Work-in-Progress: Detecting Browser-in-the-Browser Attacks from Their Behaviors and DOM Structures

Ryusei Ishikawa
Graduate School of Information Science and Engineering, Ritsumeikan University
ishikawa@cysec.cs.ritsumei.ac.jp

Soramichi Akiyama
College of Information Science and Engineering, Ritsumeikan University
s-akym@fc.ritsumei.ac.jp

Tetsutaro Uehara
College of Information Science and Engineering, Ritsumeikan University
t-uehara@fc.ritsumei.ac.jp

***Abstract*—Browser-in-the-Browser (BITB) Attacks are a new phishing attack technique that first surfaced in 2022. This attack displays an HTML element that mimics a popup window within a browser, prompting the user to enter confidential information such as their account and password into a fake login screen. The popup windows used in this attack have standard browser features (e.g., minimizing and maximizing) and show spoofed web pages (e.g., google.com). Because these features also include a fake URL bar, it is difficult to identify a BITB attack by checking the domain, unlike traditional phishing attacks. In this paper, we define four key behaviors of BITB Attacks and propose a novel detection method based on these behaviors. We also build a BITB detection system for Blink-based browsers that runs in real-time. Real-time detection is achieved by our design principle that leverages the buffer period of the completion of the web page rendering and the user trying to submit confidential information. Our evaluation shows that our system can find all 64 BITB PoCs gathered from GitHub and the TF2 Campaign, which is a real-world example of BITB Attacks.**

I. Introduction

Phishing attacks are a significant security threat. They steal confidential information, account information, or money via fake pages or emails [1] [2] from users with marginal IT literacy. The impact of phishing attacks on companies outstrips that on individuals because they can leak confidential information, install malware, and disrupt business operations [3]. In addition, the number of phishing attacks nearly doubled from April 2021 to April 2022 [4].

Although impactful once succeeded, effective countermeasures are proposed for many of the existing phishing attacks. For fake web pages that mimic well-known web pages and trick users into accessing them [5] [6] to steal confidential information, checking the domain of the web pages is an effective countermeasure.

However, there is no effective countermeasure to a recently reported phishing attack named the *Browser-in-the-Browser (BITB) Attack*. BITB Attacks were first reported by mr.d0x in 2022 [7]. As shown in Fig. 1, this attack displays an HTML element that mimics a popup window within the browser, prompting users to enter confidential information into a fake login screen [8]. Concretely, the attacker creates a web page that offers attractive content, such as discount event tickets and uses social media to lure the unsuspecting into accessing it. When they access the page, a message claims that a login is required to gain benefits, prompting users to enter their confidential information into a login screen that is mimicked by a popup window. The domain of this popup window can be easily spoofed by a previous technique, rendering existing countermeasures ineffective.

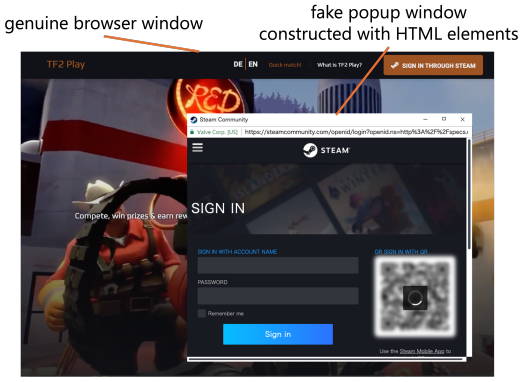


Fig. 1: Example of Browser-in-the-Browser Attacks.

The challenges in detecting BITB Attacks are twofold.

1) Both rule-based and AI-based methods do not straightforwardly work. BITB Attacks can be implemented with various HTML elements (e.g., one can use either `div` or `iframe` to create a popup), making it hard to simply find them by matching with a few representative patterns. AI-based methods require a large dataset for training to begin with, but there is no way to collect them other than manual inspection of web pages currently.
2) The detection mechanism must work in real-time. Here, being real-time means that detection must be done before the user finishes sending confidential information into a fake popup. Otherwise, the user has to wait until the detection has finished, which could bother users and make them stop using a detection mechanism.

To mitigate these challenges, this paper proposes a novel detection system for BITB Attacks that can robustly detect attacks in real-time. The main ideas are (1) to investigate the

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2025
28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-9-0
https://dx.doi.org/10.14722/madweb.2025.23010
www.ndss-symposium.org

behaviors of a web page but not the *appearance*, and (2) to use the time when the user browses and enters confidential information for detection. Here, the behaviors of a web page means how it reacts to user's inputs (e.g., clicking a certain button). This alleviates the need to distinguish different implementation methods for the same behaviors. We define four key behaviors of BITB Attacks by analyzing Group-IB's report [8] and real-world example of BITB Attacks and propose our method based on these four key behaviors. The contributions of this paper are as follows:

1) This paper is the first to propose a detection method for BITB Attacks as far as we know.
2) Based on the proposed method, we implement a BITB Attack detection system for Blink-based browsers, named BitD. It utilizes the Chrome Devtools Protocol and does not require any modification to the browser or special libraries so it can be easily used by novice users.
3) We evaluate the detection rate of our method on 64 BITB Proof-of-Concepts gathered from GitHub and a real-world example of BITB Attacks. They mimic various browsers and OSs. Our method could detect all of them as BITB Attacks while existing AI models could only detect a few of them.
4) We evaluate the execution time of BitD and found that it runs with negligible overhead for all the test cases.

II. Research Background

A. Cases of BITB Attacks

Phishing campaigns using BITB Attacks have already been conducted. The following provides three cases:

1) According to Group-IB's report [8], a phishing page used a BITB Attack in 2022 to steal Steam [9] account credentials. In this phishing campaign, users were directed to a phishing page by a URL posted on Discord and prompted to login so that they could participate in PC game tournaments or purchase discount tickets for events. When they attempted to login, an HTML element was displayed that mimicked a browser popup. This pseudo-popup element includes common browser functions (minimize, maximize, close, TLS certificate verification, etc.) and displays a fake domain.
2) Google's Threat Analysis Group explained that they had already spotted BITB Attacks used by multiple government-backed actors in March 2022 [10]. This attack is believed to be related to the war in Ukraine and targets users of the popular Ukrainian portal site (www.i.ua).
3) We discovered a phishing campaign exploiting BITB Attacks in the wild in 2024. We refer to the discovered phishing campaign as the *TF2 Campaign*. Users are prompted to login to the web page to participate in a gaming event. We discovered the TF2 Campaign via a Reddit post [11]. The post attached a screenshot of the BITB Attack with a URL (e.g., https://tf2workshop.example.com/login), but that page was no longer working. So we used a wildcard query for part of the URL (e.g., domain:tf2*.example.com) on URLScan [12], and found several fragmentary files for web pages that seemed to be identical. By combining these files, we were able to reconstruct the web pages used in the TF2 Campaign. In this paper, we refer to these pages as the *TF2 dataset*.

B. Technical Details of BITB Attacks

This section explains how the BITB Attacks are constituted by analyzing Group-IB's report and the TF2 dataset. Web pages used in BITB Attacks consist of two components: *Pseudo-Popup* and *Pseudo-Popup Builder*.

Pseudo-Popup. The Pseudo-Popup component prompts users to enter confidential information such as IDs and passwords. It consists of the following three elements:

1) HTML Forms
2) Buttons mimicking browser functions
3) A Draggable window

The *HTML Forms* prompt users to enter IDs and passwords. This form is created by replicating the login forms of commonly used services like Google.

The *Buttons mimicking browser functions* include maximize, minimize and other UI elements that are designed to deceive users that popups are real. Using the User-Agent header from the user's HTTP request, these UI elements can be displayed to match the user's OS and browser. These buttons can be implemented using both CSS property modifications and DOM modifications.

The *Draggable window* is allows the user to drag the Pseudo-Popup with their mouse, just like a real browser window. This is implemented using JavaScript such as an `Element.mousedown` event. The Draggable window also has a fake address bar, a fake title bar, and other fake UI elements.

Pseudo-Popup Builder. The Pseudo-Popup Builder component creates a Pseudo-Popup. This component is primarily implemented using one of the following two methods:

1) Modification of CSS properties
2) Modification of DOM

In the *modification of CSS properties* method, CSS properties that visually remove elements from the browser window, such as `display:none`, are pre-set on the Pseudo-Popup element. When the user clicks a button labeled like *Login with Google*, these properties are changed to display the Pseudo-Popup. In the *modification of DOM* method, the HTML elements of the Pseudo-Popup are generated using JavaScript method such as `Document.createElement()`.

C. Challenges in BITB Attack Detection

Here we elaborate on the challenges in establishing a countermeasure against BITB Attacks to motivate our work. We identify two categories of challenges.

First, both rule-based and AI-based methods do not straightforwardly work. This can further be drilled down as follows.

1) BITB Attacks cannot be detected by merely checking the domain of a web page, unlike traditional phishing

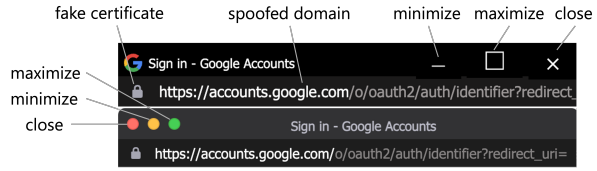


Fig. 2: Popup window title bar that mimics a browser window (Top: Windows, Bottom: macOS).

attacks. As shown in Fig. 2, the Pseudo-Popup of a BITB Attack has a user interface that resembles a regular popup window. Therefore, the attacker can trick a user into believing the spoofed domain shown in a fake URL bar.

2) BITB Attacks are implemented using standard HTML, CSS, and JavaScript with no special JavaScript APIs or browser APIs used. This makes it hard to distinguish them from legitimate web pages. In addition, BITB Attacks can be implemented in various types of HTML elements.
3) It is difficult to maintain a blocklist of known suspicious web pages because an attacker can create a BITB Attack on any domain, regardless of the domain that it spoofs (e.g, google.com).
4) Existing AI models are trained on general phishing web pages. This means that using them for detecting BITB Attacks yields low accuracy. This will be proven in Section V-C.

Second, the cost that can be spent on a detection mechanism is marginal. This can further be drilled down as follows.

1) Since BITB Attacks target individuals with low IT literacy, the deployment cost of such a detection mechanism should be relatively low so that a normal daily user could handle it.
2) To prevent the transmission of confidential information without letting the user wait, the detection mechanism should run in real-time.

D. Related Work

mr.d0x [7] defined the BITB Attack and published the first Proof of Concept (PoC). They describe two countermeasures for it. The first is to drag popup windows to the edge of the browser because those created by a BITB Attack cannot be dragged outside the actual browser frame. However, this method's effectiveness depends on the IT literacy of the user. The second uses a browser extension by Andrew [13]. This method only investigates a few elements (`iframe`, `frame`, `embed`, and `object`), but BITB Attacks can be implemented without relying on them, making this method insufficient.

There is research on rule-based detection of BITB Attacks. Alessa et al. [14] analyzed the mechanism and threat level of the BITB Attacks, proposing countermeasures. However, their proposed countermeasures are similar to those suggested by mr.d0x, making them insufficient. Asheesh et al. [15] proposed a rule-based phishing detection algorithm called PhishSpy, which can detect various phishing attacks, including BITB Attacks. However, their detection method resembles Andrew's [13], making it easily circumvented.

Phishing detection based on machine learning has been proposed, but they are not yet mature enough to detect BITB Attacks. Asiri et al. [16] proposed a real-time phishing detection system using machine learning called PhishingRTDS. They claim that it can detect three types of phishing attacks including BITB Attacks. Nevertheless, PhishingRTDS utilizes datasets from general phishing sites that appeared in 2020 for training and evaluation, whereas BITB Attacks is a technique that was publicly disclosed in 2022 [7]. Therefore, this dataset does not sufficiently include BITB Attacks, and it is unclear whether BITB Attacks are being detected. Liu et al. [17] proposed PhishIntention, which uses machine learning to infer the intent of each component from screenshots of phishing pages. PhishIntention cannot detect BITB Attacks, because the appearance of BITB Attack web pages does not differ from web pages implementing legitimate login popups. As shown in Section V-C, this methods have low detection rates for BITB web pages.

III. Key Behaviors of BITB

Based on the Group-IB's report [8], we identify four outstanding behaviors of BITB Attack web pages as follows.

- When a user clicks the login button, a Pseudo-Popup is displayed.
- Words commonly used on login pages, such as *Login*, are displayed.
- A form requires the users to input confidential information.
- Confidential information is transmitted over the Internet.

The main idea of our work is that the above four behaviors can be found automatically and robustly to detect a BITB Attack. To achieve this, we search for HTML elements that satisfy the following four criteria from a web page under suspicion:

1) *Post Rendering*: elements created after the web page's rendering is completed;
2) *Vocabulary in Screenshot*: elements that contain specific words in their screenshots;
3) *Inputtable*: elements where the user can input text;
4) *Send-to-External*: the text entered by users is sent over the Internet.

Post Rendering refers to elements created when the user clicks a button or triggers a timer (e.g., `setTimeout`). In a typical legitimate web page, popup windows are created by user activation.

Vocabulary in Screenshot refers to elements that display specific words, such as *Login*. We refer to such words as *Login Vocabulary*. A Pseudo-Popup implementing a login screen often includes Login Vocabulary. We use screenshots of elements to detect Login Vocabulary because an attacker could convert them to images and display them to users to evade text-based detection mechanisms.

Inputtable refers to elements that allow text input, such as `input` tags. This element must be detected because phishing

pages aimed at stealing confidential information always require users to input text.

When the elements satisfying these criteria also execute the Send-to-External action, they are determined to be a BITB Attack. This action sends the content entered by the user to a server on the Internet. Web pages conducting BITB Attacks always send confidential information over the Internet.

IV. BitD: A Detection System

A. Proposed System

We propose BitD, a robust detection system for detecting BITB Attacks in real-time. Fig. 3 shows the overview of BitD. When the user opens a web page, BitD first executes the Initialize Detectors function, then it retrieves the DOM of the opened web page and generates instances of each detector. After initialization, BitD Detect Loop stage and the Send-to-External stage begin. During this stage, the Post Rendering Element Detector first detects the Post Rendering elements, then detects the Vocabulary in Screenshot elements and the Inputtable elements, and saves each detector results except the Send-to-External Detector in the Detector Result Holder. This stage is repeatedly executed every two seconds since this was the optimal interval time based on our experiment results. Finally, the BITB Classifier at the end of the whole detection system determines whether or not the web page conducts the BITB Attacks based on the results of all Detectors output. The Send-to-External Detector identifies whether the information entered by the user is sent over the Internet. It monitors all the network requests sent from the browser and treats any request containing the user's input as a Warning Request. If a Warning Request is detected, this request is temporarily blocked, then the results from each detector are retrieved from the Detector Result Holder. If all the target elements are detected, an alert is triggered for a BITB Attack. If it is determined not to be a BITB Attack, the paused network request will then be resumed.

BitD is designed to operate in real-time by using the following methods:

1) BitD pre-executes each detector before the user submits their confidential information so that it can display an alert at the moment confidential information is sent.
2) BitD does not need to retrieve the entire DOM even if it is updated. BitD caches the entire DOM, and it only retrieves the modified parts when it is updated.

B. Detection of Post Rendering Elements

This function is achieved by periodically detecting modified or newly appended elements that are rendered through the CSS property or DOM. It monitors the states of the CSS properties that could remove visible elements from the browser window. To collect browser behaviors, we use the Chrome Devtools Protocol (CDP) [18].

C. Detection of Vocabulary in Screenshot Elements

BitD takes screenshots of multiple DOM elements detected in Section IV-B and processes them through Optical Character Recognition (OCR) to check if the elements contain any Login Vocabulary. OCR is performed using the Tesseract OCR [19]. Login Vocabulary is a set of words frequently used on login pages. The methods of obtaining this vocabulary are as follows: (1) Collect the content of popular login pages from a domain ranking [20]. (2) Extract vocabulary that appears 10 times or more, and then remove non-alphabetic characters and words with three or fewer letters. We obtain the following 11 words as Login Vocabulary as a result: `email`, `sign`, `password`, `privacy`, `forgot`, `account`, `policy`, `google`, `help`, `username`, `phone`.

D. Detection of Inputtable Elements

This detector searches for Inputtable elements that are text input elements such as `input` and `textarea`. This detector scans the DOM for Inputtable elements, and then extracts only ones that are child elements of the elements detected in Section IV-B. This is to exclude Inputtable elements outside of the fake popup window.

E. Detection of Send-to-External Actions

We inspect the value of input elements and network requests on web pages to detect the transmission of confidential information to be sent over the Internet. The issue here is that CDP cannot directly obtain user's input. To avoid this, we use the following two steps: (1) obtaining the user's input and (2) inspecting the network request.

In step (1), the user's input is obtained by the JavaScript runtime and then sent to the CDP runtime. The sub-issue here is that CDP does not provide an easy means of communication between the JavaScript runtime to do so. The idea is to use the JavaScript runtime to send the user's input to the network, and then the CDP runtime captures this request to obtain the user's input. In other words, the JavaScript runtime sends the user's input twice: once for BitD to receive, and once for the attacker. To capture user's input sent to the network, we add an unique header to distinguish a request containing the user's input from general ones (e.g., fetching an HTML). Requests containing user's input are discarded after being received by BitD, while general requests (e.g., fetching an HTML, image) are sent to the Internet as-is.

In step (2), BitD interrupts all requests that the web pages tries to send to the Internet. If a request contains the same information as user's input obtained in step (1), this request is discarded to prevent a BITB Attack.

V. Evaluation

A. Prerequisite for Detection

To evaluate the performance of BitD's detection capability, we collected a total of 64 working BITB PoCs from nine repositories from GitHub, in addition to the TF2 dataset. Fig. 4 shows a few examples of collected BITB PoCs.

Some BITB PoCs were incomplete and did not work. Therefore, the following modifications were made to meet the necessary conditions.

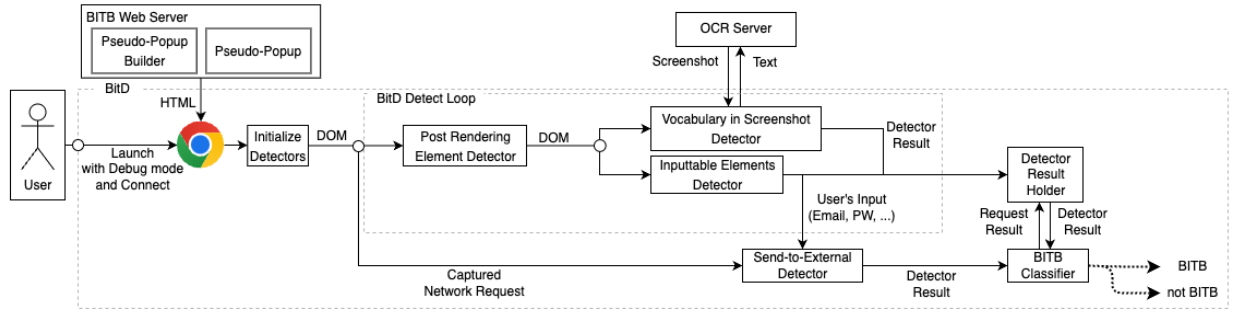
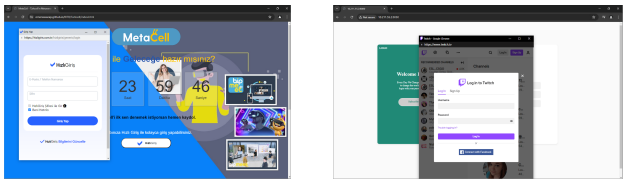


Fig. 3: The system overview of BitD. Data flows are represented by solid lines.



(a) omerwwazap [21] (b) surya-dev-singh [22]

Fig. 4: Example of BITB PoCs.

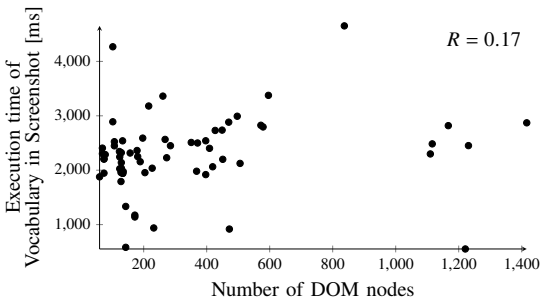


Fig. 5: Relationship between number of nodes in the DOM and execution time of Vocabulary in Screenshot Detector.

1) Modified the value of the `src` attribute in the `iframe` from a dummy URL to the URL of a server hosting a well-crafted fake login page mimicking Google [23].
2) Modified the Pseudo-Popup to be displayed when the user presses a button.
3) Modified the `button` element in the login form to send user's input to the Internet.
4) Changed the language to English.
5) Ensure that elements within an `iframe` do not affects elements outside the `iframe`. For example, pages that perform redirects during login should not redirect to the outside of the `iframe`.

B. Detection Result

TABLE I shows the detection results and each PoC repository contained multiple versions of BITB PoCs with different variations of browsers and OSs. The total number of tested and detectable variations is shown in the Detectable column of the table.

The result of the experiment shows that BitD detected all 64 BITB PoCs and the TF2 dataset. We draw four takeaways from the results:

- BitD could detect BITB Attack pages that mimic the browser windows on both macOS and Windows.
- BitD could detect BITB Attack pages that mimic both dark and light modes of the browsers.
- BitD could detect BITB Attack pages that mimic the UI interface of both Google Chrome and Firefox browsers.
- BitD could detect Pseudo-Popups that mimic the login pages of various web services.

C. Applying Existing AI Models for Detection

To prove that existing AI models do not work well in detecting BITB Attack pages, we applied two existing models (StackModel [31] and PhishIntention [17]) to nine BITB PoCs and the TF2 dataset. The PoCs are extracted from the ones we use in our experiments in Section V-B. We selected one PoC from each repository so that we could verify a variety of OSs, browsers, and services. As a result, PhishIntention could only detect two ([29] and [22]) out of the ten web pages, while StackModel could detect none.

D. Execution Time

We measured its execution time in a typical client environment as part of the experiment (OS: macOS Sonoma 14.4.1, CPU: Apple M1, Browser: Google Chrome 125.0.6422.78 (arm64), Compiler: Deno [32] 1.43.3).

We measured the execution time of each detector for the 64 PoCs and investigated the correlation between the execution time of each detector and the number of DOM nodes. The Initialize BitD Detectors function had an average execution time of 31 ms, and the correlation coefficient was 0.22 ms. The results of the execution time experiment for Vocabulary in Screenshot Detector are shown in Fig. 5 (the R parameter means the correlation coefficient). This time is almost the same as the execution time of the OCR component. The execution time of other detectors are not shown for brevity because they are negligible compared to the above two.

TABLE I: Detection result of BITB PoCs and TF2 dataset.

BITB PoC Repository / TF2 dataset	OS	Browser	Service	Detectable	Mod.
TF2 dataset	Windows	Chrome-LightMode	Steam	✓(1/1)	
Davidc96 / LogOoops [24]	macOS	Chrome-Dark/LightMode	Google	✓(2/2)	
	Windows	Chrome-Dark/LightMode	Google	✓(3/3)	
		Firefox-LightMode	Google		
mrd0x / BITB [25]	macOS	Chrome-Dark/LightMode	Google	✓(2/2)	※1, ※2
	Windows	Chrome-Dark/LightMode	Google	✓(2/2)	※1, ※2
		Chrome-DarkMode	Google (another ver.)	✓(1/1)	※1
lucthienphong1120 / BITB-Phishing [26]	macOS	Chrome-Dark/LightMode	Google	✓(2/2)	※1, ※4
	Windows	Chrome-Dark/LightMode	Google	✓(2/2)	※1, ※4
warren2i / bitb [27]	macOS	Chrome-Dark/LightMode	Reddit	✓(2/2)	※1
	Windows	Chrome-Dark/LightMode	Reddit	✓(3/3)	※1
		Chrome-DarkMode	Reddit (another ver.)		
vikashchand / browser-in-the-browser-attack [28]	Windows	Chrome-DarkMode	Microsoft	✓(1/1)	※1
daniseis4 / browser-in-the-browser [29]	Windows	Chrome-Dark/LightMode	Microsoft	✓(2/2)	※3
deFr0ggy / BITB-Browser-In-The-Browser-Attack [30]	macOS	Chrome-DarkMode	Original	✓(1/1)	※3
omerwwazap / BITB [21]	Windows	Chrome-DarkMode	Turkcell	✓(1/1)	※4
surya-dev-singh / BITB-framwork [22]	Windows	Chrome-DarkMode	Facebook, Instagram, ... and 40 services	✓(40/42) Not Working (2/42)	※5

VI. Discussion

A. Undetectable BITB Attack Patterns

In this section, we consider three different BITB Attack patterns that were not included in our test cases and also the ones that BitD might fail to detect due to the limitations of the proposed method.

Pattern 1: If the Pseudo-Popup appears immediately after a web page is rendered, it cannot be detected as a Post Rendering element. This is because Post Rendering Detector only focuses on changes in the DOM. In practice, login popup windows are created by user activation [33]. Therefore, if a Pseudo-Popup is found immediately after the web page is rendered, it does not mimic the behavior of a real browser.

Pattern 2: In the TF2 dataset, confidential information was transmitted without encoding. However, for certain real-life scenarios, confidential information may sent by the attacker's script in the encoded form. Since the proposed method checks the transmitted content, if encoding is performed on the client side before transmission, our method will not work. We could tackle this issue by applying taint analysis (e.g., [34]) to track user's input.

Pattern 3: In order to detect the Inputtable elements, BitD searches for `input` and `textarea` tags. However, various other methods can be used to realize a text input mechanism on a web page, such as the `contenteditable` attribute. Nevertheless, BitD could also address other input methods with the knowledge of related HTML specifications.

B. BitD's Performance

The bottleneck in the execution time of BitD is the Vocabulary in Screenshot Detector, which takes around 2000-3000 ms. This is sufficiently fast because it can be completed between the time a typical user opens the page and the time the confidential information is finished inputting and about to be sent.

VII. Future Work

First, a large dataset of BITB Attack web pages in the wild must be constructed to improve machine-learning based detection mechanisms. Because our method mainly focuses on the behavior of web pages, a legitimate web page that satisfies all of the behaviors we define can be falsely detected as a BITB Attack. By combining machine learning techniques that investigate the appearances of web pages, this type of false positive could be decreased. In order to construct a dataset, our system could be utilized to crawl the Internet to find more BITB Attack in the wild besides the TF2 campaign.

Second, our system must be embedded into browsers in order for it to be deployed to the real-world. To this end, the reliance of BitD on running Chrome in the debug mode must be removed. A better implementation method would be to implement BitD as a series of patches to an OSS browser such as Chromium.

Third, Login Vocabulary must be more carefully collected to reduce false positives. The current Login Vocabulary contains words that are likely to cause false positives, such as `google` and `help`.

VIII. Conclusion

For this research, we proposed a robust detection method specifically for Browser-in-the-Browser Attacks. To evaluate the general performance of BitD, we conducted a series of experiments on BITB PoCs alongside real-world dataset. The overall experiment results showed that not only does BitD successfully maintain the high accuracy that was reflected by the result of being able to detect all of the suspicious activities hidden inside the experiment cases, but it also has the capabilities to operate quickly.

Therefore, it is confident to say that the proposed method is an effective approach for detecting the BITB Attacks.

References

[1] R. Zieni, L. Massari, and M. C. Calzarossa, "Phishing or not phishing? a survey on the detection of phishing websites," *IEEE Access*, vol. 11, pp. 18499–18519, 2023.

[2] J. S. Tharani and N. A. Arachchilage, "Understanding phishers' strategies of mimicking uniform resource locators to leverage phishing attacks: A machine learning approach," *SECURITY AND PRIVACY*, vol. 3, no. 5, p. e120, 2020.

[3] M. A. Qbeitah and M. Aldwairi, "Dynamic malware analysis of phishing emails," in *2018 9th International Conference on Information and Communication Systems (ICICS)*, pp. 18–24, 2018.

[4] "Phishing activity trends report, 1st quarter 2022." https://docs.apwg.org/reports/apwg_trends_report_q1_2022.pdf. (Accessed on 05/21/2024).

[5] E. Lin, S. Greenberg, E. Trotter, D. Ma, and J. Aycock, "Does domain highlighting help people identify phishing sites?," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, (New York, NY, USA), pp. 2075–2084, Association for Computing Machinery, 2011.

[6] F. Barr-Smith and J. Wright, "Phishing with a darknet: Imitation of onion services," in *2020 APWG Symposium on Electronic Crime Research (eCrime)*, pp. 1–13, 2020.

[7] "Browser in the browser (bitb) attack — mr.d0x." https://mrd0x.com/browser-in-the-browser-phishing-attack/. (Accessed on 05/23/2024).

[8] "Letting off steam — group-ib blog." https://www.group-ib.com/blog/steam/. (Accessed on 05/22/2024).

[9] "Steam website." https://store.steampowered.com/. (Accessed on 05/23/2024).

[10] "Tracking cyber activity in eastern europe." https://blog.google/threat-analysis-group/tracking-cyber-activity-eastern-europe. (Accessed on 01/03/2025).

[11] "Spotting a browser-in-the-browser (bitb) phishing attack : r/steamscams." https://www.reddit.com/r/SteamScams/comments/1drvt34/spotting_a_browserinthebrowser_bitb_phishing. (Accessed on 01/03/2025).

[12] "Search - urlscan.io." https://urlscan.io/search/#domain%3Atf2*.pages.dev. (Accessed on 01/03/2025).

[13] "Malwarecube/enhanced-iframe-protection: A lightweight extension to automatically detect and provide verbose warnings for embedded iframe elements in order to protect against browser-in-the-browser (bitb) attacks.." https://github.com/MalwareCube/enhanced-iframe-protection. (Accessed on 05/23/2024).

[14] K. Alessa, B. Alhetelah, G. Alazman, A. Bader, N. Alhomeed, L. Almubarak, and F. Almulla, "Browser-in-the-browser (bitb) attack: Case study," *Journal of Engineering Research and Sciences*, vol. 3, pp. 14–22, 05 2024.

[15] A. Tiwari, V. Chaturvedi, R. K. Gupta, and P. Upadhyay, "Phishspy – a phishing detection tool and defensive approaches," in *2022 International Conference on Industry 4.0 Technology (I4Tech)*, pp. 1–6, 2022.

[16] S. Asiri, Y. Xiao, S. Alzahrani, and T. Li, "Phishingrtds: A real-time detection system for phishing attacks using a deep learning model," *Computers & Security*, vol. 141, p. 103843, 2024.

[17] R. Liu, Y. Lin, X. Yang, S. H. Ng, D. M. Divakaran, and J. S. Dong, "Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach," in *31st USENIX Security Symposium (USENIX Security 22)*, (Boston, MA), pp. 1633–1650, USENIX Association, Aug. 2022.

[18] "Chrome devtools protocol." https://chromedevtools.github.io/devtools-protocol/. (Accessed on 05/28/2024).

[19] "tesseract-ocr/tesseract: Tesseract open source ocr engine (main repository)." https://github.com/tesseract-ocr/tesseract. (Accessed on 05/28/2024).

[20] "Domain rankings — cloudflare radar." https://radar.cloudflare.com/domains. (Accessed on 05/31/2024).

[21] "omerwwazap/bitb: Browser in the browser (bitb) attack - turkcell fastlogin/hızlıgiriş." https://github.com/omerwwazap/BITB. (Accessed on 05/30/2024).

[22] "surya-dev-singh/bitb-framwork." https://github.com/surya-dev-singh/BITB-framwork. (Accessed on 06/01/2024).

[23] "lucthienphong1120/google-login: Demo google login form." https://github.com/lucthienphong1120/google-login. (Accessed on 06/01/2024).

[24] "Davidc96/logooops: Tool to perform browser-in-the-browser attacks." https://github.com/Davidc96/LogOoops. (Accessed on 06/13/2024).

[25] "mrd0x/bitb: Browser in the browser (bitb) templates." https://github.com/mrd0x/BITB. (Accessed on 05/30/2024).

[26] "lucthienphong1120/bitb-phishing: Browser in the browser (bitb) attack is a sophisticated phishing and hard to detect.." https://github.com/lucthienphong1120/BITB-Phishing. (Accessed on 05/30/2024).

[27] "warren2i/bitb: Command line generation browser in the browser exploit framework." https://github.com/warren2i/bitb. (Accessed on 05/30/2024).

[28] "vikashchand/browser-in-the-browser-attack." https://github.com/vikashchand/browser-in-the-browser-attack. (Accessed on 05/30/2024).

[29] "daniseis4/browser-in-the-browser." https://github.com/daniseis4/browser-in-the-browser. (Accessed on 05/30/2024).

[30] "defr0ggy/bitb-browser-in-the-browser-attack: A recent browser in the browser attack working example!." https://github.com/deFr0ggy/BITB-Browser-In-The-Browser-Attack. (Accessed on 05/30/2024).

[31] Y. Li, Z. Yang, X. Chen, H. Yuan, and W. Liu, "A stacking model using url and html features for phishing webpage detection," *Future Generation Computer Systems*, vol. 94, pp. 27–39, 2019.

[32] "Deno, the next-generation javascript runtime." https://deno.com/. (Accessed on 05/29/2024).

[33] "Html standard." https://html.spec.whatwg.org/multipage/interaction.html#tracking-user-activation. (Accessed on 01/15/2025).

[34] "perlsec - perl security - perldoc browser." https://perldoc.perl.org/perlsec. (Accessed on 01/03/2025).