

With Great Power Comes Great Responsibility: Security and Privacy Issues of Modern Browser Application Programming Interfaces

Harun Oz, *Cyber-Physical Systems Security Lab, Florida International University*

Daniele Cono D'Elia, *Sapienza University of Rome, Italy*

Güliz Seray Tuncay, *Google, Mountain View, CA, USA*

Abbas Acar, *Cyber-Physical Systems Security Lab, Florida International University*

Riccardo Lazeretti, *Sapienza University of Rome, Italy*

Selcuk Uluogac, *Cyber-Physical Systems Security Lab, Florida International University*

Abstract—This article provides an overview of security and privacy challenges and concerns that come with modern browser application programming interfaces. We aim to inform the community about intrinsic risks associated with their usage and suggest possible directions to tackle them more effectively.

Index Terms: *Emerging technologies, Web Browser, Web Security, Browser Security, Browser API Security*

Since its inception in 1993, the World Wide Web has undergone a remarkable evolution, transitioning from a repository of static HTML pages to a dynamic and feature-rich ecosystem. This transformation has been largely driven by the introduction and integration of new technologies into the web ecosystem. A prominent example is the modern browser application programming interface (API), which has been instrumental in equipping developers with tools to create web applications that are not only dynamic but also feature-rich, rivaling the capabilities of traditional desktop applications [1].

Browser APIs enable web applications to interact with and retrieve information from the system's hardware components, manipulate data within both the browser and the user's local file system, and customize their user interface to facilitate more engaging user experiences. As web applications are platform-agnostic (e.g., operating system independent) and can be more cost-effective to maintain and update than some mobile or desktop applications, the introduction of modern browser APIs also motivates major companies to port their native applications to the web.

However, *with great power comes great responsibility* (https://en.wikipedia.org/wiki/With_great_power_comes_great_responsibility). While browser APIs en-

able the development of powerful web applications, they are a double-edged sword. On the one hand, they significantly extend the capabilities of web applications, leading to improved user experiences and the ability to fully utilize the underlying hardware and software infrastructure. On the other hand, the power and flexibility of these APIs introduce new security and privacy challenges. As web applications gain deeper access to system resources, these APIs may extend the attack surface and become potential vectors for various types of attacks.

While browser vendors are actively working to integrate these APIs securely into the web ecosystem, it is important to acknowledge that, like any other software, these APIs may contain vulnerabilities that malicious actors could exploit. The implications of these vulnerabilities depend on the very features of browser APIs but also on factors like implementation errors, unintended use, and unexpected composition [2]. Even though state-of-the-art browsers readily ship various mitigations to potential attacks, in some cases these might still fall short. For instance, we are aware of a few studies that exposed vulnerabilities within specific browser APIs related to battery status [3], filesystem [4], and screen sharing [5]; Snyder et al. also explored selective restrictions on a per-site basis [2].

In this work, we underline and review general security and privacy concerns within browser APIs. We first provide our readers with an overview of typical browser APIs functionality, identifying three groups:

Device Interaction APIs, Data Management APIs, and User Interaction APIs. Next, we present a research agenda that sheds light on the intricate balance between functionality and security within the realm of modern web applications. We also further detail a few scenarios where existing defenses struggle to comprehensively address browser security problems. Our objective is to inform the community about the potential risks of these APIs and stimulate a discourse that could lead to a more secure and private web ecosystem.

Browser APIs

Browser APIs are essential components of web browsers. In essence, they are a set of programming interfaces and standards for accessing and integrating the functionalities offered by a web browser.

From an implementation standpoint, these APIs are exposed as objects within the global scope of a web page. Developers can utilize browser APIs directly in their code, without needing plugins or external software. When a web page is loaded, the browser's JavaScript engine initiates a runtime environment that, among other things, grants access to a variety of browser APIs. Code can easily interact with these APIs via Javascript methods and properties associated with each API.

Community members, working groups, and browser vendors continuously work on new APIs to accommodate emerging technological opportunities and processes. Once an *experimental API* proposal reaches a sufficient level of maturity, it undergoes the scrutiny of standardization organizations. Passing this step does not necessarily imply universal availability, as a vendor may choose to not implement an API or offer only a restricted version of it due to business strategy, security considerations, or privacy concerns.

Browser vendors take a comprehensive approach to safely introduce new APIs, focusing on enhancing functionality while maintaining security. Before their release, new APIs undergo rigorous security reviews and testing, including automated and manual evaluations to identify and mitigate potential vulnerabilities. Some vendors also have trial programs that allow developers to test new features on their websites and provide feedback before these APIs are fully integrated into the vendor's platform. Additionally, some APIs may initially be available only behind feature flags, limiting their use until they are proven stable and secure.

The methods and properties inherent in mainstream browser APIs have significantly enhanced the capabilities and functionalities of web applications from various perspectives. In this work, we categorize

browser APIs into three groups based on their core objectives and distinct capabilities (Figure 1):

- 1) *Device Interaction APIs*: These allow web applications to interact directly with the user's device.
- 2) *Data Management APIs*: These allow web applications to interact with various storage mechanisms for storing user and website data.
- 3) *User Interaction APIs*: These enhance user interactions in web applications.

Next, we provide examples and use cases for each API group and explore their features in more detail.

Device Interaction APIs. These APIs facilitate direct communication between web applications and the user's device hardware, providing a standardized interface for data exchange and control in either direction.

The use of device-specific features can enhance the overall user experience, making it more interactive and responsive to the user's environment. Interaction is typically event-driven, meaning the API responds to changes in device state or user actions. The features of Device Interaction APIs include:

Accessing Hardware Features: A notable example is given by the `Media Capture and Streams` API, which enables web applications to access device hardware such as cameras and microphones. This API mediates the interaction by requesting permissions from the user and then providing the web application with a stream of data from the hardware, enabling tasks like capturing photos, recording audio and video, and real-time communication. Modern web applications can also probe the battery status of the host system via the `Battery Status` API and adjust their resource usage accordingly. In the realm of experimental APIs, the `WebGPU` API can support the drawing of complex images and also general-purpose GPU computations by harnessing GPU resources and APIs within the browser.

Sensing Device State and Environment: Device Interaction APIs can be utilized to sense the state of the device and its environment. For example, the `DeviceOrientation` and `DeviceMotion` APIs, by accessing data from internal sensors such as gyroscopes and accelerometers, allow applications to detect and respond to changes in device orientation and motion. The information can be used for various purposes, such as rotating the user interface to match the device orientation or creating motion-driven games.

Geolocation and Mapping: Device Interaction APIs can also facilitate location-based services. For instance, the `Geolocation` API leverages GPS, nearby cellular networks, Wi-Fi nodes, or other sensor



FIGURE 1. Types of modern browser APIs.

inputs to provide accurate location data. Web applications can then utilize this data for services like maps, local search results, or location-aware gaming.

Data Management APIs. Modern web browsers include different options for storing user and website data. The stored data can serve various purposes, such as saving website content for offline use, caching, or retrieving the user's information. To interact with data storage mechanisms, web applications utilize Data Management APIs. Key features of these APIs include:

Local and Session Storage Interaction: Data Management APIs offer key-value storage mechanisms that are fundamental to modern web application functionality. Specifically, the `Web Storage` API provides local and session storage: the former for persistent data to keep across multiple sessions, the latter for transient data needed only for the duration of a session.

Complex Data Resource Interaction: The `IndexedDB` API offers a robust solution for more advanced data storage needs. This API allows for the storage of significant amounts of structured data, including files and blobs, and supports high-performance searches using indexes. It is ideal for applications that need to handle large datasets, such as multimedia content, email clients, or complex web-based games.

Management of Cached Data: Modern web applications can also resort to cached data. The `Cache` interface available within the `Service Worker` API lets them manage a cache of network responses: this is a crucial capability, for example, to work offline or load faster by caching essential resources and assets.

Interacting with the Local File System: Web applications may also interact with the local file system of the user device. Through the `File System Access` API, they can modify a user-selected subset of files and directories in the file system much like native applications. This is essential for applications that deal with document editing, image and video processing, or any other form of user-generated content.

User Interaction APIs. These APIs can significantly

enhance modern web applications by facilitating more engaging and tailored user experiences. They encompass a variety of capabilities, from modifying the display environment to actively engaging users with timely notifications and understanding user activity. Key features of these APIs include:

Maximizing Display in a Browser: User Interaction APIs can change the display size in a browser. For example, the `Fullscreen` API enables web applications to expand their content to fill the entire screen, eliminating the browser interface elements for a more immersive viewing experience. This is particularly impactful in applications like media players or gaming platforms, where a larger viewing area provides a more immersive experience.

Direct User Engagement: By utilizing User Interaction APIs, websites can engage with users. For example, with the `Web Notifications` API, applications can send notifications through the native system of the device. With this feature, users can be engaged through real-time alerts and messages, even when they are not actively on the web page.

Activity Insights: In addition to interactive capabilities, User Interaction APIs can also be used to track user activity. For example, the `Idle Detection` API enables a web application to detect when the user becomes idle based on various indicators, such as keyboard or mouse activity and screensaver activation. This enables the application, for instance, to detect which of multiple devices the user is actively using and deliver messages to that device.

Security & Privacy Concerns

Browser APIs have become integral to modern web development, powering a vast array of interactive and dynamic features that create seamless user experiences. As browser vendors constantly innovate and introduce new capabilities, comprehensive security reviews and testing processes are essential to mitigate potential risks. Despite these efforts, the introduction of new

APIs can inadvertently open avenues for novel attacks or raise unexpected privacy concerns, highlighting the ongoing challenge of balancing functionality and security in the ever-evolving web landscape. In the following, we elaborate on prominent threats that we foresee.

Security Concerns

Abusing Device Resources: Despite the security provisions available in browsers, the interaction of web applications with device hardware may still introduce a variety of threats. For example, an adversary could create a web application that continuously activates the vibration feature (via the `Vibration` API), leading to significant CPU strain and rapid battery depletion, especially in mobile devices. Researchers have also demonstrated [6] how a threat actor may abuse resources for persistent and stealthy computation (e.g., cryptojacking), creating botnets able to perform unwanted computation or harmful operations even after the user closes the tab of the malicious website.

Enhanced XSS Attacks: Traditional Cross-Site Scripting (XSS) attacks typically involve injecting malicious scripts into static web pages, which then execute in the context of the browser. The introduction of modern browser APIs has inadvertently broadened the attack surface for XSS attacks. As these APIs enable more complex storage mechanisms and interactions with the DOM, attackers may find new vectors to inject malicious scripts into web applications. For example, using the `IndexedDB` API, an attacker could insert a malicious script into the database that a web application uses for storing user data [7]. This could lead to unauthorized actions such as stealing user session tokens or manipulating stored data. A pertinent study [8] examining XSS threats from insecure use of client-side storage found that more than 8% of the Alexa Top 5,000 domains have unfiltered data flows from persistent storage to dangerous sinks, highlighting a significant reliance within browser APIs on the assumed integrity of storage content.

New Malware Strains: The capabilities behind modern browser APIs may be abused for enhanced malware designs. For example, their powerful features can help adversaries create browser-based ransomware that operates directly from browsers. Researchers demonstrated [4] the attack using the `File System Access` API. Specifically, an adversary can create a web application that tricks users into granting access to sensitive areas of their local file system, bypassing the security model of the API. Once this access is obtained, the malicious web page encrypts the user's files directly through the browser. Unlike traditional

malware, the attack does not require a covert infection vector to stay undetected and runs entirely within the browser.

Enhanced Phishing Attacks: As some browser APIs are capable of modifying the user's screen, attackers may abuse them to create deceptive interfaces that mimic familiar and trusted user interfaces. These interfaces can then be used for phishing attacks. For example, by utilizing the `Fullscreen` API, an attacker could create a webpage that closely resembles the login page of a popular application (<https://textslashplain.com/2023/09/12/attack-techniques-fullscreen-abuse/>). Once the user navigates to it, the attacker can trigger the API to display the page in full-screen mode, where interface elements that help users identify whether the website is legitimate, such as the address bar, are not visible. The resemblance to the legitimate website may also likely overshadow, in the user's perception, the visual indicators from the browser about the webpage entering this mode. Along similar lines, the `Screen Capture` API offers another vector for phishing attacks. For example, an attacker, after compromising a user's account, could trick their friends into joining a screen-sharing session and visiting a malicious website. Unsuspecting users might not realize the malicious website can access cross-origin data, potentially capturing sensitive information from logged-in accounts (e.g., banking or social media) by opening pages through hidden iframes or new windows [5]. This may go unnoticed due to the limitations of human vision, such as with briefly displayed flashing content or content that is almost transparent. In a similar fashion, the attacker may steal cross-site request forgery tokens, which could enable further attacks to the user.

Privacy Concerns

Enhanced Fingerprinting Risks: Traditional fingerprinting techniques, such as collecting stateful data like cookies, offer limited insights and can often be controlled by the user. In contrast, web applications leveraging modern browser APIs can access a much broader spectrum of data about a user's device and browser environment. This can favor the development of sophisticated browser fingerprinting methods and an extended range of fingerprinting surfaces. Researchers have shown [3] that seemingly benign data retrieved via the `Battery Status` API, such as the frequency of battery status changes, can serve as a fingerprinting vector. The abundance of browser APIs can inadvertently facilitate such attacks: a recent study has revealed that 231 APIs (constituting 3.1% of all

Chromium APIs) are actively utilized for browser fingerprinting across the Alexa Top 10,000 domains [9]. A subsequent repetition of the experiments 11 months later uncovered an additional 18 APIs being exploited for fingerprinting purposes. On the bright side, fingerprinting methodologies from researchers can also be used to strengthen the efforts that browser vendors already put in privacy protection, as with standardized API outputs and stringent permission requirements.

Location Tracking Concerns: As discussed, certain browser APIs can access user location information. Although these APIs are designed to enhance user experience with location-based services, they can be misused for unauthorized location tracking. A main issue lies in their too coarse-grained permission and location models: for example, web applications can access either precise location information or none, even in situations where only a rough estimate would suffice. A study on 1,196 websites that use the `Geolocation` API revealed that about half of them required excessive privileges for their functionality [10].

Research Agenda

We believe the community is faced with a clear and pressing need for ongoing research to develop more efficient, automated, and developer-friendly security solutions that can keep pace with the rapid development of browser APIs and their potential exploitation in the wild. This may include a mix of new defenses and adapting existing mechanisms for their current ineffectiveness against threats involving these APIs, ensuring that security measures evolve in tandem with the capabilities and complexities of web applications. In the following, we share our reflections and present a research agenda to motivate future research on security and privacy issues of modern browser APIs.

Designing Defense Solutions

Browser APIs have not only revolutionized the capabilities of web applications, but also transformed the landscape of cybersecurity threats. Despite the efforts made by the browser vendors to secure them, these APIs have given rise to novel attacks and added complex dimensions to existing attack vectors, challenging the effectiveness of traditional defense solutions. Many consider sandboxing mechanisms the first line of defense within browsers, as they enhance the overall safety of web applications and thwart traditional browser attacks by isolating execution environments. However, sandboxing an execution instance would not prevent a web application from abusing trusted

browser APIs, as for example with the ransomware [4] and the botnet [6] attacks mentioned earlier in the article. Based on the knowledge gained from our study, in the following, we focus on two relevant research directions that elude traditional security measures available within browsers and the operating system. By describing possible challenges we foresee in defense design, we hope to motivate the research community to develop robust defense solutions against the security and privacy issues rooted in modern browser APIs.

Attacks through ‘Benign’ Browser Processes.

Some attack vectors traditionally exclusive to native applications have found a new breeding ground within web applications, as with ransomware attacks [4] and botnets [6] deployed through modern browser APIs.

Current defense mechanisms may fall short in detecting threats operating within a browser. In the case of malware, antivirus products aim to capture and block suspicious activities involving file system access and network communications, particularly when they originate from untrusted executable files or abused (e.g., injected) processes. However, in this new scenario, execution is subtly interspersed with the benign activity of the browser, posing a harder detection challenge.

This increased difficulty affects general security methodologies, such as monitoring system calls and other events from process activity, which have been instrumental in detecting various types of issues in the past. Constant monitoring of browser-initiated activity, especially if bolstered to also recognize interspersed threads of activities, may lead to significant runtime overhead and degrade user experience.

Future research should look at developing specialized detection techniques to handle behavioral patterns unique to browser environments. Cross-fertilization opportunities may exist with defensive research on multi-process malware, for which activity-based detection faces analogous challenges but data-flow correlation via specific objects seems possible [11].

Research Direction 1: Future research should concentrate on detection techniques capable of distinguishing between benign and malicious activities within browsers, taking into account the unique execution patterns of emerging web technologies.

Revamping Known Attacks. The integration of browser APIs into web applications has inadvertently provided new avenues for revitalizing traditional security threats. In the Security Concerns discussion, we anticipated how browser APIs may enable XSS attacks via their storage mechanisms. Traditional defenses are

capable of significantly hindering conventional XSS attacks: for example, output encoding effectively neutralizes scripts injected in the HTML DOM by sanitizing special characters in user input before rendering them on the web page. However, they do not extend to local storage from the IndexedDB or Web Storage APIs. These storage mechanisms can be used to store and retrieve complex data structures, including executable scripts, without any sanitization.

Trusted Types have thus been proposed as a broader mitigation, locking down risky injection sinks by ensuring that data can reach them only after a developer-supplied sanitizer function processes it. This design limits by construction potential threats involving risky browser APIs, offloading to developers the trustworthiness check on data. A recent study conducted with web developers reveals that implementing Trusted Types, at least in the proposed form, can be time-consuming and demand significant engineering effort to properly write a secure filtering function [12].

Research Direction 2: Browser APIs may enable traditional attack vectors in new contexts, hitting blind spots in current defenses. Future work should evaluate and adapt them to these new threats.

Analyzing Web Application Behavior

Given the complexity and dynamic nature of browsers, another critical area requiring concerted effort is the development of solutions for fine-grained analysis of web application behavior. These strategies may focus on understanding and responding to the nuanced behaviors from the heterogeneous uses of browser APIs.

Fine-grained analysis requires a detailed understanding of how browser APIs interact with web applications and the underlying browser environment. Static analysis tools and techniques might be useful for a broad review of API usage patterns but also have limitations, particularly when dealing with obfuscated code or dynamic runtime behaviors. Analyzing the web application as it executes may be inevitable, and will therefore be the focus of the remainder of this section. Dynamic analysis can be performed on web applications by utilizing different techniques, two of which we will explain here.

Instrumentation and API Hooking. These techniques involve modifying the browser environment to monitor browser API calls by using browser debugging protocols, such as Chrome's DevTools Protocol. These protocols open a gateway to the browser's internal operations and enable deep access to its functionalities. Moreover, by hooking into specific browser API calls,

every invocation can be captured and logged, enabling the recording of the parameters being passed and the responses from these APIs. As an example, Listing 1 shows how the `getCurrentPosition` method from the `Geolocation` API can be intercepted to log and possibly shepherd its usage.

```
Runtime.evaluate({
  expression: `
    if (navigator.geolocation) {
      const originalGetCurrentPosition =
        navigator.geolocation.
        getCurrentPosition;
      navigator.geolocation.
        getCurrentPosition = function() {
          console.log('Location Accessed');
          return originalGetCurrentPosition
            .apply(this, arguments);
        };
    }
    awaitPromise: true
  });
```

Listing 1. A sample instrumentation snippet.

Techniques of this kind assist defense mechanisms in identifying unauthorized or unusual API usage and getting a complete picture of the behavior of a web application. Moreover, they can contribute to enhance broader security analysis frameworks. An example of their usefulness is given in [13], where the authors identify points in obfuscated JavaScript code where privacy-sensitive APIs are used and provide resource replacements compatible with content blocking tools.

Similar interposition mechanisms have historically been instrumental in tackling many traditional security problems. However, implementing reliable and accurate interposition mechanisms poses well-known challenges when it comes to compatibility, coverage, performance, and transparency properties of the instrumentation, especially for security uses. Looking at what related software and systems security research fields experienced (for example, with continuously evolving static and dynamic binary instrumentation techniques), we advocate for further research in easing the capabilities, usability, and performance of instrumentation mechanisms for web browsers. The downstream users of these improved mechanisms can be innumerable: for example, new mechanisms that like [13] provide plug-and-play security and privacy mitigations for users, or tools for execution dissection and understanding tasks (e.g., as in [4] by pausing execution upon specific events to examine the ongoing effects of an attack).

Tracing. Tracing is a pivotal technique for analyzing web application behavior as it offers a detailed view

of the operational dynamics between web applications and the browser's rendering engine, along with their interactions with browser APIs. This method allows developers to capture a comprehensive snapshot of a web application's runtime behavior to shed light on the intricate operations of browser APIs, JavaScript execution, and other crucial processes.

Figure 2 depicts an exemplary trace collection process. By employing custom scripts or leveraging the browser's built-in tracing tools, developers can trace web applications with a fine-grained granularity. These traces, once collected, can undergo different meticulous analyses such as timeline analysis, which involves scrutinizing the sequence of events, JavaScript execution patterns, and memory usage to gain a thorough understanding of web application behavior.

Currently, the state-of-the-art tracing system is VisibleV8 [14], which logs interactions involving JavaScript calls to native functions provided by the browser, such as accessing properties on browser-backed objects like `window` and `document`. This capability allows it to effectively monitor many traditional web interactions, and by now several research works have built on it.

Unfortunately, VisibleV8 fails to observe behaviors when browser APIs orchestrate operations that do not directly call native functions or access native properties. Some APIs may handle data and execute operations internally within the JavaScript engine or via optimized pathways that do not hit the probes of the tracing system. Also, core parts of some operations (for example, a network request with the `Fetch` API) may be handled by browser layers that do not necessarily expose native interactions to the JavaScript engine, which is what tracing tools monitor. This highlights a critical gap in the monitoring capabilities of state-of-the-art tools in the face of subtle and complex interactions within modern web applications. Additionally, as current tracing systems are often tied to a particular browser implementation, manual work is currently needed to use them with newer browser versions.

Research Direction 3: Future research should focus on developing tools to comprehensively capture events related to browser API usage, pursuing completeness of the interposition, fine-grained granularity of the collected information, and compatibility across browser products and releases.

User Studies

While technical users who design and analyze browser APIs might be aware of the security and privacy issues that come with them, a significant portion of the general

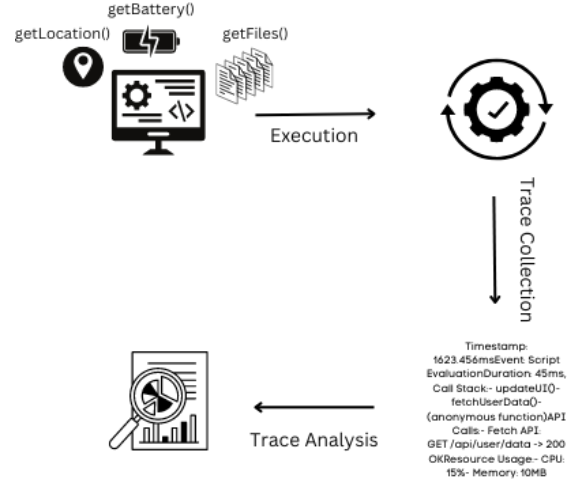


FIGURE 2. An illustration of a trace collection process for analyzing web application behavior.

population most likely does not. Non-technical users, often the most susceptible to the risks associated with new technologies, ideally should not be burdened with understanding the technical intricacies of mitigating the security and privacy concerns posed by browser APIs.

Browser vendors have recently investigated user's behavior on permission dialogs using telemetry and large-scale user studies [15] to optimize the user interface and the activation logic, so as to minimize interruptions to the browsing experience without undermining legitimate website functionality enabled by browser APIs that the user needs to grant permission to. Orthogonal research efforts [2] have then explored ways to enforce the least privilege principle on permissions without impacting website functionality.

We believe all these efforts tackle a dimension of remarkable importance. However, we note a shortage of studies on how non-technical users perceive and interact with requests generated from different browser APIs and applications using them. In particular, an understudied aspect in users' trust is the limits within which users feel safe when different sources of sensitive information are involved in requests. These limits contribute to determine which requests users perceive as legitimate and which they see as suspicious or potentially harmful. One way to implement such a study would be to simulate attack scenarios in a controlled environment to assess how users detect and react to security or privacy threats and obtain insights into their innate ability to identify suspicious or potentially harmful behavior. The resulting insights may be beneficial for improving the way permission prompts are presented to users to favor effective decisions.

On the other hand, we previously presented security and privacy concerns that a user would be unable to react to, such as with cross-site scripting attacks. Another dimension worth analyzing would thus be the visibility of risk in the first place: studies of this kind would unveil where gaps are and how users would perceive potential solutions for their mitigation.

By conducting these varied and targeted studies, the community may understand user behavior better and use this knowledge to develop more user-friendly security measures, ensuring a safer and more secure browsing experience without overwhelming the user with additional permission dialogs and controls.

Research Direction 4: Non-technical users incur additional risk exposure from the plethora of scenarios enabled by modern browser APIs. Future research should investigate risk visibility and trust in users to obtain insights for the design of new impactful, user-friendly security measures.

Platform Heterogeneity

Currently, more than half of the online population use traditional desktops and laptops to browse the Internet, while the rest utilize alternative platforms such as mobile devices, tablets, or smart home devices (<https://explodingtopics.com/blog/mobile-internet-traffic>). The diversity of devices, each with its own set of hardware capabilities and user interaction patterns, creates a complex landscape for browser API implementation, requiring careful consideration to ensure functionality and security across platforms.

For instance, resource exhaustion attacks via browser APIs on resource-constrained systems such as mobile devices can be particularly impactful. These attacks may target the limited processing power, memory, and battery life inherent to mobile technology. Additional countermeasures might be needed depending on the specificity of the threats, such as more stringent controls over browser APIs that access hardware features or improved management of background data access from web applications to mitigate unauthorized tracking and data leakage risks. As a closely related point, the integration of browser APIs into browsers for mobile devices and other platforms can introduce unique privacy concerns. For example, mobile devices naturally have access to more sensitive data (such as location, contact lists, and personal media) and to more types of sensors compared to desktop systems.

From a technical standpoint, all the factors mentioned above make us call for research in developing analysis environments that can assess the behaviors

of web applications on heterogeneous device types.

Research Direction 5: Future research should look at the security and privacy issues of modern browser APIs across different platforms and tackle the further challenges that non-desktop platforms bring by developing appropriate techniques and tools.

Conclusion

The introduction of modern browser APIs has revolutionized web content development and enabled the creation of more advanced and interactive web applications for all Internet users. Modern web browsers employ a variety of defense mechanisms to counter potential attacks; nevertheless, there are still instances where these measures fall short. In this work, after providing a general categorization of browser APIs in three groups, we identify and examine security and privacy issues associated with these APIs. We then discuss issues learned from our study, enumerating five research directions to motivate work capable of addressing the intricate balance between functionality and security when dealing with modern browser APIs. We hope the insights derived from this work will aid the community in identifying and understanding the multifaceted security and privacy concerns within these APIs.


Acknowledgments

We thank our anonymous reviewers for their precious feedback. This work was partially supported by the US National Science Foundation (Awards: 2039606, 2219920), Florida International University Graduate School, Cyber Florida, the Google ASPIRE Program, and the Italian MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU through projects SERICS (PE00000014) and Rome Technopole (ECS00000024). The views expressed are those of the authors only, not of the funding agencies.

REFERENCES

1. P. Snyder, L. Ansari, C. Taylor, and C. Kanich, "Browser feature usage on the modern web," in *ACM Internet Measurement Conference*, 2016, pp. 97–110.
2. P. Snyder, C. Taylor, and C. Kanich, "Most websites don't need to vibrate: A cost-benefit approach to

- improving browser security,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 179–194.
3. Ł. Olejnik, G. Acar, C. Castelluccia, and C. Diaz, “The leaking battery: A privacy analysis of the HTML5 battery status API,” in *Data Privacy Management, and Security Assurance*, 2016, pp. 254–263.
 4. H. Oz, A. Aris, A. Acar, G. S. Tuncay, L. Babun, and S. Uluagac, “RøB: Ransomware over modern web browsers,” in *USENIX Security Symposium*, 2023, pp. 7073–7090.
 5. Y. Tian, Y. C. Liu, A. Bhosale, L. S. Huang, P. Tague, and C. Jackson, “All your screens are belong to us: Attacks exploiting the HTML5 screen sharing API,” in *IEEE Symposium on Security and Privacy*, 2014, pp. 34–48.
 6. P. Papadopoulos, P. Ilia, M. Polychronakis, E. Markatos, S. Ioannidis, and G. Vasiliadis, “Master of web puppets: Abusing web browsers for persistent and stealthy computation,” in *Network and Distributed System Security Symposium*, 2019.
 7. S. Kimak and J. Ellman, “The role of HTML5 IndexedDB, the past, present and future,” in *International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, 2015, pp. 379–383.
 8. M. Steffens, C. Rossow, M. Johns, and B. Stock, “Don’t trust the locals: Investigating the prevalence of persistent client-side cross-site scripting in the wild,” in *Network and Distributed System Security Symposium*, 2019.
 9. J. Su and A. Kapravelos, “Automatic discovery of emerging browser fingerprinting techniques,” in *ACM Web Conference*, 2023, pp. 2178–2188.
 10. H. Kim, S. Lee, and J. Kim, “Exploring and mitigating privacy threats of HTML5 Geolocation API,” in *Annual Computer Security Applications Conference*, 2014, pp. 306–315.
 11. D. C. D’Elia and L. Invidia, “Rope: Bypassing behavioral detection of malware with distributed ROP-driven execution,” in *Black Hat USA 2021*. Las Vegas, NV, USA: Black Hat, 2021.
 12. S. Roth, L. Gröber, P. Baus, K. Krombholz, and B. Stock, “Trust me if you can – how usable is trusted types in practice?” in *USENIX Security Symposium*, 2024, pp. 6003–6020.
 13. M. Smith, P. Snyder, B. Livshits, and D. Stefan, “SugarCoat: Programmatically generating privacy-preserving, web-compatible resource replacements for content blocking,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2844–2857.
 14. J. Jueckstock and A. Kapravelos, “VisibleV8: In-browser monitoring of JavaScript in the wild,” in *ACM Internet Measurement Conference*, 2019, pp. 393–405.
 15. M. Harbach, I. Bilogrevic, E. Bacis, S. Chen, R. Uppal, A. Paicu, E. Klim, M. Watkins, and B. Engedy, “Don’t interrupt me - a large-scale study of on-device permission prompt quieting in Chrome,” in *Network and Distributed System Security Symposium*, 2024.
- Harun Oz** is a Ph.D. candidate at Florida International University, Miami, USA, where he is currently a graduate research assistant in the Cyber-Physical Systems Security Lab. His research interests include application of advanced machine learning techniques to explore the security and privacy implications of emerging technologies and their associated threats. He received a M.Sc. in computer science from the Florida International University. Contact him at hoz001@fiu.edu.
- Daniele Cono D’Elia** is a tenure-track assistant professor at Sapienza University of Rome, Italy. His research spans several fields of software and systems security, with a main focus on how program analysis techniques can boost accuracy and performance aspects of security policies. D’Elia received his Ph.D. in engineering in computer science from Sapienza University of Rome, Italy. Contact him at delia@diag.uniroma1.it.
- Güliz Seray Tuncay** is a senior research scientist in the Android Security and Privacy group at Google. Tuncay received her Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. Her research interests include mobile and IoT security, usable security, web security, and mobile computing. Contact her at gulizseray@google.com.
- Abbas Acar** is a postdoctoral associate in the Cyber-Physical Systems Security Lab at Florida International University, Miami, USA. His research interests include privacy-aware technologies, alternative authentication methods, and security/privacy issues related to the Internet of Things. Acar received his Ph.D. in electrical and computer engineering from Florida International University. Contact him at aacar001@fiu.edu.
- Riccardo Lazzeretti** is an associate professor in engineering in computer science at Sapienza University of Rome, Italy. His research focuses on security and privacy, with a particular focus on the Internet of Things. Lazzeretti received his Ph.D. in information engineering from the University of Siena, Italy. He is a Senior Member of IEEE. Contact him at lazzeretti@diag.uniroma1.it.



Selcuk Uluagac is a professor in the School of Computing & Information Sciences at Florida International University, Miami, USA, where he leads the Cyber-Physical Systems Security Lab. His research focuses on cybersecurity and privacy with practical and applied aspects. He received his Ph.D. from Georgia Institute of Technology and M.Sc. from Carnegie Mellon University. Contact him at <https://users.cs.fiu.edu/~suluagac/>.