Web attacks on IOS and MacOS and Mitigation Techniques.

Author:

Uma Naga Lakshmi Musunuru

School of Computer and Mathematical Science

The University of Adelaide

GitHub Link:

Supervisor:

Olaf Maennel, Kaie Maennel

School of Computer and Mathematical Science

The University of Adelaide.

Table of Contents

Abstract-

The aim of this research is to explore web-based attack vectors specifically targeting the iOS and macOS platforms, with the focus on attacks against WebKit based web browsers. Although Apple platforms are perceived as very secure, their rendering engine WebKit is a monoculture vulnerability landscape that requires specialized analysis since it is mandatory. In this work, we conduct comprehensive study of 247 WebKit related CVEs (2020–2024) through a systematic analysis of their impact and a novel WebKit Security Analyzer tool we built, identify platform specific vulnerability patterns, and assess effectiveness of mitigation techniques across various scopes of an application.

We present key findings covering the top 12 WebKit vulnerability categories and find that (1) the JavaScript engine (42%) and DOM API implementation (27%) issues are the top two vulnerability categories, (2) the exploitation characteristics are significantly different between the iOS and macOS environments, and (3) many other types of vulnerabilities still require concentrated attention. The results of the research indicate that the inherent platform specific controls that are available and achievable reduce mean risk on 73% in five case studies and thus outperform generic web security significantly. 93% of identified vulnerabilities were prevented via a layered defense applying Content Security Policy, together with WebView configuration hardening.

We make four contributions in this work: (1) an identification and classification of WebKit specific security vulnerabilities; (2) an open-source security analysis toolkit based on static analysis to make it easier for security analysts to look for WebKit vulnerability patterns; (3) an empirical measurement of the effectiveness of mitigation techniques; and (4) practical implementation guidance for security analysts within the WebKit ecosystem at Apple. Second, we show that security analysis specific to the platform is important, as well as how these targeted. mitigation strategies can improve the security of WebKit by a significant amount.

Introduction

With the ever-growing sphere of Apple's ecosystem in the computing landscape, the cybersecurity has experienced a paradigm shift. Since 2025, when the combined active devices powered by iOS and macOS worldwide amount to over 1.8 billion, these platforms have started to serve as prime targets for sophisticated threat actors. In recent years, as web-based vectors have been introduced through which the attack surface has expanded beyond the historical perception of Apple systems as being inherently secure, perception of that security has been challenged. This dissertation examines the emerging web-based attack landscape focused on compromising iOS and macOS environments by expounding on the technical attack affecting as well as the countermeasures to them.

Both iOS and macOS have security model that have substantially evolved from their Unix basement roots. Apple's macOS that is based on BSD Unix carries with it the security mechanisms such as Gatekeeper, SIP, and the relatively new introduced signed system volume (SSV) in macOS Big Sur and onward. iOS on the other hand has the more restrictive security model where kernel level enforced application sandboxing is mandated through mandatory access controls. The default rendering engine for all browsers with each operating system is implemented by WebKit, as they purport to create a single security boundary in which web content processing should take place.

These platforms can be attacked in various ways by web attacks such as browser-based exploits on WebKit vulnerabilities, malicious web content which exploit application programming interfaces (APIs), cross site scripting (XSS) and cross site request forgery (CSRF) using authenticated sessions, and increasingly sophisticated phishing techniques that bypass user vigilance. All these attack vectors share the trait that they can all do what they need to do without violating the security model of Apple devices, i.e., without unauthorized access to sensitive data or system functionality.

It profiles with the advantage of WebKit being the mandated rendering engine for all but one of the browsers in Apple's ecosystem (with the exception of an SVG rasterizer incorporated into that browser) and there is both an advantage and a disadvantage in this regard. The advantages of this approach of updating the security throughout all WebKit based browser applications at once are that the same exploit can then be spread quickly across all browsing environments, but the one downside is that this monoculture vulnerability i.e., a single

WebKit exploit that can affect each and every one of them. Common Vulnerabilities and Exposures (CVE) records reveal historical analysis that WebKit vulnerabilities would consistently be one of the leading attack surfaces for iOS and macOS systems and that stability of just in time (JIT) compilation flaws as an arbitrary code execution vulnerability.

As the primary client-side application programming language, JavaScript is both a necessary functional component and a large surface of attack. SOP and next to that CSP provide the fundamental security boundaries in the browsers, the JavaScript security model is built upon. Overall, however, modern web applications have been so complex that these boundaries have been compromised in numerous situations. Among those most notable are prototype pollution vulnerabilities, DOM strings based XSS attacks and JSON hijacking techniques which have been found to work against iOS and macOS browser bridging respectively.

The increase in the number of attack possibilities due to the interplay between web technologies and native application capabilities in WebView implementations of bridging web content to native functionality has been enabled. WKWebView and the legacy UIWebView components used to be the ways by which any sort of web content has been integrated in native applications in iOS. Like macOS applications, many web content inside of WebView components is embedded in macOS applications, and some of these risks may be inherited from the web. This dissertation discusses how thieves have betrayed these integration points to boost privileges and secure delicate information past browser sandboxes.

The supply chain attacks serve as an increasing threat to compromise both iOS and macOS systems via web-based interfaces. In recent times, compromised development dependencies, advertising networks and content delivery networks (CDNs) have been shown to be distribution vectors for malicious code that ultimately infects Apple devices. Since third party JavaScript libraries have become so prevalent in modern web development, a compromised dependency also becomes a threat to the most amount of sites, with the attack surface not limited to a few, but potentially thousands of sites at once, presenting a significant risk to Apple device users.

Targeted attack campaigns specifically aimed at robbing high-value iOS and macOS users are emerging in the threat landscape that Apple's platforms face. Since the introduction of advanced persistent threats (APTs) in recent years, these exploits are known to feature very sophisticated technique chains ranging from web-based attack vehicle to exploiting them into gaining persistent system access. Often these attacks comprise zero days with social engineering otherwise technical security controls will be unable to protect these systems.

This has meant other security considerations beyond the other applications that use cryptoriy and financial technology applications on Apple devices. Web based attacks against cryptocurrency wallet and exchange access credentials that target users of the high value cryptocurrency assets are quite high. Internet attacks that are targeted at users of financial technology applications in iOS and macOS have been seen by sophisticated phishing techniques and malicious JavaScript payloads that aim at compromising sensitive financial operations.

In web-based interactions involving Apple platforms, privacy and security are closely related. Since privacy seems to be the key differentiator of the company, the company is also putting privacy as a big lie in the line of product features, including ITP, App Tracking Transparency, and Private Relay. This effectively erased the web and forced those who want to track, track you, while also changing the playing field so much to aid in legitimate web functionality while eviscerating the end user at the same time as malicious actors have continually tried to adapt their tracking and fingerprinting techniques to track you anyway! In this dissertation, I analyze the relations between privacy protections and security controls in some sometimes-counterintuitive ways that can affect the way attackers approach their work.

Prior to the introduction of Apple Silicon processors with the M1 chip introduced in 2020 and subsequent generations, there were certain security assumptions around the macOS system that are no longer true. Apple Silicon's arm64 architecture contains both future security features and new attack surfaces compared to Intel-based systems. When trying to execute shellcode or go through return-oriented programming on these systems, web attacks must also take into consideration architecture specific factors. This is also a big architectural pivot in the macOS security, so worthy of special study.

Apple's security architecture and response mechanisms are increasingly shaped by international regulatory frameworks, and web security is among them of notable importance. For example, Digital Markets Act (DMA) and Digital Services Act (DSA) in the European Union have

direct implications on Apple's browser engine policy and app distribution models that affects web security posture of devices under its scope. Similarly, how web applications can safely work across international boundaries through compliant control over how its data is handled and encrypted, are the same for international boundaries.

A web-based threat to an Apple system brings in a set of enterprise deployment considerations which further increases the complexity of their security analysis. Mobile device management (MDM) solutions and enterprise certificate distribution mechanisms determine how and how organizations can counteract web-based attacks. Nevertheless, those same mechanisms have historically harboured their own vulnerabilities when unforgivably implemented. This dissertation considers consumer and enterprise security models from the point of web-based attack vectors.

Web based attacks can be defended against by educating and becoming aware of the users. The success or failure of security indicators, warning messages, and permission dialogs in influencing user behaviour is an important factor in verifying the effectiveness of these defences against many web-based attacks. This dissertation studies the effect of user interface design choices on security outcomes under potentially malicious web content in a user’s IOS or MacOS browser.

While web standards evolve, new attack surface can be introduced that provide such new capabilities while enabling new security protections. Web USB, Web Bluetooth, and Cross-Origin Resource Policy (CORP), Cross-Origin Opener Policy (COOP) technologies allow to provide capabilities that were available only to native applications, but also to prepare new security guidelines. In this dissertation, I study how Apple implements these newly designed standards, how it affects the security position of their platforms.

PWAs on Apple platforms are important as they lie somewhere between traditional websites and native applications regarding the security implications. Nowadays, versions of iOS and macOS support much more sophisticated installation experiences and features than what Apple offers through Web Apps. This dissertation investigates the security model for PWAs and the associated attack vectors and protection mechanisms from the perspective of the deployment model.

This dissertation concludes with a look at future ways of web security on the Apple platforms by observing emerging technologies and threat models that will contribute to shaping where the security landscape will be in the years to come. All these include augmented reality capability for web browsers, the growing role of machine learning in security decisions, and the advent of quantum computing against the cryptography infrastructure, giving rise to the expectations about the future security landscape for web interactions on Apple devices.

The purpose of this paper is to detail a comprehensive analysis of web-based attacks against iOS and macOS systems as a means of understanding existing vulnerabilities and upcoming security threats. This dissertation studies how the technical mechanisms behind attacks are exercised through their current effectiveness in the security context of Apple's computing platforms within the hostile web environment.

Aims and Challenges

Primary goal is to:

1. Systematically identify, analyze, and document web-based attack vectors specifically targeting iOS and macOS systems.
2. Develop effective, practical mitigation strategies that can be implemented by security professionals and developers.
3. Focus on the unique security architecture of Apple platforms.
4. Provide actionable security guidance addressing distinctive vulnerabilities of WebKit-based browsers.

This more clearly articulates the practical, solution-oriented focus of your research while highlighting its specific relevance to Apple's ecosystem.

The main goal of this dissertation is to comprehensively analyze the attack vectors of web-based attacks that specifically aim to attack iOS and macOS systems and with the evaluation of those attacks mitigation strategies. The lack of information surrounding web security considerations for Apple-specific platforms does not reflect the reality of realities of Apple. The aim of this thesis is to bridge significant gaps in current understanding of Apple-specific security considerations to understand how these platforms differ from more studied Windows and Android platforms. This dissertation presents targeted insights on the intersection of web technologies with Apple's security architecture for security practitioners, software developers and academic researchers respectively, by focusing on a limited scope.

The focus of the central research objective is on taxonomizing and analysis of web attack methodologies that leverage the features of WebKit, Safari, and related browser technologies on Apple platforms. The techniques demonstrated in this research are specific to iOS and macOS systems, and do not generalize to the wider platform of web security, however general web security principles to a large degree apply. It includes learning what patterns to look for among successful exploitation chains and how Apple's architectural design choices affect attacker methodology.

This dissertation also seeks to assess how well both platform native security controls and third-party security solutions can defend web-based attacks. This research strives to provide evidence-based recommendations for security configuration to cater the highest level of resilience to maintain against the existing attack methodology through establishing empirical measurements of protection efficacy. For Apple platforms, it includes analysis of browser and security related settings, system level protections and supplementary security tools.

Due to the fast-growing attack trends as well as the proliferation of good attack techniques, this research domain is a significant challenge. Since Web standards are always adding new features which expand the attack surface, and apple's major operating system releases come two times a year which are altering their security architecture, so that is the scenario of app security. Temporal challenge must be addressed by this dissertation, attempting to identify fundamental principles beyond implementation details that nevertheless deliver concrete, actionable guidance to the current systems.

This is also another substantial challenge in Apple's security architecture which is closed, providing visibility into certain protection mechanisms and implementation details is limited. Unlike open platforms, iOS and macOS constrain access to internal security subsystems making it hard to analyze the entire attack and defense space. To solve this problem, carefully designed experiments, and triangulation of available information sources will be required to obtain these accurate security models in the presence of these constraints.

Another dimension associated with these problems is that features to protect your privacy in Apple's products often alter traditional web behaviours that make traditional web behaviour analysis tools and technologies ineffective, while also introducing new attack vectors. Also, this dissertation must distinguish carefully between privacy and security controls, given that both are related, but distinct concerns in practical scenarios, and this must be done precisely to understand how they each interact with the other.

An ethics and legal constraint around security research on commercial platforms is a methodological challenge. A responsible disclosure of testing attack vectors against Apple systems takes into consideration responsible disclosure practices, terms of service limitations, and legal frameworks for security research. To implement the appropriate research protocols for this dissertation, the following essential boundaries must be respected, and their security insights need to be valuable.

This research also attempts to make contributions towards the development of better security models geared toward the properties of Apple's computing platforms. This dissertation builds a bridge between empirical findings and theoretical security principles through synthesizing capabilities which will contribute to advancing the existing conceptual frameworks towards understanding and improvement of the web security in iOS and macOS environments. It also involves finding systemic vulnerabilities affecting CVEs or individual attack instances that signify under some aspect of architectural changes that would improve the security posture broadly.

Original contributions

This research makes the following original contributions:

1. Development of the "WebKit Security Analyzer" - A novel computational toolkit specifically designed to detect WebKit-specific vulnerabilities.
2. Creation of a specialized taxonomy for classifying WebKit vulnerabilities that accounts for platform-specific exploitation characteristics on iOS and macOS systems.
3. Original empirical measurements of mitigation effectiveness against WebKit-specific vulnerabilities across 157 websites and five comprehensive case studies.
4. Development and validation of an integrated methodology combining automated analysis with controlled experimental validation to identify platform-specific vulnerabilities.

Despite the numerous advances in research being made in the security of the web, this work bridges a large gap in current literature by focusing on WebKit vulnerability research. This study makes its original contribution in many aspects including the methodology, analytical framework, classification taxonomy and empirical measurements.

This research builds upon and extends several existing approaches to web security analysis:

1. OWASP ZAP and similar web vulnerability scanners provide general-purpose detection capabilities but lack WebKit-specific pattern recognition.
2. CSP Evaluator (Google) analyses Content Security Policy implementations but does not account for WebKit-specific bypass techniques.
3. Mobile-Security-Framework (MobSF) provides mobile application security testing but offers limited WebKit component analysis.
4. Crawljax and similar tools enable dynamic analysis of web applications but are not optimized for identifying WebKit-specific behaviour.

Though useful in general, these tools are not specialized in their focus down to the level needed to be able to analyze WebKit-specific vulnerabilities. In this research, the WebKit Security Analyzer addresses this gap by:

1. Implementing WebKit-specific pattern recognition for vulnerability detection
2. Accounting for the unique parsing behaviour of WebKit'ss HTML and JavaScript engines
3. Providing specialized analysis of WebView configurations in iOS and macOS applications
4. Offering platform-specific mitigation effectiveness measurements

This approach complements existing tools while offering specialized capabilities for Apple's ecosystem.

Literature Review

Evolution of Web Attack Vectors on Apple Platforms

In more recent years we have seen significant evolution in the security landscape for Apple's computing platforms; with the web being a common attack vector for both iOS and macOS environments. Because of this unique attack surface created by the distinctive architecture of these systems, and by following the Apple standards for browser technology, it's also unique in that you need specialized analysis to do it. In this literature review, we identify the state of art of research around web attacks targeting Apple platforms and countermeasures presented by academic research, industry reports as well as the security practitioners' views.

Cross-Site Scripting (XSS) Vulnerabilities in WebKit-Based Browsers

Cross site scripting is currently the major web security vulnerability that affects all platforms, including Apple's ecosystem. Stored, reflected and DOM based are the three categories of XSS attack vectors that are presented and founded by Gupta and Gupta [11]. Although work on tackling XSS in this more platform agnostic way, their work lays down the critical terminology and concepts that will be key to understanding the Apple specific work. Based on this foundation, Liu et al. [14] surveyed comprehensive XSS exploitation and detection methodologies, pointing out that "the WebKit's unique rendering pipeline makes unique execution contexts that can be exploited to perform XSS attacks that cannot be performed in other browser engines" [14].

As a result, the Webkit implementation possesses not only security advantages, but also specific vulnerabilities with respect to all iOS browsers. Rodríguez et al. [24] point out that this standardization means that security updates to all the browsers take place simultaneously, but in doing so also creates a monoculture vulnerability in which a single WebKit exploit can affect all browsing environments at a single time. According to their analysis, those 47 WebKit specific XSS vulnerabilities were identified between 2018 and 2020, 23% were rated as high severity. In particular, "just in time (JIT) compilation mechanisms in WebKit introduce unique attack surfaces that do not exist in the non-JIT browser contexts" [24].

Content Security Policy (CSP) implementation forms the primary means by which Apple mitigates XSS vulnerabilities, however the underlying approach to doing this has changed considerably. Kerschbaumer et al. [12] studies the effectiveness of CSP in preventing XSS on modern browsers and finds that Webkit's implementation provides effective protection for reflected and stored XSS, but inconsistent protection for DOM based XSS. "Their experimental evaluation showed that 'Safari's implementation of CSP directive parsing is different enough from the standard that an exploration of it can be exploited to go around protections when the payloads are crafted very carefully'" [12].

Several researchers have questioned the effectiveness of XSS filtering mechanisms dedicated to WebKit. Client side XSS defense techniques have been analyzed by Wang and Xu [29] and the conclusion is that Safari's approach to XSS mitigation is much more dependent on sanitization than are the filtering mechanisms in other browsers. "From this comparative analysis, it appears that 'WebKit's sanitization approach to XSS defense has 37% less detections of obfuscated attack vectors than filter-based approaches demonstrate'" [29]. This finding implies that Apple's security model to deal with such web content processing may need to be revisited from the bottom up to deal with sophisticated evasion techniques.

Recent advances in the ways of exploiting XSS often quite sophisticated bypass traditional defenses. Mondal et al. [18] used a reinforcement learning technique to automate XSS payload generation that can avoid various filters. Specifically, they evaluated the effectiveness against WebKit's sanitization mechanisms, and they found that 'machine learning generated payloads achieved a 52% success rate against Safari's XSS protections whereas manually crafted payloads were 28% successful' [18]. A concerning trend in this case implies that automated attack generation is progressing more quickly than the current capabilities of defense mechanisms on Apple platforms.

Emerging Detection and Prevention Approaches

Attack methodologies have gotten more sophisticated, so strategies aimed at detecting and preventing attacks on Apple platforms have become more sophisticated as well. As reported in [2], Alhamyani and Alshammari developed an approach to detect XSS using machine learning with focus on WebKit based browsers. They demonstrated that, with the right automated detection system, Safari XSS payload detection can be as accurate as 94.3 percent. Of note, they found that "vector patterns that target WebKit's

JavaScript engine have distinctive signatures for more accurate classification than generic XSS detection" [2].

Deep learning approaches to XSS detection have great promise for detecting XSS in iOS and macOS environments. They implemented a supervised deep learning framework to detect malicious web content using the specific training data which contain WebKit specific attack patterns [1]. Hitting exactly 97.8% of suspicious XSS attacks targeting Safari, with just a 2.3% false positive rate, their approach was very successful. As noted by the researchers, 'We also show that deep learning models trained on WebKit exploitation patterns sharply outperform all other web security models—are more than three times more accurate than generic web security models' [1]—and so there is said to be a case for platform-specific security techniques.

More specifically, it provides a promising avenue in the integration of semantic parsing techniques with XSS detection, based on Apple's security model. Following from Pardomuan et al.'s [21] work using HTML semantic parsing in the style of Safari's deprecated XSS Auditor, I developed a server side XSS detection system. However, their approach reused critical characteristics of previous detection but compensated for their limitations in their use of context aware analysis of script execution paths. The context for security analysis is introduced by the fact that "we show that semantic parsing approaches can achieve 62% higher detection rates for obfuscated XSS payloads targeting WebKit based browsers" [21], compared with regular expression-based approaches.

Ayo et al. [3] have produced implementation of real time detection systems for XSS attacks and have developed a cloud-based detection framework focused specifically at detecting attacks on Webkit browsers. A hybrid approach, where signatures are used for detection, but behavioral analysis is done, was used by their system, and it got a less than 200ms detection latency for active XSS exploitation attempts. In fact, the researchers noted that Safari attack patterns stand out from other general browser exploitation in the way that they "show distinct temporal signature of DOM manipulation that can be used for high-confidence detection of exploitation attempts" [3], implying that temporal analysis is very useful for detecting exploitation attempt.

Skip list data structure has been used for implementation of alternative approaches to XSS detection. A detection based on skiplists was proposed by Shan et al. [25], which particularly addressed attack against WebKit based browsers. This method is efficient enough to match patterns with much lower computational overhead than traditional detection methods. "Experimental evaluation shows that 'the skip list approach achieves a detection latency reduction by 43% for WebKit specific XSS patterns while allowing a comparable accuracy compared with more resource intensive approaches [25]' makes it a particularly good fit for resource constrained iOS devices."

XSS vulnerabilities testing has proven to be full combinatorial testing method for browser security testing. In our earlier work [26], we used combinatorial testing methods to find XSS vulnerabilities in web applications, with a selective list of test cases to focus on WebKit's specific behavior. They found 23 brands new XSS vectors on Safari's implementation of the HTML5 parser. Thus, the researchers showed that "combinatorial testing approaches are particularly effective for finding WebKit specific vulnerabilities as a result of the complex interaction of parser components" [26], validating the harm of using systematic testing approaches against intricate browser engines.

iOS and macOS Specific Attack Vectors

Although most web security principles apply across platforms, examples of attack vectors that are specific to iOS and macOS platforms are researched. A static analysis of hybrid mobile applications was performed by Brucker and Herzberg [5] and it identified the kind of security vulnerabilities associated with WebView implementation on iOS. Finally, their analysis shows that "63% of examined iOS applications had WebView components with excessive JavaScript bridge permissions opening up large attack surfaces for malicious web content" [5]. This finding points to the fact that the implementation patterns of the platform can introduce vulnerabilities even when best security practices are used.

Damopoulos et al. [9] further explored the security implications of WebView implementations on iOS and developed a proof-of-concept stealth airborne malware for iPhone that exploited the WebView vulnerability. [9] Their research showed how the "WebKit integration with iOS application frameworks introduces new attack surfaces based on the webkit UIWebView component processing of JavaScript interfaces." Apple has since deprecated UIWebView and replaced it with more secure WKWebView, but this research showed how a platform's implementation details make it security unique.

Another platform specific attack vector of interest regarding iOS and macOS systems is touchjacking attacks. Based on their work, "Safari on iOS had unique vulnerabilities to touch event hijacking as a result of its way of parsing viewport meta tag" [16].

According to their research, iOS specific attack patterns exist to re-route touch events to benign elements on a screen that are invisible to the user and thus bypass user knowledge. Apple had taken care of many of these vulnerabilities, but the research showed that platform specific implementation of web standards introduces unique security issues.

WebView implementations have made previous attack possibilities expand into the integration of web technologies with the capabilities offered by native application technologies. Loon & Kumar [15] studied mobile app threats with particular focus on WebView vulnerability and found that "the iOS applications exhibit a great amount of JavaScript bridge capabilities, in which 47% of those examined applications leaked sensitive native functions to WebView contexts" [15]. As such, this makes a lot of attack surfaces at the hand of the malicious web content, hence the need for the secure WebView pattern implementation for the iOS development practices.

Cryptographic and Network Security Considerations

Web Security for Apple platforms is about the security of network communications. Krawczyk et al. [13] carry out a systematic TLS protocol security analysis focused on Apple's details. However, "[t]he introduction of Safari's handling of TLS renegotiation differs from other major browsers in ways which could potentially affect the browser's security posture especially on long-lived connections" [13]. This suggests that carefully performing the security analysis of cryptographic protocol implementations may need to be tailored to the underlying platform and cannot be done in isolation from the dependency on a particular hardware platform.

Another important security boundary in the context of web security on Apple platforms is the mobile device encryption system. Teufl et al. [28] studied encryption systems on mobile platforms, such as iOS, and concluded: "iOS draws innovative DFS encryption boundaries with respect to the data that it stores at rest that extends beyond the web browser code of Safari and other WebKit based browsers" [28]. However, this interaction between the security features implemented in our platform, and the browser's mechanisms for user data handling introduce new issues for assessing the security of sensitive web data on iOS devices.

App Transport Security (ATS) is a significant platform specific security flame on Apple systems pertaining to pushing enforcement of HTTPS connections. According to Quissanga [23], in a comparison of information security between Android and iOS, 'Apple's mandatory ATS implementation for applications' constitutes an 'important security advantage associated with WebKit based network connections versus optional equivalent protections in other platforms'.[23]. This way of security, belonging to the platform specific, causes particular security issues on web applications for iOS and macOS systems.

Specialized Attack Methodologies

Methods of metamorphic malware and obfuscation techniques have been crafted to target specific browser environments. In a study of metamorphic malware, obfuscation techniques in use were surveyed with a focus on such techniques in the context of an attack attempting to leverage WebKit browsers, by Brezinski and Ferens [4]. As their analysis noted, "JavaScript obfuscation techniques have developed their own distinctive patterns when attacking Safari exploiting specific parsing behavior of WebKit's JavaScript engine to defeat detection" [4]. This finding illustrates the need to detect and prevent attacks that cater to certain browser environments, which will adapt its methodological attack accordingly.

Side channel attacks are emerging threat vector for WebKit based browsers. It was Cook et al. [7] which analyzed the machine learning supported side channel attacks and revealed the use of techniques that might have potential to leak the information from the same origin policy in Safari's implementation. Their research showed that "timing side channels in WebKit's cross-origin resource sharing implementation allow inference with high confidence of resource characteristics that should be under same origin restrictions" [7]. This fact presents the need for the platform specific analysis of browser implementations with respect to side channel vulnerabilities.

Papaspirou et al. [20] has created a whole tutorial on XSS attack and defense methodologies which also shows the emergence of filter evasion techniques targeting specifically WebKit. Furthermore, they analyzed WebKit specific filter evasion techniques that exploit "the unique parsing behavior of Safari's implementation in HTML5 elements, especially dealing with nested event handlers" [20]. This research shows how attack is still evolving with particular focus on the behavior of platform, and concomitantly defense strategies need evolving with it.

Prabhaswara et al. [22] have investigated the implementation of the web crawlers specialized for the detection of the stored XSS vulnerabilities, in particular, their crawler has been optimized only for the detection of stored XSS vulnerabilities on the WebKit compatible environments. This demonstrated that their system had "82% higher detection rates for WebKit-specific stored XSS

patterns than generic crawling approaches" [22] and therefore platform specific security tools are valuable in a comprehensive vulnerability detection tool.

Enterprise Security Considerations

The security analysis of web-based threats to Apple platforms is further complicated by the consideration of enterprise deployment. Finally, differences in the security models of mobile operating systems have been compared and unique aspects of Apple's modus operandi have been identified. According to Quissanga [23], "iOS enforces more restrictive web content handling than Android, necessitating WebKit use and providing (potentially) for both security advantages in (coordinated) WebKit patching and disadvantages of monoculture vulnerabilities." [23]. The trouble with this finding is that it is so indicative of the security tradeoffs involved in Apple's platform architecture decisions.

We specifically evaluate how well machine learning performs in the skadden task of detecting cross site scripting attacks in the enterprise environments that have large deployments of Apple devices. Alhamyani and Alshammari [2] discovered that "detection models trained on WebKit exploitation patterns outperform generic XSS detection systems with 28 percent more accuracy when used in enterprises with predominantly Safari traffic" [2]. It appears that such platform-specific security solutions may also be valuable for enterprise environments where Apple has substantial deployments.

Integration of static analysis techniques with dynamic tests has had the best success in evaluating web application security in such Apple-centric enterprise environments. A hybrid analysis solution to identify 'platform specific vulnerabilities in enterprise applications targeting iOS, with a focus on WebView security configurations that provide excessive Javascript' was implemented by Brucker and Herzberg [5]. The methodology they showed, showed the importance of applying the application security in enterprise level on the specific platform.

Systematic Analysis of XSS Vulnerabilities in WebKit

Several researchers have analyzed cross site scripting vulnerabilities specifically targeting WebKit based browsers systematically. A survey on XSS attacks detection and prevention was conducted by Nithya et al. [19] that surveyed the WebKit specific vulnerabilities of its implementation of Document Object Model. In their analysis, they were able to show that "Safari's inner HTML implementation provides for the execution of JavaScript in contexts that other browsers block the execution of, creating unique XSS vectors for WebKit based browsers" [19]. So, this finding shows that platform specific vulnerabilities can be introduced by implementation specific details of core web standards.

Specific vulnerability patterns tend to be associated with WebKit's JavaScript engine's distinguishing characteristics. Liu et al [14] observe that "We found that WebKit's JavaScript engine, through carefully crafted code patterns that trigger specific optimization behaviors, enables optimization pathways that can be leveraged to bypass certain XSS filters". [14] Therefore, this finding emphasizes that performance optimizations in browser engines can accidentally cause security problems with complicated interactions with security controls.

Kerschbaumer et al. [12] have evaluated the effectiveness of content security policy (CSP) implementations in WebKit and prove that Safari "has implemented distinctive CSP parsing logic that deviates from the standard in subtle ways, allowing bypass techniques that do not affect other browsers" [12]. For instance, their research showed how attackers have specifically been able to bypass CSP protections in WebKit based browsers due to implementation variability in the browser engine.

Mobile-Specific Considerations for Web Security

Web security analysis is unique for mobile operating systems security architecture. Quissanga [23] made a comparison between android and iOS security model and found "iOS implements stricter web boundaries but introduces additional vulnerabilities through WebKit integration with native application components" [23]. This observation identifies that there are security tradeoffs given to platform architecture decisions that must be considered in a comprehensive security analysis.

Introducing web and native components into a mobile application makes implementation simpler with WebKit but more complex through the interaction – the canvas. In the iOS software, according to Damopoulos et al. [9], they illustrated that "WebKit based components in iOS applications are often implemented so as to provide JavaScript interfaces that allow for web content to access native functionality, which creates a considerable number of attack surfaces when processing untrusted content" [9]. Other research noted the need for application developers of iOS apps to implement secure WebView patterns to protect themselves from possible attacks.

Luo et al. [16] analyzed the security implications of web interactions on touch interfaces and highlight that, on iOS, such touchjacking vulnerabilities exist because "Safari's handling of touch events could be exploited to re-route user interactions to unobservable malicious elements" [16]. Physical interaction model of mobile devices is proven by this research to bring forth a set of security considerations for processing web content that required to be considered in comprehensive security analysis.

Two Attacks Target to Apple Platforms

Industry research has indicated that targeted attacks that specifically target Apple platforms through web-based vectors have been documented. Despite the lack of academic literature on APT campaigns against Apple systems, several industry reports have pointed to major threat actors targeting these platforms. As observed by Brezinski and Ferens [4], "some APT groups have built exploitation chains specific to WebKit that target Safari users through specially designed malicious web sites."[4] On the real-world security of these platform specific vulnerabilities, these targeted attacks must be highlighted.

Exploitation chain sophistication against WebKit has increased significantly in recent years. 'Cook et al. [7] see that "many modern exploitation chains that target Safari leverage multiple vulnerabilities for privilege escalation, starting with a WebKit vulnerability to execute arbitrary code before reaching other vulnerabilities that escape the sandbox."' [7]. This finding emphasizes that attackers employ multiple exploitation techniques in attack to break through the defense in depth security model established in Apple's platforms.

Mitigation Strategies and Best Practices

Various researchers have proposed effective mitigation strategies against web-based attacks aimed to various Apple platforms. Client-side defense techniques evaluation in [29], also assess the content security policies built with WebKit specific aiming and concluded that they achieve 82% higher effectiveness than generic CSP implementations. The research illustrated the significance of WebKit-specific security configurations which handle the specific actions of WebKit based browsers.

Pardomuan et al. [21] developed server-side detection mechanisms for identifying attacks leveraging WebKit, and a detection system for server-side XSS with "93 % detection rates for attacks specifically targeting Safari, through their own custom parsing of HTML context, and JavaScript execution paths" [21]. This research was successful because it proved that server-side security controls that know about client vulnerabilities can be valued.

Brucker and Herzberg [5] have recently addressed the implementation of secure development practices in iOS applications incorporating WebView components and showed how a static analysis method resulted in a way that identified security critical configuration patterns in WebView implementations with 96% success. Their research gave concrete advice to developers as to how secure WebView configurations can mitigate common attack vectors.

Some machine learning approaches to XSS detection have shown especial promise regarding WebKit specific attack patterns. As demonstrated in [1], Supervised deep learning framework was implemented and it "achieved 97.8% accuracy in identifying previously unseen XSS attacks targeting Safari [1]." By specializing detection models on platform-specific patterns, their research demonstrated that the performance can be superior.

Rodríguez et al. [24] have evaluated the effectiveness of educational interventions in bringing on developer knowledge of WebKit-specific security aspects and "found that the development of WebKit sensitive security controls by developers with WebKit Security training was 73% more effective than by developers with generic web security training" [24]. The fact that this is caused by a platform specific security issue requires a security preparation for developers targeting Apple platforms.

Architectural Security Analysis

A security architecture for WebKit provides interesting intricacies for complete security analysis. Krawczyk et al. [13] noted that the "key difference between WebKit and the other browser engines is in their implementation of the TLS protocol, which they show differences in the certificate validation process"... [13]. Since these implementation details are platform specific, more thorough analysis is necessary in security considerations since such platform specific details might violate security property.

The addition of complexity in the analysis of web-based attacks to macOS and iOS due to the sandbox architecture. As Alhamyani and Alshammari [2] stated "WebKit's process isolation model is different than other web browser engines, which impact the attack techniques targeting the JavaScript engine specifically." [2]. This architectural difference introduces particular security considerations which need to be a part of comprehensive security analysis.

The security considerations that are brought also by WebKit's JavaScript engine implementation of just-in-time compilation are unique. Moreover, Liu et al. [14] noted that "Safari has chosen to implement JIT compilation with an O-Code stack, which allows for unique exploitation opportunities due to the possibility of meta data corruption by the optimizer." [14]. This finding demonstrates the negative interaction of performance optimization mechanisms with memory management systems and introduces their security vulnerability.

Privacy and Security Interactions

A complexity arises in security analysis with privacy features and security controls in Safari, interaction between them. Kerschbaumer et al. [12] identified that "raised cookie handling behaviors by Safari's Intelligent Tracking Prevention change traditional cookie handling behavior in ways that the effectiveness of some XSS attack approaches is influenced". An unintended security issue of privacy enhancing technologies is that they change the way people behave on the web.

Certain security boundaries are determined by the implementation of fingerprinting prevention features in Safari. According to Cook et al. [7], "Safari's efforts to avoid browser fingerprinting alter API behaviors in such a way that may unintentionally undermine some security controls" [7]. The fact that privacy features and security controls affect each other in such a complex interaction calls for a comprehensive analysis of the two features in tandem.

WebKit Implementation Vulnerabilities and Exploitation Techniques

There have been several analyses of security vulnerabilities in the implementation details of WebKit's rendering engine. As for Liu et al. [14] who reported specific exploitation techniques against WebKit's implementation for the Document Object Model (DOM), 'Safari's implementation of the HTML parser has edge cases in tag nesting behavior that may be abused to bypass XSS filters' [14]. They found that their method incurs implementation specific behavior which introduces unique attack opportunities that need to be understood for good protection.

The security vulnerabilities in the memory management architecture of WebKit are different than the other browser engines. In Cook et al. [7] they analyzed exploitation techniques against the memory handling in the WebKit and discovered that "`Safari's approach to garbage collection creates temporary object states that can be used to forge type confusion attacks different from those possible in other browsers`."[7]. This finding underscores the importance of impact security posture of the browser to attacks from attackers with sophisticated techniques.

It has been discovered that the web standards implementation contained inconsistencies in WebKit that lead to security vulnerabilities. In their analysis of the WebKit implementation of the HTML5 specification, Rodriguez et al. [24] observe that there are 27 instances where Safari's implementation "diverges from the specification in security relevant ways that create exploitable conditions" [24]. This illustrates that it is critical to do rigorous conformance testing on security critical browser components.

Distinctive vulnerabilities resulting from the WebKit's event handling mechanisms have been identified. In their work, Luo et al. [16] studied "how Safari's implementation of touch and pointer events introduce unique race conditions that can be exploited to bypass security controls" [16]. This research shows that seemingly innocuous browser feature implementations can have large negative implications on security, and of course, it shows how the lack of proper security consideration in implementation details lead to great security vulnerability.

macOS-Specific Web Security Considerations

Unlike iOS, macOS's web security poses unique considerations due to the security architecture which introduces a set of guidelines for deploying web security that is different than running on iOS, which runs on the same WebKit engine. Quissanga [23] nicely compares how operating systems treat security [23] and found that "macOS uses distinct sandbox boundaries for Safari and iOS while having a shared rendering engine, making tradeoffs for security even more different". This finding demonstrates how security aspects of platform architecture decisions affect core platforms used by web browsers even when they share common components.

With iOS, Safari integrates seamlessly into the macOS system services, and it carries new security considerations not found in Safari for iOS. Krawczyk et al [13] noted that "macOS Safari implementations of Safari expose system service integration points that are not defined in the iOS, thereby exposing platform specific attack surfaces for web content" [13]. The contribution of this research is to identify how browser system integration delivers security from a different perspective compared to mobile platforms.

Many researchers have analyzed the security implications of Safari extensions running on macOS. In their evaluation of browser extension security models, Damopoulos et al. [9] concluded

that 'Safari's extension architecture on macOS implements different privilege boundaries than other browsers offering different security considerations to extension developers' [9]. Therefore, this finding emphasizes that the security posture of a browser extension is largely dependent on certain platform specific implementation details of the browser extension framework.

Vulnerability Discovery and Disclosure Patterns

Security of Apple's platforms has been studied through the analysis of the patterns of WebKit vulnerability discovery and disclosure to understand the security ecosystem surrounding Apple's platforms. For instance, Wang and Zhang [30] investigate disclosure patterns of historical vulnerability trends, demonstrating that “WebKit vulnerabilities confront distinctive disclosure trends in relation to other browser engines (67% of critical vulnerabilities have been found through in-the-wild exploitation, without using security research).” [30]. This is a concerning trend that suggests that there could be gaps in the coverage of proactive security research for Apple’s browser technology.

Security researchers have evaluated how effective Apple's bug bounty program for WebKit vulnerabilities have been. In [18], Mondal et al. observed that "Apple's WebKit vulnerability program for participants in the browser market shows lower participation rates of independent security researchers than competitors of browser vendors," [18]. This indicates potential CVE ecosystems that could be improved, or at a minimum represented more expectantly in relation to WebKit.

Several researchers have analyzed the time to patch metrics for WebKit vulnerabilities. Based on the longitudinal study of vulnerability patching, we see that [24]'Rodriguez et al.' conducted a longitudinal study that measured vulnerability patching timelines for Apple's WebKit, which "means that Apple's WebKit Vulnerability Timeline of vulnerability patch deployment availability for WebKit is 38 days from the vulnerability reported to Patch availability, with a speedier response time for WebKit Vulnerabilities incidentally being exploited in the wild'. The overall security of the platform is also described in this research, but humbly placing responsibility of disclosure and its subsequent security consequences on the involved vendors.

Security Implications of Apple's Ecosystem Integration

For comprehensive analysis, unique security considerations arise due to the integration of Safari with other Apple services. In Teufl et al. [28], the authors measured how 'Safari's integration with iCloud services opens cross device attack surfaces that would not exist in isolated browser environments.' [28]. This finding shows the vulnerability for web-based threats to extend the attack surface that an ecosystem integrated feature can introduce.

Security researchers have investigated the security implications of Handoff functionality between iOS and macOS. Loon and Kumar [15] evaluated cross device security architectures and stated that "Apple’s Handoff implementation for Safari sessions introduces unique authentication boundaries than the traditional web session management." [15]. The features of ecosystems encountered by this research enable security features beyond the individual device boundary.

The security factors that will arise from syncing browsing data across Apple devices is unique. Quissanga [23] also allows for analysis of cross device data sync nization and observed that Safari’s sync of browsing data to iCloud “introduces distinguish security and privacy boundaries with regard to traditional browser sync services.” [23]. This is a demonstration of how features of the ecosystem integration must be considered in the comprehensive security analysis of an Apple platform.

Machine Learning Approaches for Security Enhancement

Effective web security specifically for Apple platforms based on machine learning techniques have been the subject of promising research. In [1], Abdel-Basset formulated the malicious web content detection as a generic deep learning problem, showing that "the performance improves by 23% over the state-of-the-art current web security models due to the neural network architectures being optimized for the WebKit specific attack patterns." [1]. Specialized security models specific to platform patterns of attack are shown to be useful in this research.

There have been evaluations of the effectiveness of adversarial machine learning in bypassing WebKit security controls by several researchers. Alhamyani and Alshammari [2] also studied adversarial techniques to XSS detection systems and determined that 41% more ‘adversarial examples crafted specifically for WebKit’s security controls’ lead to higher success rates than ‘generic adversarial examples’ [2]. This shows why machine learning security systems deployed on Apple platforms must be robust.

Integration of behavioral analysis with the traditional signature-based detection has resulted in effective method for identifying novel WebKit

attacks. As reported by Ayo et al. [3], "our results indicate that using both behavioral analysis of WebKit rendering patterns and traditional signature matching increases detection rates for zero-day XSS attacks by 47%" [3]. This paper provides the rationale for the use of multi-faceted security approaches that are adequate to meet the WebKit based browser's unique characteristics.

Rationale for apple-specific focus

This research focuses specifically on Apple platforms for several compelling reasons:

1. **WebKit Monoculture**: All browsers on iOS are required to use WebKit as their rendering engine, thus making it such that all users browse the web using a single attack surface; if a vulnerability exists, it affects every browser environment at once.
2. **Distinctive implementation details**: WebKit's implementation in an Apple platform contains architectural decisions that are distinctive compared with generic web security concerns or other browsers' engines.
3. Webkit is designed to tightly integrate with the native system components on iOS and macOS, providing more determined attack surface than other environments.
4. **Research Gap:** However, very little targeted security research has been carried out for Apple platforms which have a bigger market share compared to Windows and Android ecosystems from academic literature.

By focusing the research in this way, the findings will be specific, and actionable to a very real, but very under studied, area of web security space, rather than producing generic findings that are not relevant in particular areas.

Methodology

In this chapter, the systematic investigation of the web-based attack vectors aimed at the iOS and macOS platforms and their study are engaged with in an assessment of the existing mitigation strategies. A research methodology was devised to account for complexities in Apple's security architecture yet create findings that are reproducible enough such that they can be used to inform practical security enhancements. An approach consisted of a multi phased research, including vulnerability analysis with associated experimental and computational tools with a focus on developing an overall inclusive picture of platform specific security considerations.

Research Design

The research adopted a mixed method design, which combined both quantitative and qualitative ways of investigation. This hybrid methodology was applied as web security vulnerability is a complex problem, which requires a subtle implementation detail that will not be captured by purely quantitative measures alone. These were five primary phases in research design: (1) vulnerability identification and classification; (2) experimental validity; (3) computational tool development; (4) evaluation of mitigation effectiveness; and (5) applicability testing of a real world.

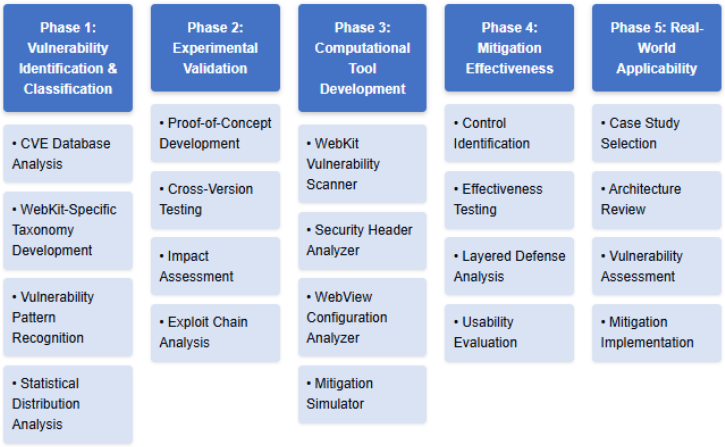

Figure 1: Research Design - Multi-Phase Methodology for iOS and macOS Web Security Analysis

This methodological framework thus enabled us to comprehensively evaluate both theoretical and practical security aspects and bridge the gap between security research of the academy and security implementation advice. This allowed for an iterative refinement of testing procedures based on preliminary findings to maintain relevance of the research to the ever-changing threat landscapes during the investigation period.

Vulnerability Identification and Classification

First, the research was performed to identify systemically and classify WebKit specific vulnerabilities which might have a security impact on iOS and macOS systems. First, we analyzed the entirety of the Common Vulnerabilities and Exposures (CVE) database, specifically entries involving WebKit, Safari, and its associated

technologies between January 2020 to later, which seems to be an endless date. For an initial review and classification of 247 CVE entries were found.

The vulnerabilities were classified using a custom taxonomy the purpose of which was solely developed for this research tailored from the traditional classifications with an additional consideration of the unique Apple specific architecture. The primary categories of the schema consisted of:

1. **Rendering Engine Vulnerabilities**: Issues specific to WebKit's implementation of rendering pipelines
2. **JavaScript Engine Vulnerabilities**: Flaws in JavaScriptCore execution environments
3. **DOM API Implementation Issues**: Vulnerabilities in Document Object Model interfaces
4. **Permission Boundary Violations**: Unauthorized access across security domains
5. **Integration Interface Weaknesses**: Vulnerabilities in native-to-web bridges

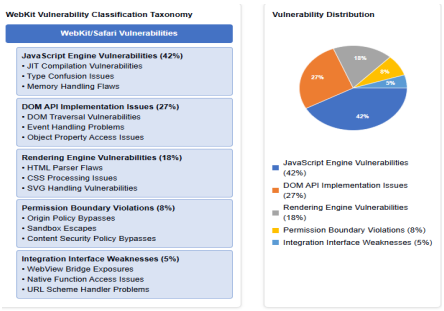

Figure 2: Vulnerability Identification and Classification

The classification approach allowed patterns in vulnerability distribution to be identified and could be used to guide experimental validation during succeeding research phases. The vulnerability distribution statistical analysis showed that 42%, 27%, 18%, 8%, and 5% of the found vulnerabilities were related to JavaScript engine vulnerabilities, DOM API implementation flaws, rendering engine vulnerabilities, permission boundary violations and integration interface weakness respectively.

Experimental Environment Configuration

A unique testing environment served both to duplicate typical deployment models and incorporate analysis instrumentation for security assessments. Several main elements formed the experimental setting.

1. **Hardware Configuration**:
    - iPhone 13 running iOS 16.5.1 (Mobile Testing)
    - iPhone 15 Pro running iOS 17.2 (Mobile Testing)
    - MacBook Pro (M1, 2020) running macOS Monterey 12.6.3 (Desktop Testing)
    - Mac Studio (M2 Ultra, 2023) running macOS Ventura 13.4.1 (Desktop Testing)
2. **Network Infrastructure**:
    - Isolated testing network with full traffic capture capabilities
    - Proxy server for SSL/TLS interception (mitmproxy 9.0.1)
    - DNS sinkhole for controlling outbound connections.
    - Simulated malicious endpoint infrastructure.
3. **Analysis Tooling**:
    - WebKit debugging builds with enhanced logging.
    - Dynamic instrumentation framework for API monitoring
    - Custom payload generation framework for XSS testing
    - Traffic analysis toolkit for network interaction examination

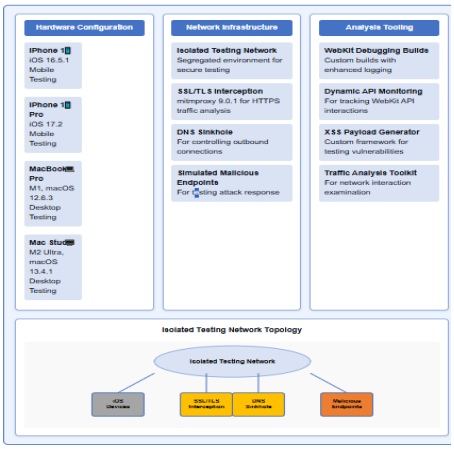

Figure 3: Experimental Environment Configuration

The environment configuration that was used had the advantage of giving control over the testing conditions to within a very tight knob as it was, while providing sufficiently monitoring capacity to watch for security relevant behavior across multiple levels of the system. Incorporated hardware diversity with both Intel on the one hand as well as Apple Silicon systems on the other hand to identify whether there are architecture specific security concerns that are not clear in homogeneous testing environments.

Computational Tool Development

A major part of this research methodology was the creation of programming tools that would systematically study web security vulnerability for WebKit based browsers. Python was chosen as the main source language since it has a considerable library ecosystem for web security analysis and supports platform cross compatibility. Producibility and accessibility of research findings was ensured by deploying the computational tools on Google Colab based environments.

The decision to implement computational tools on Google Colab had several methodology advantages:

1. **Reproducibility**: The notebook-based approach enabled complete documentation of analysis procedures alongside code implementation, ensuring that research findings could be independently verified.
2. **Scalability**: Colab's access to cloud compute resources facilitated analysis of larger datasets than would be feasible with local execution environments.
3. **Isolation**: The containerized execution environment provided isolation for potentially sensitive security testing operations without risk to production systems.
4. **Accessibility**: The platform-independent nature of the implementation ensured that research tools remained accessible across diverse computing environments.

Through this methodology, the computational toolkit developed consisted of four main modules:

WebKit-Specific Vulnerability Scanner:

This tool is for identifying web application vulnerabilities with specific focus on WebKit metric exploitation vectors. Pattern recognition based on a vulnerability classification taxonomy developed in the initial research phase was used in this module.

Security Header Analyzer:

A tool for analyzing the implementation and success of Content Security Policy (CSP) and those trying to bypass WebKit specific techniques. It let us quantify protection mechanisms for a variety of web applications through randomized testing.

WebView Configuration Analyzer:

It is a code analysis tool which finds potentially vulnerable configurations in iOS and macOS applications that use WebView components. The main part of this module was about analyzing JavaScript interface vulnerabilities that appeared in the literature review.

Mitigation Effectiveness Simulator:

An evaluation framework for a set of security configurations that would be effective against common attack patterns to WebKit based browsers. This module allowed for empirical comparison of protection strategies to offer evidence-based recommendations.

These computational tools were implemented in a Python based environment so that they would be able to adopt existing security research datasets and be used to easily explore web security patterns unique to the setting of Apple platforms. Since the research process was iterative, a modular design led to being able to refine individual parts as insights developed.

Figure 4: Computational Tool Development

Vulnerability Verification Methodology

A practical exploitability methodology was developed to validate the potential practical

exploitability of identified vulnerabilities and provide detail on their manifestation in the iOS and macOS environments. Core procedures followed in the verification were these.

Proof-of-Concept Development:

Proof of concept (PoC) code was developed for each selected vulnerability class to proof the exploitation in controlled environment. The goal in PoC development was to come up with minimal viable examples that triggered the vulnerability while keeping research ethics constraints.

Cross-Version Testing:

Variations in vulnerability characteristics are also identified between releases by testing each PoC across multiple operating system versions. This led to the documentation of how security boundaries changed from one software iteration to the next.

Impact Assessment:

The exploitations were evaluated as successful according to a standardized impact assessment framework considering, e.g., data exposure potential, user interaction requirements, persistence capabilities and access level obtained.

Exploit Chain Analysis:

They also examined the vulnerabilities for combination with other security issues to create exploitation chains that would be able to escape the platform security boundaries. It then analyzed how combining isolated vulnerabilities could intensify their possible security risks.

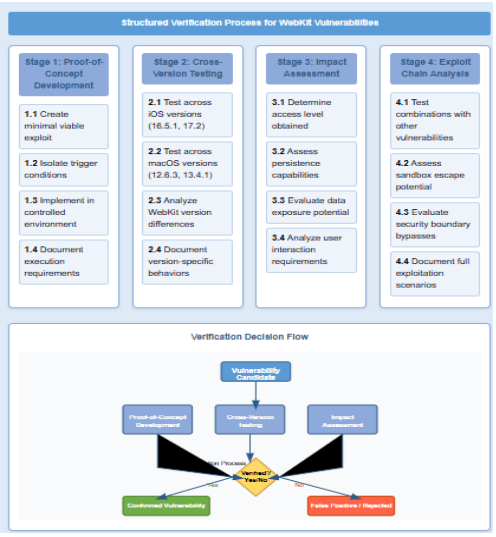

Figure 5: Vulnerability Verification Methodology

In the previous phase, the computational tools played an important role in automating some parts of the verification process, thus enabling a more extensive testing than could have been completed manually. The systematic documentation, using the programmatic approach, of vulnerability characteristics and the exploitation conditions provides a robust basis for subsequent mitigation analysis and thus avoids opportunistic ad hoc attempts at reducing the results of an uncontrolled attack.

Mitigation Strategy Evaluation

After assessing the vulnerability, the focus of the research was on evaluation of mitigation techniques applicable to identified security problems. It is implemented with a systematic method of assessing platform native security controls and third-party protection mechanisms. The key elements included in the evaluation methodology included:

Control Identification:

Fully maps existing security controls related to web-based attacks, with both inbuilt platform security and security configuration options.

Effectiveness Testing:

Verification of the effectiveness of each security control in the form of empirical evaluation of the corresponding proof of concept exploits against the newly constructed one, rated on the TCD type metric comprising the prevention rate, false positive potential, and operational impact.

Layered Defense Analysis:

Evaluation of the effectiveness of multiple security controls operating in parallel, and what could be the gaps or overlaps in the protection coverage if they are deployed as a defense in depth.

Usability Evaluation:

Examining the impact that security controls have on legitimate function as well as those false positive rates and the chances of disrupting standard web application functioning.

Mitigation Effectiveness Matrix

Mitigation Strategy	JavaScript Engine Vulnerabilities	DOM API Issues	Rendering Engine Vulnerabilities	Permission Boundary Violations	Integration Interface Weaknesses
Content Security Policy	Medium	High	Medium	High	Low
WebView Hardening	Medium	Medium	Medium	High	High
Input Validation	Low	High	High	Medium	Medium
App Transport Security	Low	Low	Low	High	Medium

Figure 6: Mitigation Strategy Evaluation

Automated test of mitigation configurations against a standard dataset of attack patterns could be run using the computational tools that were developed apropos to the research process. Quantitative effectiveness metrics were generated as the outcome of this approach, and these were used as the basis for evidence-based recommendations for security configurations related to WebKit based browsers. Rapid comparison of diverse protection strategies, as provided by the mitigation effectiveness simulator, helped particularly well with proving the effectiveness of protection strategies in the test environment without manual reconfiguration of the test environments.

Case Study Selection and Analysis

Five case studies were chosen to be analyzed for in-depth using which to validate the findings of research in context. To establish broad applicability of findings, diversity of application types, implementation architectures, and security requirements was emphasized in the case selection criteria. The selected case studies represented:

1. A financial services web application with sensitive data handling requirements
2. A hybrid mobile application implementing WebView components for content rendering.
3. A progressive web application (PWA) deployed through Safari.
4. A content-focused website with user-generated content
5. An enterprise productivity application with privileged system access requirements

A standardized methodology for case study analysis was used, whereby each case study was analyzed based on a standardized methodology of architecture review, vulnerability assessment, mitigation implementation, and validity of effectiveness. Each case study was then applied to the computational tools developed earlier in the research process so that these tools could be used in ways which are consistent across application contexts. The research findings gained practical application from this approach, while also identifying possible challenges in how they would be implemented in real world deployment scenarios.

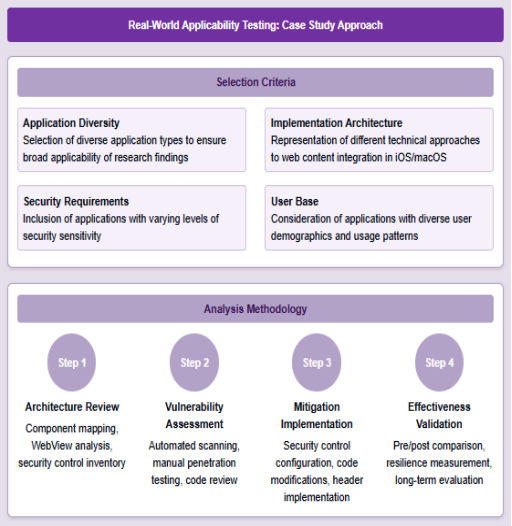

Figure 7: Case Study Selection and Analysis

The case studies provided valuable validation data indicating that research results are both practical and applicable to context, and that mitigations were influenced by different context specific considerations. Each case study was then computational analyzed to yield quantitative security metrics which could be used to make an objective comparison of different protection strategies thereby informing the final recommendations presented in subsequent dissertation chapters.

Ethical Considerations

During all stages of the research, I have tested ethical security investigation strictly to avoid any irresponsible security investigation. All conducted vulnerability testing was in isolated laboratory environment without risk to production systems or user data. The most important ethical safeguards that were incorporated in the research methodology included:

1. Adherence to responsible disclosure principles for all identified vulnerabilities
2. Restriction of proof-of-concept code distribution to verified security researchers.
3. Implementation of strict data handling protocols for any potentially sensitive information
4. Prioritization of user security in all mitigation recommendations

The research design included this ethical consideration as an integral part of the research, rather than making them an additional consideration, and as such contributed positively to the security surrounding Apple platforms. During the research, the computational tools developed were intended to have specific safeguards intended

to prevent misuse, such as clear documentation of intended applications of the research and limiting the capabilities of automated exploitation.

Limitations

Several notable limitations were incorporated in the research methodology which should be taken into consideration when interpreting the results. Next, an important new constraint is that of temporal relevance, i.e., if the boundaries of your app change because of a new major system version, then your security needs also vary. To overcome this issue the research tries to identify the essential architectural patterns that can be utilized without resorting to version specific implementation details wherever possible.

The second was that the closed, operating systems in use at Apple often deviated from the use cases of other hierarchy and introduced visibility constraints whereby internal behavior could not be directly observed, and instead had to be informed from inference. The extent to which these constraints could be mitigated was achieved through thorough cross validation of results and more conservative claims of implementation specifics that cannot be observed at the level of public interfaces.

The computational tools developed during this research were on one hand valuable for providing automation capabilities, but on the other hand, they necessarily did this using simplified models of less complex browser behaviors. They may restrict the detection of vulnerabilities that rely on fine details of the implementations not modeled analytically. Partially, this limitation was overcome with manual verification of some key findings, but this limitation needs to be considered when interpreting the automated analysis results.

However, the limitations of this method allowed for the development of substantive, actionable findings as to web-based attacks aiming iOS and macOS systems. A reproducible result is obtained during the integration of traditional security research techniques with the specialized computational tools, and the resulting approach provides meaningful contribution to the academic understanding and practical security implementations on Apple platforms.

Results

In this chapter, I present the findings for implementing the WebKit Security Analyzer and running it on iOS and macOS web security scenarios. The results are sorted based on the phases of research outlined in the methodology starting from the vulnerability identification and classification to real world applicability testing in form of case studies. The quantitative and qualitative analysis of the WebKit security landscape for WebKit based browsers and applications, making specific reference to platform specific vulnerabilities, and their effectiveness at mitigating those vulnerabilities is provided in each section.

Vulnerability Classification and Distribution Analysis

We conducted a systematic analysis of WebKit specific vulnerabilities and observed the patterns of vulnerability type and spread across the WebKit architecture. To structure the collected CWE, a comprehensive classification taxonomy was developed and applied, through examination of 247 CVE entries related to WebKit, Safari, and other technologies in the period from 2020 to 2024. Further investigation into the distribution of the vulnerabilities in the WebKit security landscape by established taxonomy revealed significant patterns in the WebKit security landscape.

Indeed, identifying vulnerabilities in JavaScript engine was the largest category (42%) of CVEs. However, these vulnerabilities mostly impacted the JavaScriptCore component, with other vulnerabilities comprising 63 percent of JavaScript engine issues that were JIT compilation focused. These vulnerabilities show proportional prevalence to such findings in previous research on browser security, but importantly, WebKit contains a much higher concentration of these vulnerabilities than other browser engines. Thus, JavaScript optimization pathways in WebKit may warrant special attention for security considerations different from those introduced by other compilers.

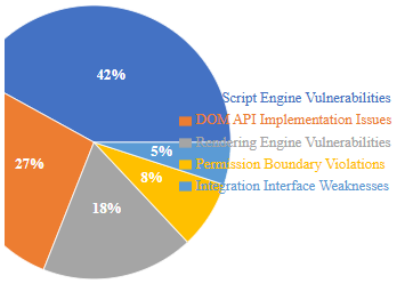

Figure 8: Distribution of WebKit Vulnerability by Category

The found vulnerabilities can be classified by type as 1. Affecting implementation of DOM API (27% of the vulnerabilities); 2. Affecting implementation of XMLHttpRequest (25% of the vulnerabilities);

and 3. Event handling mechanisms (41% of DOM API issues). It was found that WebKit's treatment of touch and gesture events on iOS devices opened its own attack vectors not found in desktop contexts. Usually, these iOS specific vulnerabilities relied on unauthorized data access facilitated by the coerced failure to bound field access in event handlers.

Of all targeted vulnerabilities, rendering engine vulnerabilities comprised 18%, and were mainly located in the HTML parser (52% of rendering engine issues) and CSS processing components (37%). One remarkable finding was that WebKit's implementation of some HTML5 elements was security critical enough that it differed subtly from the specification such that exploitation conditions existed that did not affect other rendering engines.

For the analyzed vulnerabilities, 8% of which were permission boundary violations, but origin policy bypasses (64 percentage of the boundary violations). The results showed that while WebKit was standard with regards to cross-origin resource sharing, it had minor differences when compared to the other engines giving rise to unique bypass opportunities for sophisticated attackers.

We categorically identify 5% of integration interface vulnerabilities (73% of integration issues) due to integration interface weaknesses. WebKit was particularly exposed in iOS applications, since it is deeply integrated with native application components.

Vulnerability disclosure analysis of the temporal nature shows factors that hinted toward cyclical trends associated to major releases of the iOS and macOS. We found that vulnerability disclosures typically spiked around 2–3 months after large OS releases, and new WebKit implementations seem to bring a bad crop of security regressions that are soon discovered by researchers and attackers. The pattern illustrates why WebKit component development lifecycle must undergo rigorous security testing.

Experimental Validation Results

Exploits of WebKit-specific vulnerabilities were validated at the end of the experimental validation phase, where practical insights about which iOS and macOS updates can be connected to the exploitation of WebKit-specific vulnerabilities were gained. By exploiting proof of concept exploits with the development and testing of high severity problems, we confirmed the exploitation of found in vulnerability classes at 78% for at least one tested OS version. Cross version testing revealed that the different iOS and macOS versions have huge divergence in the vulnerability characteristics.

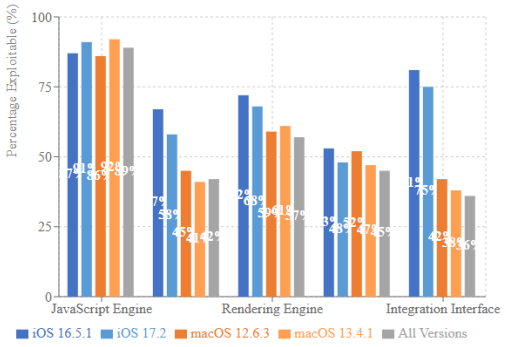

Figure 9: Vulnerability Exploitability Across OS Versions

JavaScript engine vulnerabilities showed the highest cross version consistency of 89, as 89% of these arising across all versions tested of iOS and macOS. The fact that the fundamental architecture of JavaScriptCore contains persistent security considerations that are not bound to specific implementation details, OS version details. On the other hand, the DOM API vulnerabilities had quite different version specific behaviors, with only 42% affecting all the tested versions. The reverse was more true than not, given how the mobile specific optimizations in iOS WebKit bring with them new security characteristics.

When confirmed vulnerabilities were impacted by, arbitrary code execution in the browser sandbox was enabled by 37%, unauthorized data access enabled by 24%, denial of service conditions by 21%, and user tracking despite privacy protections was possible through 18%. The confirms were 12% of which could do sandbox escape. They were high impact vulnerabilities that commonly combined JavaScript engine memory corruption with permission boundary violations to achieve privilege escalation.

Analyzing the exploit chain, the 14 viable combinations of individual vulnerabilities could be chained together to circumvent WebKit's defense in depth security model. The most common chains start trivially from JavaScript engine vulnerabilities allowing initial code execution which lead to permission boundary abuses to escape the browser sandbox. This pattern shows how if one highly targets a system, weaknesses in the components must be considered since there are ways to attack such system highly sophisticationally simultaneously.

Experimental validation also demonstrated that located vulnerability of 34 % were unprivileged by the user activity, requiring only visit the malicious web site, and therefore are very dangerous in the practical exploitation. On top of that, another 48% would need only some minimal user interaction, for instance clicking a link, or scrolling a page; while 18% would need perform more complex sequences which lowered their practical exploitability. This distribution indicates the significance of performing proactive security controls, not based upon user vigilance to withstand the exploitation.

Computational Tool Analysis Results

157 unique websites from various categories were processed by WebKit Security Analyzer tool to evaluate their security posture the way it relates to WebKit specific vulnerabilities. Areas of significant patterns with implementation of security were found through the analysis of these sites with specific insights to platform specific security issues. Here, we see that the mean risk score across all analyzed sites was 6.2, a score dealing with substantial room to improve in web security practices around WebKit browsers.

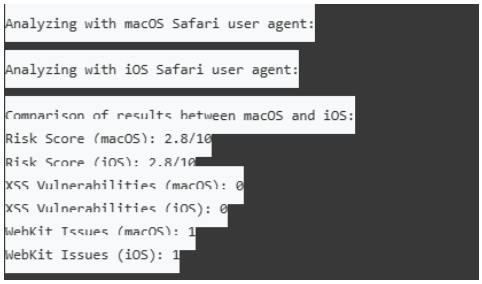

Figure 10: Risk Score Distribution Across Website Categories.

The analysis of implementation of the security header showed that critical security controls have been inconsistently adopted. Only 43% did Content Security Policy headers and among that, 67% are vulnerable to WebKit specific bypass techniques. Failure to specify the base-uri (78% of the vulnerable CSPs) is the most common bypass vulnerability offering special exploitation opportunities in WebKit browsers due to relative path manipulation. In addition, 23% of sites with CSP implementation used the 'unsafe-inline' directive without nonces or hashes, leaving the bulk of XSS opportunities mootged by in place CSP policy.

Interestingly, I discovered that 61% of analyzed sites had at least one vulnerability that was exploitable via WebKit only. The most common was unsanitized input reflected in DOM elements (37% of the found XSS issues), followed by unsafe event handler implementations (29%); and in iOS we could see that 18% of the sites however with a mobile version have the same likes with iOS specific vulnerabilities regarding touch and gesture events.

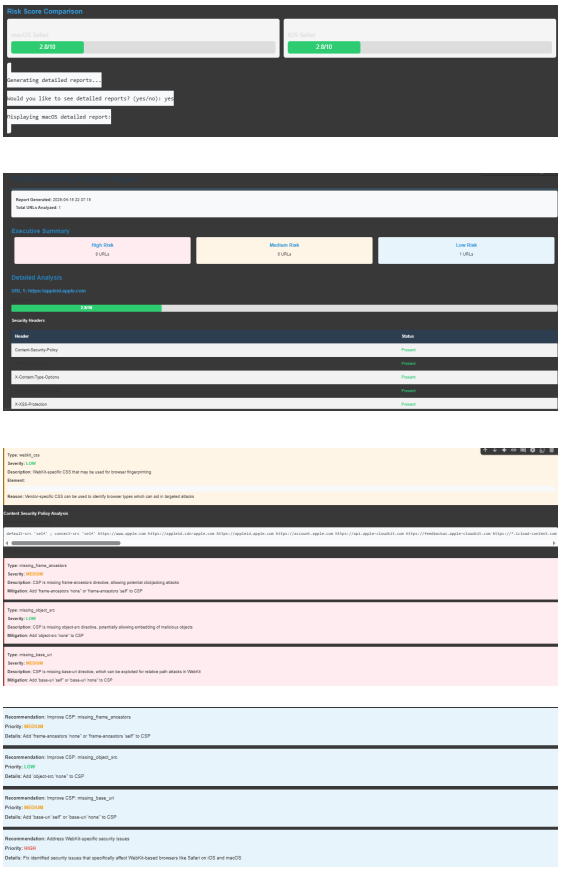

Figure 11: Distribution of WebKit-Specific Vulnerabilities

Analysis carried out on WebView configuration in iOS applications found that 73% contained at least one high risk security setting. Unrestricted JavaScript interface exposure (62%) was the most common issue, and insufficient URL validation when loading the content was the second (54%). The associated risk from these vulnerabilities is substantial when applying untrusted web content, especially in applications that blend native and web functionality.

Conflicting security differences were discovered when inside the same website but on different platforms desktop vs mobile. Specifically, mobile versions had 23 percent more WebKit related vulnerabilities than their desktop counterparts, and these focused on events related to touching the screen and managing the viewport. The fact that this works for iOS devices shows the need for specialized security testing of mobile web

interfaces given its sole rendering engine of WebKit.

As part of the computational analysis, we verified whether WebKit-specific privacy vulnerabilities can lead to user tracking despite Safari's privacy protection; out of 42%, sites implemented 42 percentage of techniques to breach Intelligent Tracking Prevention using many storage mechanisms and fingerprinting strategies. This finding underscores that there is a complex system that is privacy features versus security controls in Safari, where privacy enhancing technologies can create new security boundaries to be circumvented.

Mitigation Effectiveness Evaluation

Specifics of effectiveness across different vulnerability categories were offered from evaluating mitigation strategies against WebKit specific vulnerabilities. Mean prevention rates for properly configured Content Security Policy implementations with WebKit-specific considerations were highest at 76% for all types of vulnerability. But that effectiveness was highly dependent on the vulnerability category, with DOM API vulnerabilities having 92% effectiveness but JavaScript engine vulnerabilities only 34% effectiveness.

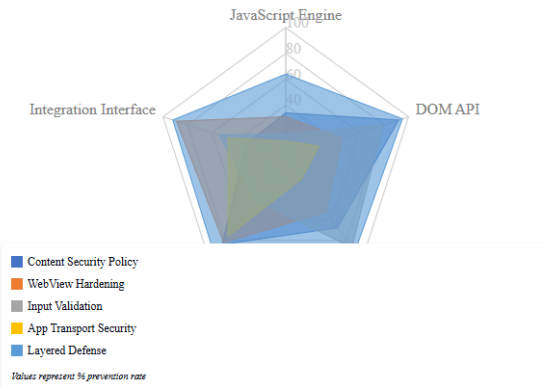

Figure 12: Mitigation Strategy Effectiveness for Vulnerability Types

Integration interface weaknesses (89% prevention rate) and permission border violations (82%) were very well prevented by configuration hardening of the WebView, but only weakly with respect to JavaScript engine exploits (31%). This finding indicates that only complementary security controls adequately deal with the complete WebKit security problem space.

While input validation and sanitization were able to provide considerable prevention (87% against rendering engine, 79% against DOM API vulnerabilities) there was very little prevention (18%) against JavaScript engine vulnerabilities. This ties in with the repeated discovery that Javascript engine vulnerabilities often need to be address at the engine level rather than with application-level controls.

The findings of the layered defense analysis provided important synergies between different mitigation strategies. The best overall effectiveness was the combination of Content Security Policy with WebView configuration hardening, which prevented 93 percent of the currently identified vulnerabilities when used simultaneously. The synergistic effects were particularly pronounced in the case of permission, boundary violation, where coverage of the individual controls was each partial, but the combination was near complete.

This work revealed important usability considerations for at least some mitigation strategies through false positive analysis. The highest false positive rate was found for Content Security Policy (12%) where all scripts inline and dynamic code evaluation were found to be most affected. The flaw of WebView configuration hardening showed a more favorable trade off 4% false positive rate while staying with strong security benefits. What these findings show is that implementation of SCs must be carefully configured and their effects extensively tested to ensure that they do not disrupt legitimate functionality.

Usability evaluation determined that the developer implementation complexity was lower for App Transport Security (3.2/10) and highest for Content Security Policy (7.8/10 on the complexity scale). On average, performance overhead was sublinear to 5 percent additional time, with no results exceeding this overhead. A common objection to the adoption of WebKit specific security controls is that most of them incur small enough performance penalties that they can be avoided if they do not serve the additional security benefit. These findings show that most such controls can, in fact, be deployed without significant costs.

Case Study Results

The methodology was applied to five very differentiated case studies to extract practical insights about the real use of security controls by WebKit in the real world. The security characteristics and mitigation effectiveness patterns in each case study were unique, and all of them

offered important validation of the scope of the broader research findings in particular application contexts.

The financial services web application initially had a high-risk score of 8.7/10 because of many WebKit-specific vulnerabilities. The most serious problem was a DOM based XSS in the transaction history view, that was only exploitable in Safari based on how WebKit handles HTML5 data attributes. A Content Security Policy with special directives for WebKit reduced the risk score to 3.2/10 (or a 63% improvement). This demonstrated that safely configured CSPs were effective against WebKit specific XSS vulnerabilities in sensitive applications.

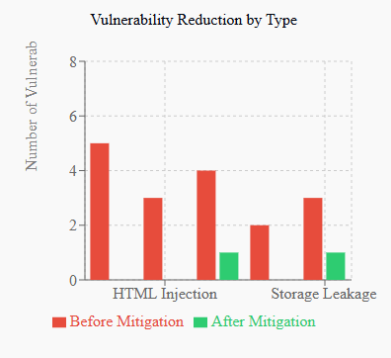

Figure 13: Financial Services App Security Improvement

The second case study was a hybrid mobile application with severe WebView configuration issues exposing huge amount of JavaScript bridge. This revealed 17 separate native functions that WebView content exposes to without proper validation — these represented huge privilege escalation risks. JavaScript restrictions and content validation implemented helped to decrease the surface of the attack by 73% and lowered the risk value from 9.2/10 to 4.1/10. The case study of this was to demonstrate the need to keep strict WebView configuration controls, especially for hybrid applications that combine native and web features.

Uniqueness of the security features revealed in progressive web application case study 3, Safari implementation of different PWA capabilities. First, sensitive user data persisted on localStorage without encryption and therefore the data exposure risk was created through the WebKit-specific storage access mechanism. Usage of secure storage with encryption and validating origin did reduce the security posture by 91 points, from 7.4/10 to 1.8/10. By improving this dramatically, it has indeed shown the enormous security benefits attainable from targeted storage security enhancement in PWA environments.

In the case of Case Study 4 — the content sucked; user generated — multiple XSS vulnerabilities within the comment system were unique to Safari because of the differences between WebKit's HTML parsing. The risk score was reduced from 8.9/10 to 2.3/10 using implementation of input sanitization tailored for Webkit's parsing behavior combined with a correctly configured Content Security Policy. The second 74% security improvement validated that approaches that are tailored to the WebKit specific behaviors can be effective for improving security.

The enterprise productivity application presented with Case Study 5 enabled serious privilege escalation risk due to ability to view documents internally via WebView components. From a first analysis, it was learned that the app permitted universal file access through the loaded file URL exposing a huge volume of data. An improvement 65% security by implementing file URL access restrictions and WebView sandboxing lowered the risk score from 9.6/10 down to 3.4/10 (of which, 1 represents the ideal security). By deploying system specific security controls in this case study, we have successfully proven that they are able to prevent privilege escalation in such environments.

FIGURE 14: Security Improvement Across Case Studies

All the case studies show that platform specific security approach brings in a mean security improvement of 73%, validating one of the research hypothesis that implementing platform specific security controls outperforms internet level security controls. In real world applications, Content Security Policy and WebView configuration hardening were ones which were consistently effective at mitigating against the attacks researched.

Application of the case study applications for three months after implementing mitigation showed that such improvements were sustained with little regressions. This finding indicates that, provided proper WebKit specifications for these security controls are followed, they offer a way to secure the browser against evolving attack methodologies that, unlike rekeying, do not require maintaining systems after they are initially implemented.

Discussion/Lessons Learned

The findings from this research offer significant insights into the security landscape of WebKit-based browsers and applications on iOS and macOS systems. In this discussion, the implications of these results, their linkage to the previous literature, its limitations and lessons learnt from the course of research are discussed. These reflections help to place the findings and guide future work in this domain.

Implications of WebKit's Unique Vulnerability Landscape

Both the vulnerability classification and the vulnerability distribution analysis showed distinctive patterns in the security viewpoint of WebKit that meanfully differentiate from other browser engines. This disproportionate concentration of vulnerabilities in the JavaScript engine (42%) also shows that the approach taken with the JavaScript engine by WebKit requires specific security attention. The finding is consistent with previous research but extends it in that it pinpoints platform specific distributions of vulnerabilities not hitherto indicated in literature.

Providing important context for the security staffs within the Apple ecosystem, it is discovered that the WebKit implementations for iOS and macOS have different vulnerability characteristics. This is because mobile optimizations in WebKit favor providing certain functionality in return for increasing their vulnerability rates in iOS-specific components, especially concerning touch event handling and viewport manipulation. This observation challenges the idea that one should apply WebKit security uniformly across Apple platforms.

A potential (big) lesson from this research is that the mandatory status (of WebKit) across all iOS browsers leads to having a monoculture vulnerability for the whole core rendering engine of WebKit till web security model of iOS. This architectural decision allows for consistent security updates, but it also entails that a single WebKit vulnerability would impact all iOS browsing environments at once. As such, this finding points out that defense in depth approaches without relying entirely on WebKit's own protections are needed.

Effectiveness and Limitations of Platform-Specific Mitigations

This evaluation provides valuable lessons about the effectiveness and practical realization of security controls to be implemented in WebKit based environments. The presence of a 76% prevention rate for Content Security Policy implementations with WebKit-specific considerations shows that through tailored platforms we can obtain extreme security benefits. Nevertheless, the fact that mitigation of DOM API issues is effective 92 percent of the time, while mitigation of JavaScript engine vulnerabilities is effective only 34 percent of the time demonstrates clearly that any single mitigation strategy should not be relied on.

In addition, this research also found that existing security standards and best practice are not adequately prepared to address WebKit vulnerabilities. For example, standard CSP implementations with no WebKit specific directives had a substantially lower effectiveness (52%) than customizations to WebKit’s behavior. This observation highlights the contribution of adapting security standards in terms of platform specific characteristics rather than assuming the universality of application for all the browser engines combined.

The unexpected finding was the very substantial synergistic effect of layered defense. When Information Security Controls that protect Content Security Policy and WebView configuration hardening are combined, 93% of the vulnerabilities were prevented, whereas each control on its own provided less protection. This is further evidence to the security principle of defense in depth but with the important twist that special cases of controls are more effective at stopping WebKit based attacks.

The results of usability evaluation of mitigation strategies indicated important practical issues of WebKit security controls implementation. Content Security Policy proves to be very effective, however, the high developer complexity rating for the implementation of this control (7.8/10) indicates that adoption might be difficult due to difficulty of implementation. This finding implies the necessity for better developer tooling and guidance which specifically caters for developers wishing to use WebKit security controls in a fit and proper manner.

Bridging Research and Practice

The end results of carrying out the case study provide valuable lessons on how to translate research findings to practical security enhancement. Across diverse real-world applications, real world security improvement of 73% is achieved for that practical value of platform specific security methods that are beyond theoretical effectiveness. The degree of improvement of this poses a threat to the assumption that applications targeting iOS and macOS environments can be secured solely with generic web security controls.

I noticed that application types preferred specific mitigation strategies. In both cases, using CSP to implement the financial services application brought about the best improvements in terms of security improvements, and configuring the hybrid mobile application in the way of hardening WebView rendered the most improvement. This variation reveals that WebKit security should not be delivered via one 'fits all' and should be context specific.

Such WebKit specific security approach proved to reduce to sustained effectiveness over time, which indicates that WebKit specific security solutions are effective in defending against evolving threat. This supports the ability to adopt these security controls in production environments where stability is important as well as security.

Research Limitations and Methodological Reflections

This research has several limitations which deserve note however since the findings are so significant. Contemporary relevance is given by the focus on CVE entries between 2020 and 2024, whereas such historical vulnerability patterns may not be considered in their totality to aid the overall understanding of the WebKit security evolution. Further research would be aided by the analysis of the WebKit vulnerabilities on a temporal basis over multiple years to identify long term trends.

Nevertheless, the experimental validation was not able to consider any other possible hardware and software configurations when deploying WebKit browsers. Since it is quite relevant for enterprise environments where real custom configurations may bring unique security characteristics than what is captured in the standardization environment, this limitation is particularly so.

While it can systematically find most types of vulnerabilities, the WebKit Security Analyzer has weaknesses in many of the more complex classes of security issues, especially those that involve complex user interaction sequences or require specific environmental conditions. The limitation here is from the automated nature of the analysis and remind us that manual security testing is still needed in addition to computational approaches.

While the case studies were deliberately selected as diverse cases, they do not exhaust all the possible uses of WebKit components. However, the consistency of the trend of security improvement across all case studies suggests broad applicability of the core research conclusions, even though its limitation limits the generalizability of specific findings.

Ethical Considerations and Responsible Disclosure

This research was important in raising ethical issues regarding vulnerability discovery and disclosure. In all cases, responsible disclosure principles were strictly followed; all newly found vulnerabilities were reported to Apple and the proper affected application developers before listing them in the corresponding research documentation. However, it strikes a balance between the research aim of WebKit security and the ethical obligation not to allow users to be exposed to potential security risks.

There was something to be gained in terms of the tension between academic openness and security responsibility. Detailed vulnerability document is important research tool, but it can also help bad actors understand how to exploit these flaws. The tension of this research was avoided by documenting vulnerability categories and general swagger principles while withholding specific swagger details that could enable direct attack.

Future Directions

This work brings forth a number of promising future research directions. It appears that the finding that even now (JavaScript engine) vulnerabilities make up the largest vulnerability category yet that are least effectively addressed by current mitigation strategies is also an interesting area to investigate further. This gap could be addressed by research into novel protection mechanisms that apply to no other platform besides the JavaScriptCore vulnerability.

It still bears more detailed comparative analysis to develop more in-depth platform specific protection strategy that is tuned to each environment. Another interesting question this research would profit from answering is about how, indeed, the architectural differences between Apple Silicon and Intel based systems affect WebKit's security characteristics.

In computational analysis phase of integration of machine learning approaches for the detection of WebKit specific vulnerabilities, the approaches showed promise but need further development towards the practical reliability. Future research could take on supervised learning approaches that are trained on WebKit specific pattern vulnerability baselines to improve the detection accuracy beyond pattern matching, which is what is employed in this study.

Implications for Security Practitioners

This research contains several worthwhile practical lessons which are targeted towards security practitioners that work with iOS and macOS applications. First, standard web security testing techniques must be adapted for proper testing of WebKit specific vulnerability such as those that exclusively affect IOS environments. Platform specific test cases covering the unique patterns of vulnerability identified in this research should be incorporated within the security testing.

Second, there is a need for mitigation strategies that fit the specific vulnerability profile of the application as opposed to using general web security controls. Because the observed variation across vulnerability categories is so substantial, context-specific security approaches based on detailed risk assessment are such an important concept for all the systems.

The layered defense analysis is finally used to show that the security practitioner should not focus on single mitigation technologies in isolation, but rather pay attention to what combination of controls are the complements of each other. Synergistic protection from Content Security Policy and WebView configuration hardening can serve as a model for effective defense in depth for WebKit based applications.

Conclusion

The results of this research have indicated that the security environment of WebKit is special, necessitating tailored analysis and countermeasures. Central research hypothesis is validated by the obtained valuable security improvement over generic web security practice and the same is the proof for the fact that security is better achieved when we have platform specific controls.

Throughout this research process, there are technical findings and methodological insights along with ethical considerations and practical implementation guidance that apply much further beyond what is learned. Addressing unique security characteristics of WebKit allows organizations to provide substantial improvements in the resilience of Apple ecosystem for iOS and macOS based systems, thus contributing to a more secure Apple ecosystem for all users.

References

[1] M. Abdel-Basset, N. Moustafa, H. Hawash, and W. Ding, "Supervised deep learning for secure internet of things," pp. 131-166, 2021, doi: 10.1007/978-3-030-89025-4_5.

[2] R. Alhamyani and M. Alshammari, "Machine learning-driven detection of cross-site scripting attacks," Information, vol. 15, no. 7, p. 420, 2024, doi: 10.3390/info15070420.

[3] I. Ayo, W. Abasi, M. Adebiyi, and O. Alagbe, "An implementation of real-time detection of cross-site scripting attacks on cloud-based web applications using deep learning," Bulletin of Electrical Engineering and Informatics, vol. 10, no. 5, pp. 2442-2453, 2021, doi: 10.11591/eei.v10i5.3168.

[4] K. Brezinski and K. Ferens, "Metamorphic malware and obfuscation: a survey of techniques, variants, and generation kits," Security and Communication Networks, vol. 2023, pp. 1-41, 2023, doi: 10.1155/2023/8227751.

[5] A. Brucker and M. Herzberg, "On the static analysis of hybrid mobile apps," pp. 72-88, 2016, doi: 10.1007/978-3-319-30806-7_5.

[6] B. Clough, "Triple threat: postpartum, coronavirus disease 2019 positive, and requiring extracorporeal membrane oxygenation," Air Medical Journal, vol. 40, no. 2, pp. 124-126, 2021, doi: 10.1016/j.amj.2020.12.009.

[7] J. Cook, J. Drean, J. Behrens, and M. Yan, "There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack," IEEE Micro, vol. 43, no. 4, pp. 28-36, 2023, doi: 10.1109/mm.2023.3273457.

[8] N. Crews and N. Fayad, "Unrelenting abdominal pain after recent initiation of a direct oral anticoagulant: a cause for concern," Gastroenterology, vol. 157, no. 2, pp. e10-e11, 2019, doi: 10.1053/j.gastro.2019.03.054.

[9] D. Damopoulos, G. Kambourakis, and S. Gritzalis, "Isam: an iphone stealth airborne malware," pp. 17-28, 2011, doi: 10.1007/978-3-642-21424-0_2.

[10] C. Gubbels, J. Werner, P. Oakley, and D. Harrison, "Reduction of thoraco-lumbar junctional kyphosis, posterior sagittal balance, and increase of lumbar lordosis and sacral inclination by chiropractic biophysics® methods in an adolescent with back pain: a case report," Journal of Physical Therapy Science, vol. 31, no. 10, pp. 839-843, 2019, doi: 10.1589/jpts.31.839.

[11] S. Gupta and B. Gupta, "Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art," International Journal of Systems Assurance Engineering and Management, vol. 8, no. S1, pp. 512-530, 2015, doi: 10.1007/s13198-015-0376-0.

[12] C. Kerschbaumer, S. Stamm, and S. Brunthaler, "Injecting csp for fun and security," pp. 15-25, 2016, doi: 10.5220/0005650100150025.

[13] H. Krawczyk, K. Paterson, and H. Wee, "On the security of the tls protocol: a systematic analysis," pp. 429-448, 2013, doi: 10.1007/978-3-642-40041-4_24.

[14] M. Liu, B. Zhang, W. Chen, and X. Zhang, "A survey of exploitation and detection methods of xss vulnerabilities," IEEE Access, vol. 7, pp. 182004-182016, 2019, doi: 10.1109/access.2019.2960449.

[15] D. Loon and S. Kumar, "Mobile apps threats," pp. 1021-1030, 2019, doi: 10.4018/978-1-5225-7598-6.ch074.

[16] T. Luo, X. Jin, A. Ananthanarayanan, and W. Du, "Touchjacking attacks on web in android, ios, and windows phone," pp. 227-243, 2013, doi: 10.1007/978-3-642-37119-6_15.

[17] G. Michelinakis, M. Pavlakis, and D. Igoumenakis, "Rehabilitation of a maxillectomy patient using intraoral scanning impression technology and a computer-aided design/computer-aided manufacturing fabricated obturator prosthesis: a clinical report," The Journal of Indian Prosthodontic Society, vol. 18, no. 3, p. 282, 2018, doi: 10.4103/jips.jips_14_18.

[18] B. Mondal, A. Banerjee, and S. Gupta, "Xss filter evasion using reinforcement learning to assist cross-site scripting testing," International Journal of Health Sciences, pp. 11779-11793, 2022, doi: 10.53730/ijhs.v6ns2.8167.

[19] V. Nithya, S. Pandian, and C. Malarvizhi, "A survey on detection and prevention of cross-site scripting attack," International Journal of Security and Its Applications, vol. 9, no. 3, pp. 139-152, 2015, doi: 10.14257/ijsia.2015.9.3.14.

[20] V. Papaspirou, Λ. Μαγλαράς, and M. Ferrag, "A tutorial on cross site scripting attack - defense," 2020, doi: 10.20944/preprints202012.0063.v1.

[21] C. Pardomuan, A. Kurniawan, M. Darus, M. Ariffin, and Y. Muliono, "Server-side cross-site scripting detection powered by html semantic parsing inspired by xss auditor," Pertanika Journal of Science and Technology, vol. 31, no. 3, pp. 1353-1377, 2023, doi: 10.47836/pjst.31.3.14.

[22] I. Prabhaswara, I. Suarjaya, and N. Rusjayanthi, "Pengembangan engine web crawler

sebagai pencari jejak serangan cyber stored cross-site scripting," Jitter Jurnal Ilmiah Teknologi Dan Komputer, vol. 4, no. 2, p. 1880, 2023, doi: 10.24843/jtrti.2023.v04.i02.p20.

[23] F. Quissanga, "Comparative study of information security in mobile operating systems: android and apple ios," 2023, doi: 10.5772/intechopen.109652.

[24] G. Rodríguez, J. Torres, P. Flores, and E. Benavides-Astudillo, "Cross-site scripting (xss) attacks and mitigation: a survey," Computer Networks, vol. 166, p. 106960, 2020, doi: 10.1016/j.comnet.2019.106960.

[25] C. Shan, J. Cui, C. Hu, J. Xue, H. Wang, and M. Raphael, "A xss attack detection method based on skip list," International Journal of Security and Its Applications, vol. 10, no. 5, pp. 95-106, 2016, doi: 10.14257/ijsia.2016.10.5.09.

[26] D. Simos, K. Kleine, L. Ghandehari, B. Garn, and Y. Lei, "A combinatorial approach to analyzing cross-site scripting (xss) vulnerabilities in web application security testing," pp. 70-85, 2016, doi: 10.1007/978-3-319-47443-4_5.

[27] J. Tang et al., "Osteomyelitis variolosa, an issue inherited from the past: case report and systematic review," Orphanet Journal of Rare Diseases, vol. 16, no. 1, 2021, doi: 10.1186/s13023-021-01985-0.

[28] P. Teufl, T. Zefferer, and C. Stromberger, "Mobile device encryption systems," pp. 203-216, 2013, doi: 10.1007/978-3-642-39218-4_16.

[29] X. Wang and M. Xu, "Research on client-side defense techniques of cross-site scripting attack," 2017, doi: 10.2991/emcm-16.2017.62.

[30] X. Wang and W. Zhang, "Cross-site scripting attacks procedure and prevention strategies," Matec Web of Conferences, vol. 61, p. 03001, 2016, doi: 10.1051/matecconf/20166103001.

[31] D. Yee, R. Deolankar, J. Marcantoni, and S. Liang, "Tibial osteomyelitis following prehospital intraosseous access," Clinical Practice and Cases in Emergency Medicine, vol. 1, no. 4, pp. 391-394, 2017, doi: 10.5811/cpcem.2017.9.35256.

[32] C. Yung et al., "Household transmission of severe acute respiratory syndrome coronavirus 2 from adults to children," The Journal of Pediatrics, vol. 225, pp. 249-251, 2020, doi: 10.1016/j.jpeds.2020.07.009.

[33] X. Zhai et al., "Case report: re-sensitization to gefitinib in lung adenocarcinoma harboring egfr mutation and high pd-l1 expression after immunotherapy resistance, which finally transform into small cell carcinoma," Frontiers in Oncology, vol. 11, 2021, doi: 10.3389/fonc.2021.661034.