly difficult for businesses to cope with as documentation and debugging tools for widely used in-app browsers are effectively nonexistent.

Businesses trying to make money on the web understand that webview and bundled engine in-app browsers often break their experiences, present loyal users with logged-out views, and harm their conversion metrics. If you own a website and don't want to lose visitors, you have to know about all of the issues, how to detect them, and build custom fixes for each one where your functionality no-longer works.

Developers of individual apps aren't incentivised to ensure that websites work correctly, and this is an important reason why gatekeepers ensure that the user's chosen browser is used to run websites and Web Apps.

As Adrian Holovaty, Co-Creator of the Django web framework noted:

> *"Our main content type — the thing we usually link to — is a piece of interactive sheet music, which is dynamically sized to your screen. We've specifically had problems with the Instagram WebView not properly communicating its screen size — leading our site to apply incorrect dimensions to the sheet music. Again, it's a situation where we look bad due to an obscure technical detail outside our control."*
>
> Adrian Holovaty - Web Developer

This is an additional layer of completely unnecessary bugs that are extremely hard for developers to debug as these in-app browsers, despite their massive usage and traffic, do not compete as browsers and do not provide any of the typical developer tooling.

> *Some of my stuff was broken the last time I opened it in Facebook Messenger WebView. I haven't dug into it, as* ***it's hell to debug***
>
> Bruno - Web Developer

When confronted with these bugs, users often blame the Web Apps rather than the in-app browsers, causing them to discard the Web Apps. Companies relying on the web for their products end up losing customers.

4.1.7. Amnesia - Session State, Autofill etc not shared

Adrian's Holovaty sums up the issues in his article:

> *If a native app opens a third-party website in a WebView, that third-party site will begin a new session, without existing cookies.* ***It's effectively like using a web browser's private (aka "incognito") mode.***
>
> *This means: If you're logged into a website in your phone's browser, but you click a link to that site from a native app's WebView,* ***your logged-in state will not be honored. You'll need to log in all over again.***
>
> *At best, this is irritating. At worst, it* ***gives people the false impression that the website is broken*** *or logged them out.*
> ****[[It is worth noting that state is no longer shared between Safari and SFSafariViewController since iOS 11, we cover this in this section.]]***
>
> *A specific example is Soundslice's Instagram account, where we highlight stuff people have created on our platform. If you see something on our Instagram and want to add it to a practice list in your Soundslice account, you quickly run into friction when opening the link within the Instagram app.* ***Even if you're logged into Soundslice in your phone's browser, the Instagram app doesn't honor your Soundslice login.***
>
> *[Instagram is a particularly heinous example, because it doesn't even let you add a link to a post.* ***If you enter a URL in an Instagram post, it will not turn into a clickable link.*** *Only one link, the one in your channel bio, is clickable. An entire cottage industry has formed around this "link in bio" madness.]*
>
> *Yes, you can copy-and-paste the URL (if the WebView displays it), or you can choose an "Open in web browser" option (if the WebView provides it). But either case requires nontrivial technical sophistication. Most users would just say "Aw, man, I've somehow been logged out of Soundslice" and assign the blame to us.*
>
> *Proponents of native apps would likely argue "But it's a better user experience from the perspective of the native app! Because the user doesn't have to context-switch into a different environment, aka the web browser."* ***There was indeed a time when this argument made sense: the years before 2015, which is when iOS 9 introduced a global Back button conveniently solving the problem****. And of course Android has its global Back button.* ***These days this argument holds no water.***
>
> *Proponents of native apps would also likely say: "If you had your own native app, this problem would be solved, because you can register a link handler that will*

> *automatically open all soundslice.com URLs in your native app." iOS calls this Universal Links; Android calls it App Links. This is true. It's also an unreasonable, disproportionate demand.* ***I shouldn't have to develop a native app simply to wrangle control over how my website's links are treated.***
>
> *Adrian Holovaty*
> *(emphasis added)*
> ***OWA ANNOTATIONS***

The issue is that the users' chosen browser is also the custodian of a wide range of data that are essential for the normal functioning of websites including:

1. Autofill Data (for auto-filling forms)
2. Payment Data
3. Content/Ad Blockers
4. Login Data for various websites
5. Session data and local storage (for locally stored data)
6. Browser cache (for performance)
7. Permissions that restrict or allow access to particular features like notifications, locations etc

For "Business Users", aka the Web App / Website owners this causes a wide range of issues including:

- **Reduced user engagement**
  When in-app browsers break website functionalities like logins, auto-fill users become frustrated and abandon tasks, lowering engagement and conversions.

- **Negative brand perception**
  Users blame websites for being "broken" or experiencing "unexpected logouts" when the issue lies with in-app browsers, harming brand reputation and trust.

- **Unfair Playing Field**
  To overcome in-app limitations, website owners may be forced to develop separate mobile apps or implement workarounds, increasing development costs and complexity. Websites then face disadvantages compared to native apps that can seamlessly handle logins and user data within their own environment.

For Consumers / End Users this causes issues such as:

- **Frustration and inconvenience**
  Users encounter unexpected hurdles like being "logged out" or unable to complete tasks seamlessly, leading to frustration and abandonment.

- **Disrupted Workflows**
  Frequent logins, manual form-filling, and inability to utilize browser extensions disrupt user experience and create unnecessary friction.

- **Inefficiency**
  Copying and pasting URLs or switching between apps to open links adds extra steps and slows down users' online activities.

Overall, the lack of state sharing between in-app browsers and users' default browsers creates a fragmented and frustrating online experience for both business users and end users, hindering usability, engagement, and the overall web browsing experience.

4.1.8. Benefit for the user?

It is not clear that webview and bundled engine in-app browsers provide meaningful benefits to users in the modern era. The early justifications for these systems were premised on the slow, memory-poor devices of the early 2010s. On those phones, tapping a link might put the app in the background where it would be killed by the OS for lack of memory. Tapping “back” to the app from the OS could take many taps, and if the OS had killed the app, it may have taken many seconds for the user to return to what they were doing. This was a bad experience worth avoiding.

By contrast, nearly all smartphones sold in the EU today offer gigabytes of memory and built-in remote tab in-app browser systems, removing the primary user-benefit justification for choice-disrespecting systems.

While proponents may tout potential benefits such as improved security or extra features, these assertions lack compelling evidence and often fail to withstand scrutiny when juxtaposed with the robust capabilities of established, standalone browsers that the user has actually chosen.

In response to Felix Krause’s question, “*If Meta built a whole system to inject JavaScript code (pcm.js) into third party websites to respect people’s App Tracking Transparency (ATT) choices, why wouldn’t Instagram just open all external links in the user’s default browser? This would put the user in full control over their privacy settings, and wouldn’t require any engineering effort on Meta’s end.*”, Meta made the following statement:

> *"As shared earlier, **pcm.js is required to respect a user's ATT decision.** The script needs to be injected to authenticate the source and the integrity (i.e. if pixel traffic is valid) of the data being received. Authentication would include checking that, when data is received from the In App Browser through the WebView-iOS native bridge, it contains a valid nonce coming from the injected script. SFSafariViewController doesn't support this. **There are additional components within the In App Browser that provide security and user features that SFSafariViewController also doesn't support.**"*
>
> Meta
> *(emphasis added)*

TikTok replied:

> *"Like other platforms, we use an in-app browser to provide an optimal user experience, but the Javascript code in question is used only for debugging, troubleshooting and performance monitoring of that experience — **like checking how quickly a page loads or whether it crashes**,"*
>
> Maureen Shanahan - TikTok Spokesperson
> *(emphasis added)*

This is, of course, absurd. There is no reason why TikTok (a social media short video sharing program) needs to monitor the performance of third party web pages any more than it needs to monitor the performance of other third party apps on iOS or Android, particularly considering the privacy implications of such an intrusion.

The purported benefits of webview and bundled browser engine in-app browsers appear tenuous and contended, while the costs to users and businesses remain persistently high.

While it is absolutely "technically" possible to create an excellent in-app browser, we are still waiting for a good example, 15 years after the introduction of in-app browsers. In-app browsers *could* provide compelling additional features or security, but there is no evidence that any of the widely-used in-app browsers in the market today meet this bar.

Any company willing to put in that level of effort, could simply produce, market and distribute a browser which users could freely choose. The fact that they choose not to do this, but instead silently subvert user choice, is telling. Major native apps could offer users a genuine and prominent choice, touting the benefits of their in-app browser. Instead, they resist even this basic transparency and accountability.

Indeed, as outlined above, a comprehensive assessment of webview and bundled engine in-app browsers reveals ongoing harms to users and businesses. Their features pale in

comparison to dedicated browsers, often lacking essential extensions, customization options, and advanced privacy controls. Furthermore, technical shortcomings, bugs and missing features can manifest in an extremely poor web experience.

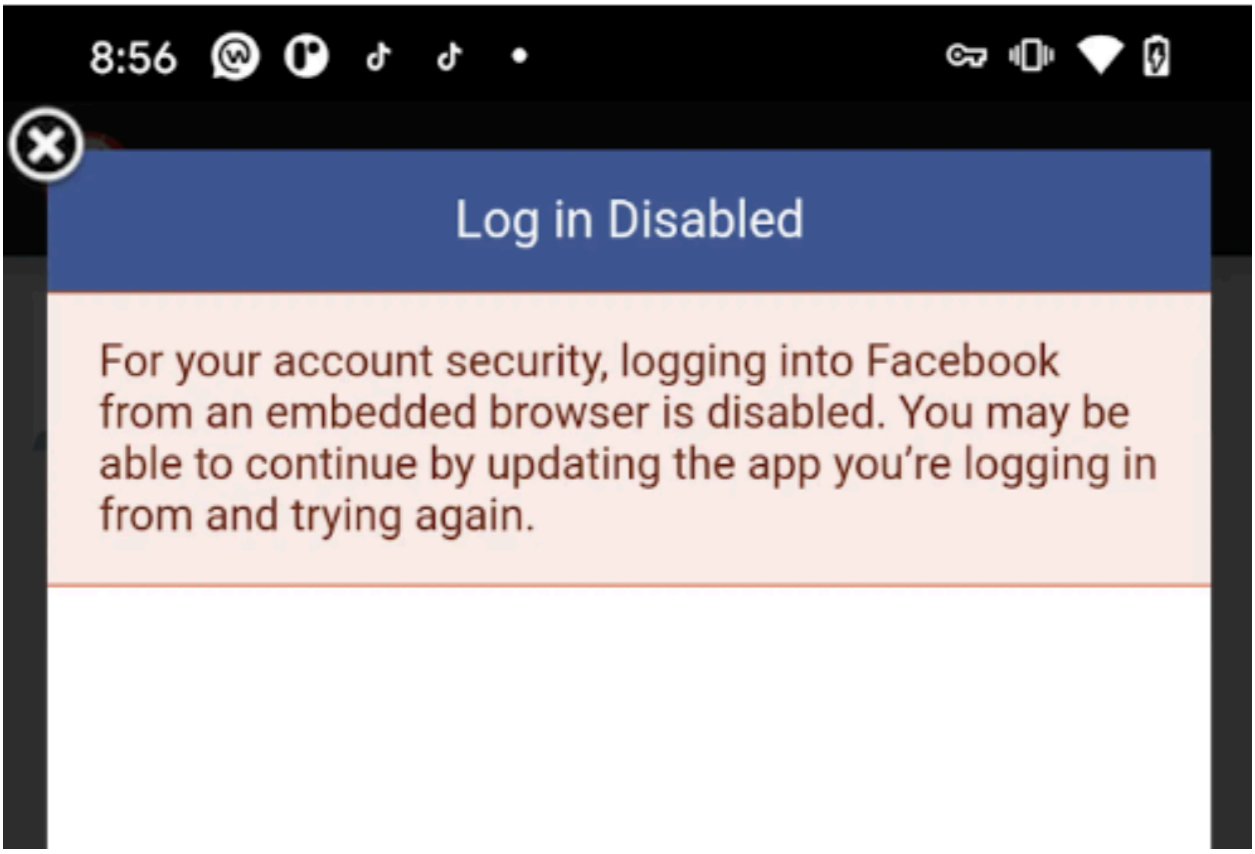
Moreover, the potential for in-app browsers to obfuscate data collection practices and impede user control over their online privacy represents a significant concern. Constraints on access to browser settings and the potential injection of app-specific tracking scripts raise troubling questions regarding user autonomy and informed consent.

The absence of tangible advantages, coupled with the aforementioned drawbacks, casts a shadow of doubt over their overall value proposition to users.

Amusingly Meta has deprecated login to facebook via webview in-app browsers on Android.

> *"Beginning October 5, 2021, Facebook Login will no longer support using Android embedded browsers (WebViews) for logging in users. To avoid a disrupted user experience, please use the following checklist to use **[Android]** Custom Tabs instead"*
>
> Meta for Developers Documentation
> ***OWA ANNOTATIONS**

Users will see the following error message:



4.1.9. Benefit for third-party websites/Web Apps?

In-app browsers also introduce a range of technical challenges and compatibility issues that negatively impact the user experience of third-party websites. These websites often express a clear preference for rendering within a user's default browser, where full functionality and performance can be ensured.

As discussed, key issues include missing features, unique bugs, unwanted JavaScript injection, security/privacy issues and difficulty debugging. These are not trivial matters for these websites to deal with and can completely break their product from a source that generates considerable traffic.

It seems clear that almost all major websites would opt-out of custom in-app browsers hijacking their interactions with the user and opt-in to be rendered using the user's chosen default browser.

4.1.10. Why do apps build these WebView In-App Browsers?

As discussed above, the historical reasons for the development of webview in-app browsers concerned user experience and device landscape questions. Thankfully, the introduction of remote-tab in-app browser systems and the progress of smartphone hardware have largely removed these concerns from consideration. Unfortunately, the market is left with many native apps that have not upgraded to these better alternatives, and may have come to exploit their powerful position as web traffic gatekeepers to the detriment of users.

Product managers at these firms may be optimizing for their own benefit instead of users, as in-app browsers:

- **Make it harder for users to leave the app (increasing engagement and time spent on the app metrics).**

  SFSafariViewController and Android Custom Tabs always display a button allowing the user to jump into their browser. Webviews do not have an interface, so native app developers are free to create one without such a button, preventing the user from leaving the app, or they can hide the open browser button in a submenu.

  *"But more often than not, **in-app browsers are simply there to keep you from leaving**. Roughly speaking, the more time you spend in an app, the more money its developers can make, especially when ad views are at stake."*

  Napier Lopez - thenextweb.com
  (emphasis added)

- **The ability to inject JavaScript into these third-party web pages.**

  *"All **apps that use SFSafariViewController or Default Browser are on the safe side**, and **there is no way for apps to inject any code onto websites**, even with the new WKContentWorld system."*

  Felix Krause - Security & Privacy Researcher
  (emphasis added)

- **Grants the ability to monitor all subsequent page visits, including duration spent on each page.**

  This does not require JavaScript injection and could be as simple as logging the reading time of external websites and which links the user then follows. This is

trivial for a webview in-app browser to do. This data by itself can be extremely private and for this reason is encrypted by browsers on any site supporting https (most websites these days). For example **there is a distinct difference in knowing a user visited wikipedia a number of times and having a list of the articles they read**, or as another example which facebook profiles they were viewing.

- **Prevent users from using ad-blockers or tracker blockers.**

*"An industry source described WebView browsers to The Register as '**tracker-blocker-blockers**' while noting the salient issue is what this does to user choice."*

The Register
(emphasis added)

- **Improve targeted advertising.**

*"In response to Krause's earlier report, Meta justified the use of these custom tracking scripts by claiming that users already consent to apps like Facebook and Instagram tracking their data. **Meta also claims that the data retrieved is only used for targeted advertising** or unspecified 'measurement purposes.'"*

Jess Weatherbed - The Verge
(emphasis added)

4.1.11. Don't compete as a Browser

"Businesses that face effective competition dare not raise prices, or cut down on quality standards, for fear of losing customers to their competitors (and so losing money)"

Dr Michael Grenfell

Normally, healthy browser competition would rectify issues regarding missing features, privacy issues, and bugs.

Importantly, none of these webview in-app browsers actually contest the browser market. They do not need to market their features (traffic is "free"), or fear replacement (browser choice cannot unseat them). Few users are even aware they are using them. Branding is strikingly absent from webview and bundled in-app browsers. Few logos or names alert users they are not using their default browser.

Given the effort each of these companies go to develop these user interfaces, it seems unlikely this is an oversight. In essence, users are being lulled or tricked into handing over their browsing experiences to these companies.

In our view it is exceedingly likely that given a direct and clear choice on whether or not to grant these apps the permission to override their choice of default browser, that a significant majority of users would vote no.

Facebook after all has argued for years that they can use the data they collect to provide their users a better service.

> *"**Data also helps us show you better and more relevant ads.** And it lets advertisers reach the right people, including millions of small businesses and non-profits who rely on Facebook every day to reach people that might be interested in their product or cause. Data lets a local coffee shop survive and grow amid larger competitors by showing ads to customers in its area. And it lets a non-profit promote a diabetes fundraiser to those interested in the cause."*
>
> Rob Goldman - Facebook's vice president for ads
>
> (emphasis added)

> *"Facebook bombarded its users with messages begging them to turn tracking back on. It threatened an antitrust suit against Apple. It got small businesses to defend user-tracking, claiming that when a giant corporation spies on billions of people, that's a form of small business development."*
>
> Cory Doctorow - Electronic Frontier Foundation

Yet when Apple introduced app tracking transparency settings in iOS 14.0 on September 16, 2020, explicitly asking users if they would like the app to track them for the purposes of advertising, an estimated 75% of users opted out.

4.2. Problems with Bundled Engine In-App Browsers

Bundled browser engine in-app browsers exhibit most of the issues of webview in-app browsers, including:

- They override the users choice of default browser
- They raise tangible security and privacy harms
- They stop the user from using their ad-blockers and tracker blockers
- Their default browsers privacy and security settings are not shared
- They are typically missing web features
- They are typically have many unique bugs and issues
- The users session state is not shared so they are booted out of websites they have logged into in their default browser
- Provide little benefit to users
- Create significant work and often break third-party websites
- Don’t compete as browsers
- They confuse users and today function as dark patterns.

A unique concern for bundled in-app browsers is the funding and pace for security engineering. Like true browsers, in-app browsers are confronted with the same hostile web, but by moving maintenance and security testing out of a dedicated browser team, the quality of bundled in-app browsers may suffer, if only because their release cadence may be tied to the launch of the hosting app, rather than their own imperative to protect users.

Facebook has announced plans to replace their webview in-app browser with a bundled engine in-app browsers. In the same post they state the primary reason for the switch is security, stability and performance.

Some in the web development community are not convinced:

> *"The timing of this change seems a bit dubious, too, as the teams building WebViews started discussing IAB security and privacy risks, and whether and how they should block IAB-rendering if the site requests it. There is no consensus yet, but they are exploring options as described in the previous chapter. If Facebook would continue to use the WebView to render IAB content, they would have no control over whatever restrictions WebViews may implement. By shipping their own engine, they can circumvent any solutions the WebView teams come up with. They could ignore a site's request to be opened in a proper browser, keep injecting javascript into sites they don't own, and continue their business of spying on users who are almost certainly unaware of the fact that they didn't leave Facebook. They would have almost unlimited access to whatever happens in their IAB window.*
>
> *So, basically, Facebook isn't really solving anything here. At best, they're doing the super difficult thing of maintaining and shipping a fork of an entire rendering engine instead of using what's safer, a better UX and already provided for: Chrome Custom Tabs.* ***They are actually making things worse by granting themselves more control and, to me, it just feels like an attempt to pro-actively circumvent any protections that WebViews may implement in the future.****"*
>
> Roderick E.J.H. Gadellaa - Designer, front-end developer

Given all this, the simplest and cheapest solution that provides the most benefit to the user would be to let them use their default browser either directly or via a remote tab in-app browser.

4.3. Problems with SFSafariViewController

In many aspects, SFSafariViewController is vastly superior to all webview and bundled engine in-app browsers. It does not share all of their myriad of problems outlined above.

4.3.1. Locked into iOS Safari

The primary issue with SFSafariViewController is that it is locked into using Safari, as opposed to the user's default browser.

This is perhaps not entirely surprising, as up until September 16, 2020 only Safari could be the default browser, even today third-party browsers can not effectively compete on iOS.

However, as this looks set to change due to legislation like the Digital Markets Act and others compelling Apple to allow companies to port their real browsers along with their browser engines, it is critical that this flaw is fixed.

In-app browsing is an incredible amount of traffic from non-browser apps. The job of the default browser is to handle and render clicks/taps on http/https links in non-browser apps. OS gatekeepers such as Apple should not be able to override the user's choice of default browser for such an important source of traffic.

It is important that SFSafariViewController be upgraded to serve as a proxy for *any* compatible system default browser so that the benefits it provides to Safari default users today can be shared by users who set different browsers as their preferred user agents.

Apple's latest proposal to be in compliance with the DMA takes no steps to fix this.

4.3.2. SFSafariViewController and Safari have Separate Siloed States

SFSafariViewController silos state (website data such as cookies, indexedDB etc) from Safari and operates as its own environment, breaking websites and creating a "forgetful web".

On launch in 2015, SFSafariViewController and iOS Safari shared state. This meant users logged into a website in iOS Safari you would be logged in when you visited it via an app in SFSafariViewController, which made sense as it was invoking your default browser (Safari was the only possible default browser at the time) and loading it as an in-app browser.

Mechanically this means that various forms of data each website stores locally such as cookies and indexedDB (a local database) were shared between Safari and SFSafariViewController for that specific domain name.

In iOS 11 (2017), Apple changed the behavior of SFSafariVewController so that state would no longer be shared between SFSafariViewController and iOS Safari. This damages complex websites' ability to interact with and retain users as those users.

This is combined with Apple's 2020 decision to delete all website data on any site that hasn't been visited for more than 7 days. Importantly Web Apps installed by Safari are exempt from this rule. This means those users will be logged out of those sites every 7 days if they have not installed the Web App.

While it could be argued that this stance is too aggressive (i.e a longer period might be more reasonable), this was in response to genuine privacy concerns related to targeted advertising.

> *"It's the direction all other web browser vendors want to pursue, eventually. It was a long fight of the previous decade, but we're slowly arriving there. The big questions*

> *are how the future web architecture will look like. We should hope that Apple will want to share their experiences with the broader web community."*
>
> Dr Lukasz Olejnik - independent privacy researcher and consultant

The issue here is that Apple is very willing to break functionality on the Web for the sake of privacy but unwilling to make equivalent changes for Native Apps sold by its app stores. This combined with its effective ban on third party browsers on iOS (who might choose other trade-offs) and its hiding of the user interface for installing Web Apps leads to a suppression of the Webs ability to compete effectively and fairly with their Native ecosystem from which they extract a 15-30% cut in addition to developer fees.

> *“Our investigation found the iPhone’s tracking protections are nowhere nearly as comprehensive as Apple’s advertising might suggest. We found at least three popular iPhone games share a substantial amount of identifying information with ad companies, even after being asked not to track.*
>
> *When we flagged our findings to Apple, it said it was reaching out to these companies to understand what information they are collecting and how they are sharing it. After several weeks, nothing appears to have changed.”*
>
> Geoffrey Fowler And Tatum Hunter - Washington Post

OWA does not believe that any browser vendor has the perfect policy regarding privacy. Indeed, it is an area of active competition between browsers. Instead of forcing a single policy on all users, OWA believes that the best solution for the ecosystem and for users is for SFSafariViewController to respect user choice in default browser, and through that choice, delegate questions related to state splitting or joining to the user’s browser.

4.4. Problems with Android Custom Tabs

Android Custom Tabs (ACT) is a near-ideal solution. An OS provided component that invokes the users default browser when clicking on http/https links in non-browser apps.

Android Custom Tabs (“ACT”) were previously known as Chrome Custom Tabs (“CCT”). The 2020 name change was welcome, as the system has always been able to invoke other browsers (e.g., Firefox).

It does not share the myriad of problems of webview and bundled engine in-app browsers.

It does however have one significant flaw, app developers are able to hard-code ACT to use a specific browser. OWA believes the most prominent example of this is the Android Google Search App which is locked to Google Chrome. This is a clear example of undermining the user's choice of default browser.

While the google.com search page is first-party and second-party content when a user clicks on a link on the google.com search page to a third-party website this should open in a remote tab in-app browser (i.e Android Custom Tabs), using the users default browser or in the users default browser directly.

The ability to hard code Android Custom Tabs to particular browsers should be removed in favor of exclusively respecting the users choice of default browser.

If the Android Google Search App wishes to compete as a browser it should do so explicitly. Although, given the high usage, it would almost certainly be declared a designated core platform service (as a browser) under the digital markets act.

4.5. Do users need In-App Browsers at all?

It's not clear that in-app browsers are necessary on modern smartphones, even those that respect the user's choice of default browser. With seamless back button functionality readily available within apps, the additional layer of abstraction offered by remote-tab in-app browsers may not be necessary.

There are two remaining arguments in favor of remote tab in-app browsers that invoke the user's default browser, as opposed to just opening in the default browser directly.

First, when jumping to your browser, the user mentally "leaves" the app, and goes into another context. The user may very well forget about the previous context they left. The relatively small button to go back to that previous context is easily forgotten in this case.

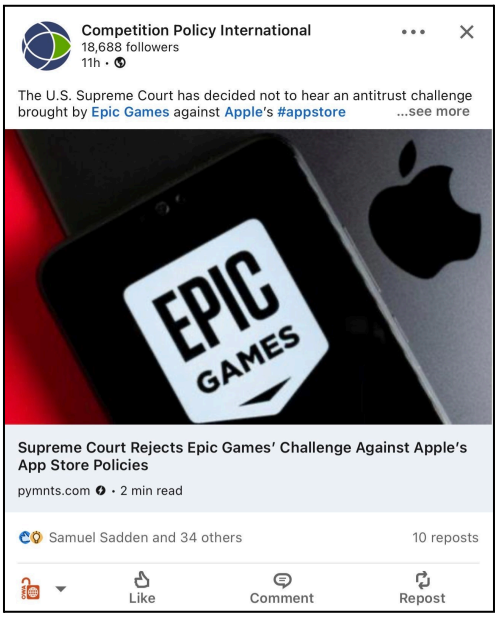
Second, on cheap android phones with particularly low memory, using such an in-app browser reduces the chance that the hosting app is cleared from the memory. This is not an issue on phones with larger amounts of memory.

It is worth noting that it is standard in desktop environments to open in the user's default browser directly. For example, clicking on a http/https link in a word document does not open in a word in-app browser, and such a future is not desirable. That said, there are differences between the screen size, user interface designs and this choice should likely be left to consumer preference on mobile devices.

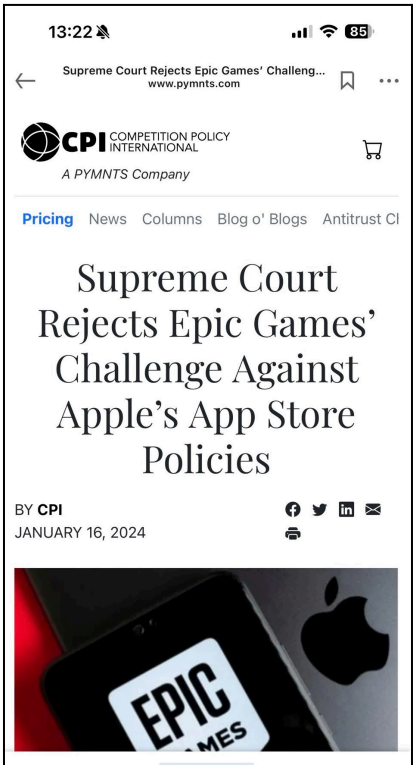
As such, OWA would support an operating system level setting allowing users to globally disable in-app browsers and instead always jump out directly to the user's default browser.

4.5.1. Customization and the LinkedIn Example

Bundled engine in-app browsers, webview in-app browsers and remote-tab in-app browsers all provide facilities for displaying extra UI elements around third-party content, and native app developers sometimes claim that this customization is beneficial to users. To evaluate this claim, it's helpful to examine the sorts of customizations that have been developed over nearly 15 years of in-app browsers. A relatively sophisticated example comes from iOS LinkedIn's webview-powered in-app browser, which subverts browser choice for users tapping on links from within the app:



Inside LinkedIn's app (prior to clicking on a link) an image preview and title of an article are shown along with buttons to like, comment and repost.



When the user then attempts to visit the link, rather than opening in the user's default browser, the website is rendered in a webview in-app browser. It provides buttons to like, comment, and repost within the user's LinkedIn feed.

Because this in-app browser fails to provide essential re-engagement features to the websites loaded within it (Push Notifications, PWA install, etc.), and because it forgets user login state to news sites, these

customizations stack the deck, favoring LinkedIn's interests over the user and the site publisher.

Given that this additional UI (like, comment, repost buttons) is available in the LinkedIn app where one can see a snippet of the original page/article, the marginal benefit to the user of having access to that same UI on every subsequent page visit does not justify overriding the users choice of default browser and the ability to inject content into third party websites.

These concerns would be somewhat reduced if LinkedIn's customizations were presented within a remote tab in-app browser context, deferring to the user's chosen browser (as LinkedIn for Android does) rather than a broken webview environment.

5. Remedies

We propose 6 remedies:

1. Designated Core Platform Services should respect the users choice of default browser.

2. App Store rules must mandate non-browser apps use the user's chosen default browser for http/https links to third-party websites/Web Apps.

3. Apple must update SFSafariViewController to respect the user's choice of default browser.

4. Third-party businesses must be provided an explicit and effective technical opt-out from non-default in-app browsers.

5. OSes must provide a global user opt-out. Apps must also request explicit permission from users.

6. Google must remove the ability to override the users choice of default browser via Android Custom Tabs.

These remedies are overlapping and cover different aspects of the same underlying problem, that is, gatekeepers either:

- overriding the users choice of default browser
- creating a system that by design facilitates third-party non-browser apps in overriding the users choice of default browser.

Remedy 1 is a narrow scope version of Remedy 2 but only for non-browser apps that have themselves been declared designated core platform services of gatekeepers.

Remedy 4 and Remedy 5 are more qualified versions of Remedy 2. If Remedy 2 is effectively and strongly enforced then remedy 4 and 5 are not needed.

5.1. Designated Core Platform Services should respect the User's Choice of Default Browser

The Digital Markets Act states that gatekeepers must not impose browser engines on their end users or business users. And yet, many of these gatekeepers effectively do this by ignoring the user's choice of default browser (a choice mandated by the act) and instead capturing http/https links leaving their apps with their own in-app browser. Due to suppressed user awareness via dark patterns, the technical complexity of the subject matter, and the neglect of Apple and Google, these companies have been able to continue this practice for years.

While disadvantageous to users, ignoring the users choice of default browser has a number of advantages for native app developers, particularly with regards to targeted advertising.

OWA believes that at a very minimum apps that are designated core platform services on iOS and Android should be prohibited from this practice and should be required to use the user's chosen default browser either directly, or via a remote tab in-app browser for http/https links to third-party websites.

A number of apps by gatekeepers already comply with this rule.

OWA tested the current compliance with this remedy:

Gatekeeper	App	Respects users choice of default browser
Apple	iOS App Store	Yes (default browser)
Amazon	Amazon (iOS)	No links to third party content found
Amazon	Amazon (Android)	No links to third party content found
Bytedance	TikTok (iOS)	No (webview in-app browser)
Bytedance	TikTok (Android)	No (webview in-app browser)
Google	Google Maps (iOS)	No (SFSafariViewController)
Google	Google Maps (Android)	Yes (default browser)
Google	Google App (iOS)	No (webview in-app browser)
Google	Android Google Search App (Android)	No (remote tab in-app browser set to Chrome)

Google	Youtube (iOS)	No (SFSafariViewController)
Google	Youtube (Android)	Yes (remote tab in-app browser, ACT)
Google	Google Play Store	Yes (default browser)
Meta	Facebook (iOS)	No (webview in-app browser)
Meta	Facebook (Android)	No (bundled engine in-app browser)
Meta	Instagram (iOS)	No (webview in-app browser)
Meta	Instagram (Android)	No (bundled engine in-app browser)
Meta	Whatsapp (iOS)	Yes (default browser)
Meta	Whatsapp (Android)	Yes (default browser)
Meta	Messenger (iOS)	No (webview in-app browser)
Meta	Messenger (Android)	No (bundled engine in-app browser)
Microsoft	LinkedIn (iOS)	No (webview in-app browser)
Microsoft	LinkedIn (Android)	Yes (remote tab in-app browser, ACT)

These apps were tested on 16th January 2024, it is possible that the results may vary based on region, operating system version etc. For this remedy, apps that make use of SFSafariViewController are listed as “No” as SFSafariViewController currently does not respect the users choice of default browser.

5.2. App Store Rules

This remedy is to compel operating system gatekeepers (Apple and Google) to add App Store rules that force non-browser apps to respect the user’s choice of default browser when users navigate to third-party web content.

Apps not classified as browsers would be required to use the system provided in-app remote tab browser or to open in the default browser directly when navigating to third-party content. Apple would need to update SFSafariViewcontroller to respect the user's choice of default browser in order to support this remedy.

Gatekeepers should be required to add conforming app store policies, demonstrate enforcement, and provide evidence to the commission of conformance.

Gatekeepers would also need to abide by this rule for their own apps.

One possible technical enforcement mechanism of this remedy could be based on Apple's App-Bound Domains. We outline in more detail here.

5.3. Update SFSafariViewController to respect the users choice of default browser

The Commission should compel Apple to update SFSafariViewController so that it respects the users choice of default browser rather than being locked to iOS Safari.

Currently iOS does not have a system provided in-app remote tab browser that invokes the default browser. The simplest compliance solution would be to upgrade SFSafariViewController to support this functionality as then all non-browser apps currently using it would then respect users choice of default browser without requiring updated code.

5.4. Third-Party Business Opt-out

Third-party businesses should be allowed to opt-out of being displayed in the apps in-app browser.

Business Users that run websites or web apps can insert either a meta-tag or header that indicates to the app (and the operating system) that they only wish to be displayed in the user's default browser.

While each of these are slightly different, they are in essence all just a mechanism by which the server delivering the html for the website/Web App can indicate the opt-out.

One obvious concern here with second party content is that websites could be silently tricked into granting permission for the non-browser app to use its own webview or bundled engine in-app browser.

For example a clause could be added to the contract for some service that the website is using (e.g. advertising or analytics) which grants the gatekeeper the ability to do this. Allowing the website to opt-out via technical means is a powerful tool to prevent business users unwittingly agreeing to this, as the gatekeepers only recourse would be to sue the website for breach of contract. Such a contract would also likely be in breach of the DMA as it would be imposing a browser engine on the business user.

Reversing the onus here is important.

Operating system gatekeepers would then need to enforce respecting this opt-out via app store rules and to abide by it for their own non-browser apps.

Apple and Google should document and implement this opt-out technical mechanism/s in their own browsers either separately or as a new web standard. It is worth noting that the Web has a number of effective standardization bodies and that this is a voluntary process by browser vendors. The DMA should not seek to create or impose web standards; rather what is important here is that this functionality exists even if it works differently in Safari and Chrome. In general the precise details should be left to Apple and Google to implement. Additionally non-gatekeeper browsers will be under non-obligation to implement or include this functionality.

Finally, we believe that Apple and Google should voluntarily broaden the rule to include bundled browser engine in-app browsers.

5.5. Global User Opt-out and per App Permission Request

Users should be allowed to opt-out of using the apps own in-app browser in two ways.

First is a per-app permission for using the app's own in-app browser to manage http/https links to non-cooperating third-party websites.

A possible message could be:

> *Allow "App Name" to use its own in-app browser instead of your default browser:*
> *Allow Once*
> *Allow*
> *Don't Allow*

Second, a global user opt-out blocking apps from asking for this permission to stop every new app bothering the user if they have already decided they wish to always use their default browser for http/https links to non-cooperating third-party websites.

A possible settings page could be:

> ***Allow Apps to request to use its own in-app browser [Toggle Button]***
> *Allow Apps to request to use their own in-app browser instead of your default browser.*

When this is off, all new requests by apps to use their own in-app browser instead of your default browser will be denied.

Apps that have asked for permission to use their own in-app browser will appear here.

This per-app opt-out paired with a global opt-out has a precedent on iOS with the app tracking transparency settings that iOS introduced in iOS 14.0 on September 16, 2020.

Operating system gatekeepers would then need to enforce respecting these settings via app store rules and to abide by it for their own non-browser apps.

5.6. Removing the ability to override the users default browser in Android Custom Tabs

The Commission should compel Google to update Android Custom Tabs so that it respects the users choice of default browser.

Currently in Android Custom Tabs, the app developer using the component can lock it to use a particular browser other than the user's chosen default browser. This overrides using the user's default browser for non-cooperating third-party websites.

Gatekeepers should be prohibited from allowing in-app remote tab browsers to invoke only a specific browser, except in extreme circumstances (e.g. failure by other browsers to load). This remedy mirrors updating SFSafariViewController so that it no longer is locked to Safari, promoting competition by ensuring that high-volume web traffic sources respect browser choice by default.

6. These changes are beneficial to consumers and businesses

These changes will benefit both consumers and business users.

Consumers benefit through:

- **Respected choice of default browser**
  Users can navigate the web with their chosen tool, empowered by familiar settings. This choice is critical to competition between browsers, leading to more performant, more feature rich, more stable and more private browsers.

- **Blocking spying/manipulation of interactions with third-party websites**
  With their default browser choice respected, users can leverage the sophisticated security and privacy controls they've invested in, blocking trackers, safeguarding data, and ensuring their online interactions remain private.

- **Settings and preferences from their default browser**
  Leads to a web experience where user preferences seamlessly migrate across applications, eradicating the friction and frustration associated with fragmented browsing contexts. It's worth noting here that these pseudo browsers offer few of the settings that the user's chosen default browser does.

- **Have access to extensions**
  Users have access to all the extensions they have installed in their chosen default browser, including those that block adverts and unwanted tracking.

- **A significantly better quality of web experience**
  User-chosen browsers ensure consistent, stable rendering across all websites, seamless functionality, and unlocking access to all the latest web features. It ultimately delivers a demonstrably superior, more personalized, more private and more secure web experience.

Companies, particularly SMEs that rely on the web, benefit from:

- **Reduced Development Burden**
  No more adapting to bugs or missing features in inconsistent in-app browser environments. Companies can focus on building websites and web apps that function flawlessly within any user-chosen browser, streamlining development and ensuring a consistent experience for everyone.

- **Blocking spying/manipulation of interactions with their users by these apps**
  In-app browser environments can potentially spy on user interactions with third-party websites or inject trackers for targeted advertising. By enforcing default browser choice, companies can be confident that their users' journeys and data remain secure, protected from potential manipulation or data collection by other apps.

- **Enhanced security and privacy for their users**
  Companies no longer have to worry about potential security lapses within in-app browser environments, creating a safer and more trustworthy online experience.

- **Higher quality of experience for their users**
  Consistent rendering, seamless functionality, and familiar user interfaces will translate to happier users and increased engagement.

7. Towards a Brighter Future

The EU's Digital Markets Act core aim is to enforce rules that allow the proper functioning of the internal market to ensure contestability and fairness in the digital sector for both business users and end users.

Central to this vision of robust competition is the ability of users to exercise genuine choice through frictionless and transparent mechanisms. Obfuscation or manipulation of user preferences hinders the very foundation of a healthy digital marketplace.

Recognizing this critical principle, the DMA specifically addresses the crucial role of web browsers in navigating the online terrain. It mandates the implementation of a dedicated choice screen on both Android and iOS operating systems, allowing users to freely select their preferred default browser. This seemingly straightforward measure holds immense significance in dismantling pre-installed monopolies and empowering users to actively engage in the shaping of their digital experience.

In-app browsers circumvent this choice by completely ignoring it in a manner that is not clear to end users. The continued existence of opaque and restrictive in-app browsers poses a significant challenge to this vision. Their inherent lack of transparency regarding data practices and limited functionality compared to established browsers significantly damages users' web experience. Additionally, the sheer ubiquity of apps employing in-app browsers poses a potential barrier to entry for competing web browsers, further entrenching existing market monopolies and hindering contestability.

The pervasiveness of in-app browsers poses a direct threat to the rights and visibility of third-party websites. By forcefully intercepting web links and routing users through captive browsing environments, in-app browsers create an environment rife with potential for the injection of unauthorized trackers, the introduction of unintended bugs, and the absence of critical features found in standard browsers. This results in a compromised web experience that not only undermines user trust and engagement but also hinders the ability of third-party websites to effectively deliver their services and content. Such practices create an uneven playing field, contravening the DMA's core principles of fair competition and equitable access within the digital marketplace.

Therefore, addressing the issues of in-app is not merely a matter of enforcing user choice for default browsers. It represents a crucial step upholding the DMA's core principles of contestability and fairness in the digital market.

Competition, not walled gardens, leads to the brightest future for EU’s consumers

8. Open Web Advocacy

Open Web Advocacy is a not-for-profit organization made up of a loose group of software engineers from all over the world, who work for many different companies and have come together to fight for the future of the open web by providing regulators, legislators and policy makers the intricate technical details that they need to understand the major anti-competitive issues in our industry and potential ways to solve them.

It should be noted that all the authors and reviewers of this document are software engineers and not economists, lawyers or regulatory experts. The aim is to explain the current situation, outline the specific problems, how this affects consumers and suggest potential regulatory remedies.

This is a grassroots effort by software engineers as individuals and not on behalf of their employers or any of the browser vendors.

We are available to regulators, legislators and policy makers for presentations/Q&A and we can provide expert technical analysis on topics in this area.

For those who would like to help or join us in fighting for a free and open future for the web, please contact us at:

Email	contactus@open-web-advocacy.org
Web / Web	https://open-web-advocacy.org
Mastodon	@owa@mastodon.social
Twitter / X	@OpenWebAdvocacy
LinkedIn	https://www.linkedin.com/company/open-web-advocacy

9. Suggested Reading

We recommend reading the following articles on this topic:

Hobson's Browser - How Apple, Facebook, and Google Broke the Mobile Browser Market by Silently Undermining User Choice
https://infrequently.org/2021/07/hobsons-browser/

iOS Privacy: Instagram and Facebook can track anything you do on any website in their in-app browser
https://krausefx.com/blog/ios-privacy-instagram-and-facebook-can-track-anything-you-do-on-any-website-in-their-in-app-browser

iOS Privacy: Announcing InAppBrowser.com - see what JavaScript commands get injected through an in-app browser
https://krausefx.com/blog/announcing-inappbrowsercom-see-what-javascript-commands-get-executed-in-an-in-app-browser

In-App Browsers are tracking you
https://webventures.rejh.nl/blog/2022/in-app-browsers-are-tracking-you/

Let websites framebust out of native apps
https://www.holovaty.com/writing/framebust-native-apps/

10. Appendix

10.1. App Bound Domains

One possible technical enforcement mechanism of this remedy could be based on Apple's App-Bound Domains.

In iOS 14, Apple released a feature for WKWebView called App-Bound Domains making it possible for developers to offer a safer in-app browsing experience. Android currently does not support an equivalent. As an app developer, you can define a list of the domains your app can access, and all web requests will be restricted to them. Since the app-developer would need to submit this list of domains to the AppStore, any app requesting access to domains that they do not own/control/have a business relationship with could be rejected from the store.

The issue is App-Bound Domains are opt-in (by the app developer) and are not mandatory, meaning the vast majority of iOS apps don't use the feature. For them to be effective they would have to be mandatory.

The other issue is that many native apps take advantage of Ad networks. These Ad networks as discussed deliver ads via the web to webviews. These ad networks are complex and updating all of the infrastructure in each of them to support an explicit "opt-in" ad solution would be complex (although this could be a potential much longer term solution).

One possible solution to this issue is to provide developers of non-browser apps with a version of the webview (or a setting) that is to be used exclusively to render adverts. The standard webview for non-browser apps could then be locked exclusively to App-Bound Domains. This would be paired with rules against using the ad-webview for anything except adverts.

Additionally developers would need to provide proof that they control or have permission of the domain listed using App Bound Domains for the standard webview. This could be done automatically via technical mechanisms such as setting DNS records as this is commonplace for many other types of verification.

Importantly webview browsers, as well as browser apps, would need to be exempt from this rule. Apple's documentation for App Bound Domains already explicitly covers this.

Note, that there are likely various solutions along these lines and that the gatekeepers would be best placed to propose solutions that would solve these issues with close

scrutiny from regulators and others like Open Web Advocacy. We would recommend that the Digital Markets Act team propose this as one possible solution to the gatekeepers to get their viewpoints, and push the gatekeepers to develop solutions including working towards standardization in one of the existing standardization bodies.

11. References

Alex Russell - Hobsons browser - definition of a browser
https://infrequently.org/2021/07/hobsons-browser/#starting-points

Apple Developer Documentation - Should I use WKWebView or SFSafariViewController for web views in my app?
https://developer.apple.com/news/?id=trjs0tcd

Ionic Framework
https://ionicframework.com/

Mobile Rich Media Ad Interface Definitions (MRAID)
https://www.iab.com/guidelines/mraid/

Apple App Store Guidelines - Rule 2.5.6
https://developer.apple.com/app-store/review/guidelines/#:~:text=2.5.6%20Apps%20that%20browse%20the%20web%20must%20use%20the%20appropriate%20WebKit%20framework%20and%20WebKit%20Javascript.

Open Web Advocacy - Walled Gardens Report - Apple has effectively banned all third party browsers
https://open-web-advocacy.org/walled-gardens-report/#apple-has-effectively-banned-all-third-party-browsers

Apple Developer Documentation - WebView
https://developer.apple.com/design/human-interface-guidelines/web-views

Felix Krause - iOS Privacy: Announcing InAppBrowser.com - see what JavaScript commands get injected through an in-app browser
https://krausefx.com/blog/announcing-inappbrowsercom-see-what-javascript-commands-get-executed-in-an-in-app-browser

Felix Krause - iOS Privacy: Instagram and Facebook can track anything you do on any website in their in-app browser
https://krausefx.com/blog/ios-privacy-instagram-and-facebook-can-track-anything-you-do-on-any-website-in-their-in-app-browser

Andy Stone - Meta Spokesperson - On JavaScript that Instagram injects into third party websites
https://twitter.com/andymstone/status/1557825414176940035

The Register - Meta iOS apps accused of injecting code into third-party websites
https://www.theregister.com/2022/08/12/meta_ios_privacy/

Spying on the spies. See what JavaScript commands get injected by in-app browsers
https://www.malwarebytes.com/blog/news/2022/08/spying-on-the-spies-see-what-javascript-commands-get-injected-by-an-in-app-browser

Ad blocking definition
https://en.wikipedia.org/wiki/Ad_blocking

Tracking blocking definition
https://en.wikipedia.org/wiki/Web_tracking#:~:text=wasted%20marketing%20funds.-,Prevention,-%5Bedit%5D

Everything you wanted to know about surveillance advertising and how to avoid it
https://www.privateinternetaccess.com/blog/everything-you-wanted-to-know-about-surveillance-advertising-and-how-to-avoid-it/&sa=D&source=docs&ust=1705301733685818&usg=AOvVaw3113Yf1SD7aEw6sjA_DQy-

Adrian Holovaty - Let websites framebust out of native apps
https://www.holovaty.com/writing/framebust-native-apps/

Soundslice
http://www.soundslice.com/

Django
https://www.djangoproject.com/

TikTok's In-App Browser Includes Code That Can Monitor Your Keystrokes, Researcher Says
https://www.forbes.com/sites/richardnieva/2022/08/18/tiktok-in-app-browser-research/?sh=26236cdb7c55

Deprecating Facebook Login support on Android WebViews
https://developers.facebook.com/docs/facebook-login/android/deprecating-webviews/

Rant: In-app browsers are annoying and mostly useless
https://thenextweb.com/news/rant-app-browsers-annoying-mostly-useless

The Register - Don't mind Facebook, just putting its own browser in its Android app: Totally not for data collection
https://www.theregister.com/2022/10/04/metas_facebook_webview_chromium/

The Verge - This site exposes the creepy things in-app browsers from TikTok and Instagram might track
https://www.theverge.com/2022/8/19/23312725/in-app-browser-tracking-facebook-instagram-privacy-tool

Dr Michael Grenfell - Effective Competition
https://www.gov.uk/government/speeches/michael-grenfell-should-competition-authorities-intervene-in-digital-markets

Hard Questions: What Information Do Facebook Advertisers Know About Me?
https://about.fb.com/news/2018/04/data-and-advertising/

Cory Doctorow - Facebook Says Apple is Too Powerful. They're Right.
https://www.eff.org/deeplinks/2022/06/facebook-says-apple-too-powerful-theyre-right

Apple - Application Tracking Transparency
https://en.wikipedia.org/wiki/IOS_14#:~:text=Application%20Tracking%20Transparency

Statista - App tracking transparency: opt-in rate of iOS users worldwide 2022
https://www.statista.com/statistics/1234634/app-tracking-transparency-opt-in-rate-worldwide/#:~:text=Among%20those%20that%20have%20already,in%20rate%20had%20been%20predicted.

Meta announcing a new bundled engine browser for Android
https://engineering.fb.com/2022/09/30/android/launching-a-new-chromium-based-webview-for-android/

In-App Browsers are tracking you(r users)
https://webventures.rejh.nl/blog/2022/in-app-browsers-are-tracking-you

The Verge - iOS 14 and iPadOS 14 will let you set default email and browser apps
https://www.theverge.com/2020/6/22/21299342/apple-ipados14-default-apps-email-browser-choice-features-wwdc-2020

Apple announcing state will not be shared between SFSafariViewController and iOS Safari
https://www.branch.io/resources/blog/ios-11-safari-view-controller-cookie-passthrough-and-the-future-of-mobile-web/

The Register - Apple: Relax, we're not totally screwing web apps. But yes, third-party cookies are toast
https://www.theregister.com/2020/03/26/apple_relax_were_not_totally/

The Washington Post - When you 'Ask app not to track,' some iPhone apps keep snooping anyway
https://www.washingtonpost.com/technology/2021/09/23/iphone-tracking/

Apple Developer Documentation - App Bound Domains
https://webkit.org/blog/10882/app-bound-domains/