Clickjacking and Double-Clickjacking: Attack Mechanisms and Prevention Algorithms

Hosung Yun
Department of Software Engineering
Kangnam University
Gyeonggi-do, Korea
hosung0504@kangnam.ac.kr

Youngeun Cho
Department of Data Science
Kangnam University
Gyeonggi-do, Korea
eun030423@kangnam.ac.kr

Jiyoon Lee
Department of Software Engineering
Kangnam University
Gyeonggi-do, Korea
ict22jy@kangnam.ac.kr

Jungsoo Park
Department of Computer Engineering
Kangnam University
Gyeonggi-do, Korea
jspark@kangnam.ac.kr

***Abstract*— This paper presents the implementation of a clickjacking attack and a novel hacking method based on it, termed "double-clickjacking." It also details the design of a Chrome extension to prevent these attacks. We move beyond the conventional technique where a user clicks on a seemingly legitimate button but is redirected to an unintended destination. The purpose is to develop a Chrome extension that blocks malicious activities such as inducing a double-click to insert malicious actions between clicks, executing a double-click action with just a single click, or covertly redirecting to a malicious website in the background without the user's knowledge. We demonstrate that the developed Chrome extension can effectively defend against not only traditional clickjacking but also double-clickjacking.**

Keywords—Clickjacking, Double-ClickJacking, Web Security, Temporal Heuristics, Event Interception, Formal Methods

I. Introduction

Double-clickjacking, an evolution of traditional clickjacking [1], [8], [13], is emerging as a new web hacking paradigm and presents a formidable challenge to web security. This sophisticated technique deceptively exploits the user's habitual double-clicking behavior. An attacker associates different events with the first and second clicks in a user's sequence, executing malicious actions without explicit consent. By leveraging this user behavior, double-clickjacking can lead to severe consequences such as background malware downloads and information leakage. Although solutions for clickjacking are being actively developed [9], [10], [12], the concept of double-clickjacking remains largely unfamiliar, making an in-depth analysis imperative.

This study, therefore, aims to systematically analyze the core attack scenarios and operational principles of double-clickjacking. Our goal is to clarify its inherent risks and provide a solid academic foundation for the development of robust security countermeasures. Based on clickjacking principles [2], our research focuses on analyzing the mechanisms and creating security algorithms for four novel double-clickjacking attack types: 'background download,' 'forced click,' 'new window background redirection,' and 'existing window background redirection' [6]. By referencing existing techniques for identifying malicious redirections [11], we will first apply preventative measures to clickjacking, specifically by blocking the opening of new windows, to eliminate the potential for double-clickjacking attacks.

II. Double-Clickjacking Attack Types

Generally, clickjacking is an attack technique where a malicious element is overlaid on a button that the user intends to click, causing the user to unintentionally click the malicious element. In modern systems, this can be defended against by setting security headers for the code and requiring double confirmation for important buttons. However, in double-clickjacking, the malicious action does not originate from the button's function itself. Fig. 1 illustrates a fundamental example of the aforementioned double-clickjacking attack.

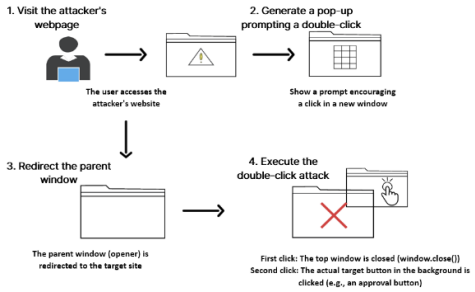

Fig. 1. Examples of Double-ClickJacking Basic Attack

Instead, it is a method where a malicious element is executed in the interval between the first and second clicks. This represents a significant security threat, as an action intentionally performed by the user can lead to a malicious outcome, leaving the user vulnerable to an attack without their knowledge. Furthermore, as new malicious activities utilizing double-clickjacking emerge, we construct four additional scenarios—A, B, C, and D—and implement an algorithm to address them.

A. Background Downloads

This scenario describes a double-clickjacking attack that induces an unintentional file download by making the user believe they are interacting with a legitimate UI component. The attacker exploits the user's rapid clicking habits to bind different events to the first and second clicks, respectively. Although the user perceives they are performing a single action (such as a double-click for security confirmation), in reality, a malicious script designed by the attacker is executed, causing a file to be downloaded in the background [3], [7]. The step-by-step execution of this attack is summarized in TABLE I.

TABLE I. BACKGROUND DOWNLOAD SUMMARY TABLE

Background Download Execution Order		
Step	***Act***	***Description***
First Click	Click Recognize and Standby	The first time a user clicks a button, changes the button style and sets the time until the second click to wait for a double click.
Second Click	Double click detection	If a second click occurs within the set time, the script considers it a double click, releases the set timer and initiates a malicious behavior.
Performing a Malicious Behavior	Download Malicious Files	The function is called, dynamically creating tags to download malicious files, and immediately clicking to run file downloads without the user's knowledge.
Traces removed and deceptive	Redirect the screen	Immediately after the file download starts, the function is called and redirects the current page to another URL. This tricks the user as if the normal process has been completed.

B. Force Click

This scenario, conceived from the three elements that compel input commands in clickjacking [2], expands a user's single click into two intentionally consecutive events to lead the user to a malicious website. Although the user clicks a legitimate button only once, an attacker's script programmatically triggers a second event after a time delay, creating a result equivalent to a double-click.

The core of this technique is to create a discrepancy between the user's action and the actual system response. The user is led to believe that their single click only produced visual feedback, while internally, a malicious action is scheduled and executed after a predetermined time. The step-by-step execution of this attack is summarized in TABLE II.

TABLE II. FORCE CLICK SUMMARY TABLE

Force Click Execution Order		
Step	***User Experience***	***Hidden behavior of the script***
Click Initial	A single click of the button changes the style immediately, leading to the belief that it has been processed normally.	Detect the click to change the button style immediately, and schedule a short delay in the next action.
Disguise the progress	After a while, the style changed again, and I mistook the whole process as part of the normal process.	Change the style back to the scheduled action to trick the user, and further schedule the final page move.
Go to a malicious site	As soon as you thought the process was over, you were forced to go to an unintended site.	Execute the last scheduled code, forcing the user to a malicious site.

C. New Window Background Redirection

This scenario describes a sophisticated attack technique that covertly redirects the original (parent) window to a malicious website in the background while drawing the user's attention to a new pop-up window. The attacker triggers two actions simultaneously with the user's double-click. First, a new window, appearing as a legitimate confirmation process, is brought to the foreground. Second, while the user's focus is fixed on this new window, the address of the original window, now obscured, is changed to a malicious page [4]. The step-by-step execution of this attack is summarized in TABLE III.

TABLE III. NEW WINDOW BACKGROUND REDIRECTION SUMMARY TABLE

Redirect to New Window Background Execution Order		
Step	***User Experience***	***Hidden behavior of the script***
Separation of events	When the user double-clicks the button, a new confirmation window immediately appears in front of the screen.	It detects a double click and runs both actions simultaneously. One is to display a fake confirmation window that will catch the user's attention.
Background manipulation	The user concentrates on the new confirmation window and is not aware of any changes in the previous window.	At the same time as floating a fake window, the original window is moved to a malicious site without the user's knowledge.
distraction and control	While the user's eyes are on the new confirmation window, the original window behind has already changed to a different page.	Taking advantage of the user's distraction, the hidden window completes loading of malicious pages and prepares to attack.

D. Redirect existing window background

This scenario represents the archetypal attack model of double-clickjacking based on window overlay and UI redressing. In this case, the attacker prepares two visually identical windows—a decoy (lure) window and a malicious window—aligned at exactly the same screen position. The user is induced to perform a double-click on the visible lure window. The first click immediately closes the lure window, thereby exposing the malicious window hidden directly beneath it at the exact same location. As a result, the user's second click is delivered directly to the dangerous button on the malicious window, which may lead to unintended permission grants or information leakage [5]. The step-by-step execution of this attack is summarized in TABLE IV.

TABLE IV. REDIRECT EXISTING WINDOW BACKGROUND SUMMARY TABLE

Redirect existing window background Execution Order		
Step	***User Experience***	***Hidden behavior of the script***
Background manipulation	A normal pop-up window appears to open, and the original window appears to be at the back.	As soon as the pop-up opens, you are ready to replace the original window behind it with a forged malicious page without the user's knowledge.
Double-click induction	View messages such as "Session Reauthentication" in the pop-up window and recognize that you need to double-click the button.	Design the first and second clicks to be applied to different windows by inducing the user to double-click.
Click Intercept	I think I double-clicked the button as instructed, but in reality, the second click is pressed on another button.	Close the pop-up window itself with the first click to reveal the fake page, and cause the user's second click to press the page's dangerous button.

III. Proposed Model

The double-clickjacking prevention algorithm proposed in this paper operates within a Chrome Extension environment to block four types of illicit activities [6]: background downloads, the use of synthetic events to mimic a double-click from a single click ('single-click → double-click mimicry'), background redirections originating from new (popup) windows, and background redirections within the existing window.

This integrated defense algorithm is designed with a sophisticated three-stage structure: it first distinguishes between legitimate user clicks and anomalous actions initiated by attack scripts, then blocks the execution of any behavior identified as malicious, and finally, prompts the user for direct confirmation in ambiguous cases. This entire process is facilitated by synergistic communication between a content script, injected directly into web pages to monitor their behavior, and the extension's core background script, which handles the main functionalities.

A. Core Function Interception (API Hooking [14])

The initial step of our defense mechanism involves intercepting (hooking) core browser functions that are susceptible to exploitation by an attacker. The moment a web page loads, the extension injects a custom JavaScript file into the page's JavaScript execution environment. This script then assumes control over the following critical functions, masquerading as their native counterparts.

- window.open: As the primary function for opening new windows and pop-ups, we take control of it to capture all relevant information, such as the intended destination URL.
- location.assign / location.replace: These functions forcibly redirect the current page. We monitor calls to these functions to prevent attackers from redirecting users to advertisement or phishing sites.
- <a>.click() / <form>.submit(): While these are standard link-clicking and form-submission functions, they can be programmatically invoked by a script to trigger unintended actions. We secure control over them to detect such forced executions.

Crucially, the invocation of these hooked functions does not lead to their immediate execution. Instead, a signal containing detailed information about the pending action (e.g., "an attempt is being made to perform X action on the current page") is transmitted to the background script, which serves as the extension's central processing unit, to request a definitive judgment.

B. Identifying Suspicious Behavior by Analyzing State and Time

While a signal is sent to the background script, the injected content script concurrently assesses whether the current situation constitutes an attack by leveraging two critical clues: a 'state flag' and 'time'.

When a user clicks on the page, if an attempt to open a new window is detected through a window.open call, the defense script immediately records a 'suspicious state' flag in the tab's sessionStorage. This is tantamount to marking the page with a label: "A pop-up attempt has just occurred on this page. All subsequent activities are to be considered suspicious!" This flag is maintained until the user performs another legitimate click on the page.

From the moment the 'suspicious state' flag is set, the defense logic operates with heightened sensitivity and utilizes temporal information to thwart attacks.

Blocking Background Redirection of the Existing Window: Previous research [11] utilized code readability and complexity analysis to detect malicious redirections that change the location property via window.opener. By de-obfuscating code and applying signature-based detection, this approach could detect attempts to covertly redirect the original window in double-clickjacking attacks. However, being based on static analysis, it has limitations in blocking a diverse range of real-time attacks, including synthetic clicks, background redirections, and programmatic navigation.

To address these limitations, this study proposes a detection and blocking mechanism within a browser extension environment that combines dynamic overriding with time- and state-based analysis. Specifically, we override key in-page APIs such as window.open, location.assign/replace, history.pushState/replaceState, anchor.click, and form.submit. This allows us to immediately block suspicious programmatic behaviors that occur directly after a user's click. Furthermore, delayed redirections initiated by an attacker within a short interval after a click, or redirection attempts from a background page, are also identified and blocked as malicious. We also leverage pointerdown and click events to record trusted user gestures, thereby neutralizing synthetic events or automated click attempts. This real-time dynamic blocking technique can defend against a broader spectrum of attack vectors than existing static analysis-based detection, fundamentally neutralizing the sophisticated malicious redirection attempts associated with double-clickjacking.

Blocking Background Redirection of a New Window: The algorithm also prevents a newly opened window from covertly navigating to a different address while it is not visible (i.e., in the background). The script checks the current page's document.visibilityState. If a redirection occurs while the page is not visible, it is identified as a malicious act and is consequently blocked.

By integrating state and time analysis in this manner, our approach fundamentally neutralizes sophisticated, time-delayed attacks that deceive the user's perception, as well as all forms of covert page manipulation occurring in the background.

C. User Confirmation (Explicit Permission Acquisition)

The algorithm treats every attempt to open a new window as a potential threat, maximizing the defense's robustness by deferring the final decision to the user.

- New Window Confirmation Prompt: When a window.open attempt from Stage 1 is detected and its signal is received by the background script, the script does not immediately open a window to the requested URL. Instead, it first displays a small confirmation prompt on the screen with the content: "This site is attempting to open a new window to the following address. Do you want to allow it?". Only when the user explicitly clicks the 'Allow' button in this prompt is the new tab opened. This simple procedure effectively blocks all unwanted pop-ups and new windows.

- Automatic Cancellation of Suspicious Downloads: The final stage of this defense logic is to prevent malicious file downloads. The user's action of clicking 'Allow' in the aforementioned confirmation prompt is logged as a 'trusted user approval' record. In contrast, the initial window.open attempt is logged as a 'suspicious event occurrence'. If a download for a potentially dangerous file type (e.g., .exe or .zip) is initiated, and this download occurs immediately following the 'suspicious event occurrence' without an intervening 'trusted user approval' record, it is deemed a malicious download initiated covertly and is automatically canceled.

Thus, through this three-tiered defense framework—combining function interception, state and time analysis, and user confirmation—the algorithm functions as a robust countermeasure against double-clickjacking, moving beyond simple pop-up blocking to comprehensively defend against sophisticated redirections and malicious file downloads.

D. Conditions for a Successful Double-clickjacking Attack

The formal conditions for a successful double-clickjacking attack are defined in Table V.

TABLE V. FORMAL DEFINITION OF DETECTION CONDITIONS

Condition Name	Mathematical Definition	
	formula	***ID***
AttackSuccess	*Exist t_v < t_o < t_c such that (A_p(t_o) and A_c(t_c) and (t_c - t_v ≤ τ))*	(1)

The variables t_v, t_o, and t_c are timestamps that indicate the temporal order of relevant events: t_v denotes the time of the first user click, t_o denotes the time of the preparatory action, and t_c denotes the time of the execution action. The constraint $t\_c - t\_v \leq \tau$ in Equation (1) requires that the time elapsed from the first click to the execution action falls within the permissible double-click interval τ. In this context:

- *A_p(t_o)* represents the malicious preparatory action (e.g., initiating a background download) executed immediately after the first click.
- *A_c(t_c)* represents the malicious execution action performed immediately after the second click.
- τ denotes the maximum permissible time interval for a double-click.

E. Observation and Marking

The conditions for detecting a user-initiated new window (Open Event) are defined in Table VI.

TABLE VI. FORMAL DEFINITION OF DETECTION CONDITIONS

Condition Name	Mathematical Definition	
	formula	***ID***
Open Event	$M\_open^{a} = t^{b} \leftrightarrow O(t)^{c}$	(2)

[a.] M_open: Timestamp of the last user-initiated window open event.

[b.] t: The timestamp of the event being evaluated.

[c.] O(t): A predicate that is true if a monitored event (e.g., a new window opening) is confirmed at time t.

The system monitors all potential core API calls, such as opening new windows, navigation, dynamic creation of anchor/iframe elements, and the generation of blob URLs. If a detection is confirmed according to Equation (2), a session marker *M_open* is recorded.

F. Time- and State-Based Anomaly Predicate

This predicate determines whether a specific action *P(t_p)* constitutes a legitimate user action or is part of an attack attempt. The determination is based on temporal intervals and the page's visibility state.

Table VII defines the four conditions used to detect suspicious user behavior.

TABLE VII. FORMAL DEFINITION OF DETECTION CONDITIONS

Condition Name	Mathematical Definition	
	formula	***ID***
WithinGrace	$M\_open\ existence^{a}$ *and* $(t\_P^{b} - M\_open^{c} \leq G^{d}$	(3)
SuspiciousDelay	$L\_min^{e} < t\_P^{b} - t\_U^{f} < L\_max^{g}$	(4)
HiddenRedirect	$V(t\_P)^{h} = hidden$ *and* $(t\_P^{b} - t\_U^{f} < L\_max^{g})$	(5)
isProgrammatic	$P(t\_P)^{i}$ *and* $\neg U(t\_P)^{j}$*(isTrusted = false etc)*	(6)

[a.] M_open existence: A boolean indicating a user-initiated window open event has occurred.

[b.] t_P: Timestamp when the popup is created.

[c.] M_open: Timestamp of the last user-initiated window open event.

[d.] G: Grace period threshold following a user action.

[e.] L_min: Minimum time threshold for a suspicious delay.

[f.] t_U: Timestamp of the last user interaction (e.g., click, keypress).

[g.] L_max: Maximum time threshold for a suspicious delay.

[h.] V(t_P): Visibility state of the popup.

[i.] P(t_P): A predicate that is true if the popup was created programmatically.

[j.] U(t_P): A predicate describing the popup's origin and trust status.

For a specific action, the following equations are evaluated to check for suspiciousness:

- Equation (3) serves as the post-pop-up predicate, which assesses actions immediately following a pop-up.
- Equation (4) is the suspicious delay predicate, used for actions that occur after a questionable delay.
- Equation (5) is the hidden (background) redirection predicate.
- Finally, Equation (6) provides a programmatic action marker.

When an action is detected, these formulas are evaluated to perform a suspicion check.

G. Decision Rules

The decision rule outputs a final action—either allowing the behavior, blocking it, or requesting user confirmation—based on the results of the predicate evaluations.

This is formally defined by a decision function D which takes an event E as input and returns a value from the set (Allow, Block, RequireConfirm).

Table VIII presents the final mathematical formulation for either blocking or allowing suspicious behavior, based on the conditions defined above.

TABLE VIII. FORMAL DEFINITION OF DETECTION CONDITIONS

Policy Name	Mathematical Definition	
	formula	***ID***
Post-Popup Programmatic Block	*if* $WithinGrace(t\_P^{a})$ *and* $isProgrammatic(t\_P^{a}) \rightarrow D(P(t\_P)^{c})^{b} = Block$	(7)

Policy Name	Mathematical Definition	
	formula	***ID***
Suspicious Delay Block	*if SuspiciousDelay(t_P[a]) → D(P(t_P)[c])[b] = Block*	(8)
Hidden Window Redirection Block	*if HiddenRedirect(t_P[a]) → D(P(t_P)[c])[b] = Block*	(9)
Require Confirmation for New Window	*if O(t_o)[e] → D(O(t_o)[e])[b] = RequireConfirm*	(10)
Final Allowance Condition	*AllowOpen(t[f]) ↔ (t[f] = t_A[g] and t_A[g] > t_o[d])*	(11)
Download Blocking Rule	*if D(t_D[h], url)[b] and url not S[i] and ((t_D[h] – M_open[j]) ≤ G_D[k]) and ¬((t_D[h] – t_U[l]) ≤ G_D[k]) → D(D)[b] = Block*	(12)

[a] t_P: Timestamp when the popup is created.

[b] D(...): Decision function for a given event, e.g., Block, Allow.

[c] P(t_P): A predicate that is true if the popup was created programmatically.

[d] t_o: Timestamp when a new window is opened.

[e] O(t_o): An event or operation of opening a new window, occurring at time t_o.

[f] t: Current time or the time of evaluation.

[g] t_A: Timestamp when an allowance condition is met.

[h] t_D: Timestamp when a file download is initiated.

[i] S: A set of trusted source URLs.

[j] M_open: Timestamp of the last user-initiated window open event.

[k] G_D: Grace period threshold for downloads.

[l] t_U: Timestamp of the last user interaction (e.g., click, keypress).

The following equations represent the specific decision rules:

- (7) Post-Popup Programmatic Block: Blocks programmatic actions that occur immediately after a pop-up is generated.
- (8) Suspicious Delayed Redirection Block: Blocks redirections that are initiated after a suspicious delay.
- (9) Hidden Window Redirection Block: Blocks redirection attempts from a window that is hidden or in the background.
- (10) New Window Opening → User Confirmation: Dictates that any attempt to open a new window requires explicit user confirmation.
- (11) Final Allowance Condition: The actual opening of a tab is permitted only if a user confirmation marker *t_A* is recorded.
- (12) Automatic Download Cancellation Rule: This is the decision rule for automatically canceling a download.

H. Theorem and Proof

The following theorems formally state the guarantees (defense properties) provided by the double-clickjacking defense.

1) Blocking of Programmatic Manipulation Following a Pop-up

For any tab, if $M\_open = t\_o$ and $P(t\_P)$ occurs in that tab, and $t\_P - t\_o \leq G$ and $isProgrammatic(t\_P)$, then $P(t\_P)$ is blocked.

Proof: By the definitions provided thus far, $WithinGrace(t\_P)$ holds true. Rule (7) states $WithinGrace(t\_P)$ *and* $isProgrammatic(t\_P) \rightarrow D(P) = Block$. Therefore, when this condition is satisfied, D returns Block.

2) Blocking of Delayed Suspicious Redirection

For any navigation action $P(t\_P)$, if $L\_min < t\_p - t\_U < L\_max$, then $P(t\_P)$ is blocked.

Proof: By definition, $SuspiciousDelay(t\_P)$ holds true. As Rule (8) is $SuspiciouslyDelay \rightarrow D = Block$, the action is blocked.

3) Blocking of Background Redirection from a Hidden Window

If $V(t\_P) = hidden$ and $t\_P - t\_U < L\_max$, then the navigation $P(t\_P)$ is blocked.

Proof: By definition, $HiddenRedirect(t\_P)$ holds true. The action is blocked by Rule (9).

4) Automatic Cancellation of Suspicious Downloads

If a download $D(t\_D, url)$ occurs, and it satisfies the conditions *url not S*, $t\_D - M\_open \leq G\_D$, and $(t\_D - t\_U) > G\_D$, then the download is automatically canceled.

Proof: The conditions precisely match the premise of Rule (12). Therefore, the outcome is $D(D) = Block$.

5) Guarantee of New Window Initiation via User Allowance

When a new window request $O(t\_o)$ occurs, the double-clickjacking defense immediately prompts for the user's explicit confirmation. The actual URL is opened only when the user confirms and provides t_A. Consequently, no automatic redirection or action through a new window can occur without user confirmation.

Proof: By Rule (10), $O(t\_o)$ returns $RequireConfirm$. According to the allowance condition in Rule (11), the actual opening is permitted only if $t\_A > t\_o$ (the user's confirmation) is provided. Therefore, the opening is not permitted without confirmation.

IV. Analysis and Evaluation

A. Correspondence experiment results

Fig. 2 illustrates the step-by-step operation of the proposed double-clickjacking prevention browser extension in a real attack environment.

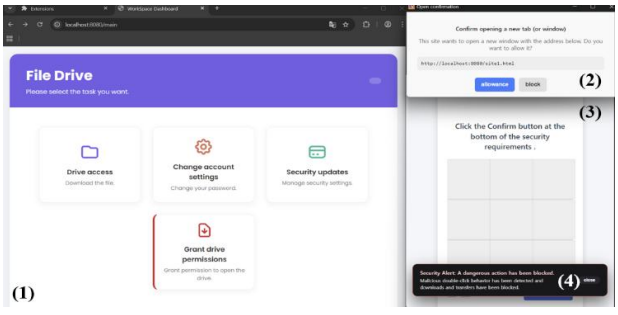

Fig. 2. Double-clickjacking defense process using the proposed browser extension: (1) Attacker main website, (2) Preemptive security confirmation popup for user consent, (3) Authentication window, (4) Blocking message displayed after detecting a double-clickjacking attempt.

Fig. 2-(1) represents the attacker's main website constructed for a double-clickjacking attack, which is

designed to appear as a legitimate service page to mislead the user.

When the user attempts to access the site, a preemptive security confirmation popup is displayed as shown in Fig. 2-(2) by the proposed defense logic. This step serves as a preventive protection phase, allowing the user to explicitly verify whether the accessed site is indeed the intended destination before any deceptive interaction can proceed.

Only if the user grants permission does a normal authentication window appear, as shown in Fig. 2-(3). This authentication process is visually identical to a legitimate authentication procedure. However, immediately after the authentication is completed, when the attacker attempts to exploit the second click for double-clickjacking to steal information or trigger additional malicious actions, the input is detected as malicious and is immediately blocked by the proposed extension, as shown in Fig. 2-(4).

In the current experiment, the implementation primarily focused on blocking download-based attacks. In addition, the same detection mechanism was also verified to effectively block other major forms of click-based attacks, including forced click manipulation that induces unintended double-click actions, as well as background redirection attacks targeting both newly opened windows and existing parent windows. These results experimentally demonstrate that the proposed system is not limited to a single attack type but provides scalable and comprehensive security protection against a wide range of click-based attack scenarios.

B. Complexity

The proposed double-clickjacking prevention system is designed to perform only a constant number of operations for each click event, enabling immediate detection and blocking in real-time browsing environments. This design allows the system to provide stable security protection without introducing noticeable input delay.

In addition, a comparative evaluation of the web browsing environment before and after applying the browser extension showed no significant performance degradation, including page loading latency, input response delay, or increased CPU and memory usage. These results experimentally confirm that the proposed system operates as a lightweight security mechanism that supports real-time protection with minimal performance overhead in practical user environments.

V. Conclusion

In this paper, we defined the threat model of Double-Clickjacking and proposed a real-time detection and blocking algorithm to counteract it. The proposed approach analyzes the time interval (Δt) between clicks, the timing of high-risk events, and user focus transitions, enabling it to block not only conventional clickjacking attacks but also stealthy malicious behaviors such as background downloads and page redirections.

Through theoretical analysis and real-world attack experiments, we demonstrated that the proposed method can reduce the success probability of double-clickjacking attacks to virtually zero within the defined threat model. A Chrome extension–based Proof-of-Concept implementation further verified that effective attack blocking can be achieved without interfering with legitimate user interactions.

A key contribution of this work is that the proposed method is formulated as a platform-independent algorithm, which enables its extension toward a general-purpose double-clickjacking defense mechanism at the web framework level, similar to CSRF protection.

As future work, we plan to validate the proposed algorithm across diverse browsers and platforms, extend it for integration into server-side frameworks such as Spring Boot, and incorporate machine learning–based anomaly detection to further enhance its robustness against unknown attack patterns.

Acknowledgment

This work was supported by the IITP(Institute of Information & Coummunications Technology Planning & Evaluation)-ITRC(Information Technology Research Center) grant funded by the Korea government(Ministry of Science and ICT)(IITP-2025- RS-2020-II201602)

References

[1] S. Y. Jeon, G. M. Jo, J. H. Im, and M. G. Lee, "Analysis of recent Clickjacking attacks on Social Network Services," in *Proceedings of the Korean Institute of Information Scientists and Engineers (KIISE)*, June 2015, pp. 2120–2122.

[2] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schechter, and C. Jackson, "Clickjacking: Attacks and Defenses," Carnegie Mellon University / Microsoft Research, Aug. 2012.

[3] J. H. Oh, C. T. Im, and H. C. Jeong, "Technical Trends and Response Methods of Drive-by Download," *Communications of the Korean Institute of Information Scientists and Engineers*, pp. 112–116, Nov. 2010.

[4] K. S. Park, "A Method to Block Clickjacking using Java EE Architecture," M.S. thesis, Graduate School of Engineering, Korea University, Seoul, Korea, Aug. 2014.

[5] J. W. Min, S. M. Jeong, and T. M. Jeong, "Design of a Mini-Page Based Clickjacking Prevention Browser," in *Proceedings of the 38th Korea Information Processing Society (KIPS) Fall Conference*, vol. 19, no. 2, Nov. 2012, pp. 1072–1075.

[6] X. Xing, S. Zhang, H. Wang, et al., "Understanding Malvertising Through Ad-Injecting Browser Extensions," in *Proceedings of The Web Conference (WWW) 2015*, May 2015.

[7] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "Mitigating Drive-by Download Attacks: Challenges and Approaches," in *Proceedings of the 2nd Workshop on Information Security (INETSEC)*, 2009, pp. 52–62.

[8] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, "Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites," in *Proceedings of the Web 2.0 Security & Privacy (W2SP) Workshop*, July 2010.

[9] M. Balduzzi, M. Egele, E. Kirda, D. Balzarotti, and C. Kruegel, "A Solution for the Automated Detection of Clickjacking Attacks," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS '10)*, Apr. 2010, pp. 135–144.

[10] J. Y. Kim and K. J. Lee, "A development of a web phishing detection service based on dynamic diagnosis," in Proceedings of the Korean Institute of Electrical Engineers Conference, Jul. 2024, pp. 2876–2877.

[11] K. M. Choi and C. S. Hong, "Detection of malicious redirection through code complexity and readability analysis," in Proceedings of the Korean Institute of Information Scientists and Engineers Conference, Jun. 2017, pp. 1033–1035.

[12] U. U. Rehman, W. A. Khan, N. A. Saqib, and M. Kaleem, "On detection and prevention of clickjacking attack for OSNs," in Proceedings of the 11th International Conference on Frontiers of Information Technology (FIT), Dec. 2013, pp. 160–165.

[13] A. S. Narayanan, "Clickjacking vulnerability and countermeasures," Dept. of Information Technology, Salalah College of Technology, Sultanate of Oman, Dec. 2012, pp. 7–10.

[14] W. K. Kim, U. Y. So, and K. Seong, "Study on API hooking detection methods in Windows," Journal of Advanced Navigation Technology, pp. 884–893, 2009.