PP3D: An In-Browser Vision-Based Defense Against Web Behavior Manipulation Attacks

Abstract—**Web-based behavior-manipulation attacks (BMAs)—such as scareware, fake software downloads, tech support scams, etc.—are a class of social engineering (SE) attacks that exploit human decision-making vulnerabilities. These attacks remain under-studied compared to other attacks such as information harvesting attacks (e.g., phishing) or malware infections. Prior technical work has primarily focused on measuring BMAs, offering little in the way of generic defenses.**

To address this gap, we introduce *Pixel Patrol 3D* (*PP3D*), the first end-to-end browser framework for discovering, detecting, and defending against behavior-manipulating SE attacks in real time. *PP3D* consists of a visual detection model implemented within a browser extension, which deploys the model client-side to protect users across desktop and mobile devices while preserving privacy.

Our evaluation shows that *PP3D* can achieve above 99% detection rate at 1% false positives, while maintaining good latency and overhead performance across devices. Even when faced with new BMA samples collected months after training the detection model, our defense system can still achieve above 97% detection rate at 1% false positives. These results demonstrate that our framework offers a practical, effective, and generalizable defense against a broad and evolving class of web behavior-manipulation attacks.

1. Introduction

Social engineering (SE) encompasses a broad spectrum of attacks aimed at deceiving users into performing actions or divulging sensitive information that may have important negative consequences for the security and privacy of the users themselves or their organizations [1]. These threats exploit weaknesses in humans' decision-making processes by using tactics such as pretext, baiting, impersonation, etc. [2].

Within the broad spectrum of SE attacks, we focus on attacks that include a significant web-based component. In this context, we can distinguish between two classes of attacks: *information harvesting* and *behavior manipulation* attacks. Information harvesting attacks aim to extract sensitive information from users, such as login credentials, financial data, or personal details. Web-based phishing websites that steal login credentials [3], [4] and smishing messages that attempt to get sensitive data such as Social Security Numbers are some examples for this class of SE attacks. On the other hand, behavior manipulation attacks seek to influence users into pursuing insecure actions, rather than giving up information, often leading to malware infections or immediate financial harm such as money transfer to criminals. Examples of behavior manipulation attacks include scareware, fake software downloads, tech support scams, fake sweepstakes, etc. [5]–[8].

Unfortunately, while information harvesting attacks (IHAs) have received significant attention from researchers [9]–[16], to the best of our knowledge no comprehensive defense framework against web-based behavior-manipulation attacks (BMAs) has been proposed or evaluated. Yet, BMAs remain a formidable SE attack class in financial terms. For instance, US consumers reported losing more than a billions dollars to tech support scams alone in 2024 [17]. Prior works have shown that a large number of these scams originate from fake tech support websites which display glaring warning messages to coax victims into making telephone calls to scam call centers [6], [8].

Previous studies on web-based BMAs are either limited to discovering and measuring attack campaigns [18], [19] or to studying only specific subclasses of attacks [5]–[8], [20], with limited or no concrete defense. Furthermore, existing practical defenses against malicious web pages mostly rely on reactive approaches that do not focus on the fundamental traits of BMAs and tend to lag behind new threats, thus leaving many users exposed to the latest attack iterations. For instance, URL blocklist services such as Google Safe Browsing (GSB) [21] focus on *where* malicious content is hosted, rather than detecting and blocking the malicious content itself. Therefore, attackers are often able to evade blocking by simply relocating their attacks to a different hosting location, as reported by previous measurements [18].

A recent work by Yang et al. [22], named TRIDENT, addresses the problem of detecting social engineering ads, which can in turn lead to behavior-manipulation attacks. However, TRIDENT narrowly focuses on detecting JavaScript code served by low-reputation ad networks that often use SE techniques, such as transparent overlays, to hijack users' clicks and further redirect them to a potentially malicious page. While TRIDENT is able to detect SE online ads, it can detect some BMA webpages only indirectly, because low-tier ad network code may or may not redirect to SE attacks. This can thus also generate false positives, namely legitimate (non-SE) web pages that may be blocked because a redirection originated from a low-tier online ad. Additionally, attacks that are not reached via malicious ads would be completely missed by TRIDENT, thus resulting in false negatives. Therefore, there is a need for a more generic defense that is able to directly detect web-based behavior-manipulation attack pages based on their content, rather than being limited to detecting malicious online ads that may in some cases redirect users to such attacks.

Problem Definition. Based on the taxonomy proposed by Zaoui *et al.* [23], we focus on *remote, psychologically triggered, web-based* social engineering attacks. Within this slice, we further distinguish between two classes of attacks based on their end-goal:

Information-Harvesting Attacks (IHAs): In IHAs, the

adversary solicits secrets, such as credentials or sensitive personal identifiers. These attacks include traditional phishing of login credentials, banking accounts, tax-related information, etc., as well as modern crypto-currency phishing attacks.

Behaviour-Manipulation Attacks (BMAs): Unlike IHAs, in BMAs the adversary's primary goal is to induce an unsafe action, rather than directly harvesting information. For instance, downloading malware [5], granting browser notification permissions to unwanted websites [19], calling a fraudulent tech support line [6], paying an advance fee for a bogus lottery or gift or for a fake service registration.

In this paper, our goal is to propose a *novel framework for in-browser defense against web-based behavior-manipulation attacks (BMAs) using visual webpage content analysis*. Specifically, we focus on detecting the following sub-categories of BMAs, which have been defined in past measurement studies [18]: *fake software downloads, scareware, tech-support scams, notification stealing, fake lotteries/sweepstakes*, and *service registration scams*. These BMAs have the following properties:

1) *Glaring visual deception.* BMA web pages rely on persuasive imagery to drive behaviour. For instance, a notification-stealing site (Table 6 in Appendix) renders a fake video player behind a genuine permission dialog, claiming playback requires clicking *Allow*, TSS pages embed counterfeit system windows to coerce phone calls [6], etc.

2) *BMAs are orchestrated via attack campaigns.* BMA pages are often part of a larger campaign, where multiple attack pages are hosted under different domains. For example, a scareware campaign may include numerous scareware web pages that all use similar visual deception techniques (e.g., claiming that the machine is infected with multiple viruses) but are hosted on different websites to evade traditional URL-based defenses.

3) *Different from IHA.* Unlike information harvesting attacks, such as phishing pages, BMAs do not require mimicking a specific site/brand or harvesting credentials; they instead invent a believable scenario with the goal of manipulating users' actions, as shown in Table 6 (in Appendix). Hence, phishing detectors, including recent systems that check for visual similarity to known benign pages [11] or for input boxes [13]–[16] are not suitable for detecting the behavior-manipulation attacks we focus on in this paper.

4) *Need for more effective BMA defenses.* Traditional defenses, such as URL block-lists [21], are reactive and relocation-sensitive; measurements show many BMA pages remain active for days, before being blocked [18]. Also, previous initial work on BMA detection only attempt to defend against them indirectly, by blocking malicious online ads [22]. Therefore, we need a content-based defense that can detect BMA pages directly, based on their attack content and regardless of their hosting location.

Motivated by these fundamental traits of BMA pages and the lack of existing solutions that address them comprehensively, in this paper we propose *Pixel Patrol 3D*: a novel framework for multimodal, in-browser detection of BMA pages. *Pixel Patrol 3D* consists of three components: a discovery module that identifies BMA campaigns, a multimodal detection model that analyzes the content of web pages to extract attack-related visual and text (via OCR) cues, and a browser extension that puts the detection model into practice, allowing it to defend web users from BMAs in real-time within the browser. Our experimental results show that *Pixel Patrol 3D* achieves a high detection rates even on previously unseen BMA campaigns (Table 4). Furthermore, our browser extension is able to detect and block BMA pages in real-time with limited overhead, providing users with a first practical content-based defense against these attacks.

Contributions. We make the following main contributions:

- ***Framework***. We introduce *Pixel Patrol 3D* (*PP3D*), the first end-to-end browser framework that discovers, detects, and defends against web-based *behaviour manipulatating* social-engineering attacks. *PP3D* can achieve a high detection rate of 99% at a 1% false positive rate, even on previously unseen BMA campaigns. When evaluated on new BMA samples collected months after our model was trained, the system still maintains a detection rate above 97% at 1% false positives. Furthermore, our browser extension is able to detect and block BMA pages with limited overhead by exploiting their visual cues, thus providing users with a first practical in-browser defense against these insidious attacks.
- ***Dataset***. We use a specialized crawler to harvest a large corpus of 7,149 recent in-the-wild BMA pages belonging to 84 different attack campaigns. By combining our new dataset with historic samples of BMA campaigns collected by previous work, we build the largest and most comprehensive labeled dataset for evaluating BMA detection systems, which we will make available to the research community to foster future research in this area.
- ***Model***. We design a novel multimodal detection model that fuses vision and text features extracted from web pages to accurately detect BMAs. Our model accepts screenshots of arbitrary size in input, to support deployment on a diverse set of devices, and can detect BMAs while generalizing across different screen resolutions and attack campaigns. We will open-source the full *PP3D* code base, including our pretrained detection models and evaluation scripts.
- ***Deployment***. To provide a practical detection solution, we deploy our detection model by implementing a lightweight browser extension for desktop, tablet, and mobile devices using the ONNX Web Runtime [24] and Tesseract.js [25]. The extension performs all inference locally to the user's browser, thus maintaining the privacy of users' browsing history. The complete extension codebase will also be open-sourced to facilitate future research.

2. PP3D Framework

Our approach towards building a BMA detection system is informed by the following key observations:

BMAs have a very significant visual component: BMA pages exploit victims' decision-making processes via a number of deception and persuasion tactics. To attract users' attention and persuade them of the veracity of an invented scenario (e.g., the user's device is compromised, a software update is needed to watch a video, etc.), web-based BMA attacks make use of strong visual cues, including fake dialog boxes, flashy messages, abused brands, etc., as shown by the examples in Table 6 and Figure 2. Our classification approach aims to detect visual cues in such attacks.

BMAs include text content that is often obfuscated: In addition to visual cues, BMA pages also include attack messages that are often "obfuscated" via various techniques, such as text-to-image conversion, font remapping, or canvas rendering. These techniques aim to evade traditional DOM-based detectors that rely on the HTML structure of the page. At the same time, attack-related text in BMAs (e.g., "Your machine is infected," "Call Micosoft Tech Support," "Click Allow to watch the video," and similar messages) is critical for attack success. To capture these cues, we leverage Optical Character Recognition (OCR) to extract the actual user-visible content from the rendered page. This allows us to recover attack-related text and detect BMA pages even when they employ HTML obfuscation.

BMAs are typically orchestrated via campaigns: To scale the attacks and facilitate their distribution to many potential victims, BMAs are orchestrated via campaigns. Specifically, a BMA campaign aims to "advertise" visually and semantically similar/same BMA attack content on multiple different websites, as noted in [18]. Thus, it may be sufficient to observe and learn from few examples of a BMA campaign to allow us to detect future instances of the same or similar BMA attacks on different websites.

Based on the above observations, our defense approach relies on (i) *continuously collecting examples of recent in-the-wild BMAs* from a variety of campaigns, (ii) *training a detection model* that can identify new BMA variants based on visual and textual cues, and (iii) *in-browser deployment of this detection model for real-time defense*. To this end, we design our *PP3D* framework, shown in Figure 1, with three main components: *Pixel Patrol Discover* (PP_{dis}), *Pixel Patrol Detect* (PP_{det}), and *Pixel Patrol Defend* (PP_{def}). In this paper, we focus most of our research efforts on PP_{det}, and build an early prototype of PP_{dis} and PP_{def} to enable the collection of a large BMA dataset and to demonstrate the viability of PP_{det} and of the entire *PP3D* defense framework.

2.1. Pixel Patrol Discover

To automatically identify web-based BMAs, we first need to collect fresh examples of web pages associated with a variety of attack campaigns. To this end, we extend the crawler architecture proposed by Vadrevu et al. [18], expanding the number of emulated devices and screen resolutions from 5 to 30, and logging the full interaction path taken to reach each potential BMA page. Additionally, we conducted a systematic labeling process using three expert human annotators to create a high-quality dataset of BMA examples. This dataset, which will be released to the research community, forms the foundation for training PP_{det}. Details on PP_{dis} and the labeling process are provided in Appendix B.

2.2. Pixel Patrol Detect

PP_{det}, the detection module within the *PP3D* framework, constitutes the core technical contribution of this work. This section details its design, rationale, and implementation.

2.2.1. Design Principles

To ensure broad applicability and robustness, PP_{det} is designed to meet several key requirements.

Resolution Agnosticism. Webpages can be rendered on devices with vastly different screen sizes and aspect ratios, ranging from small mobile phones to ultra-wide desktop monitors. Traditional image classifiers typically assume fixed input dimensions, limiting their ability to generalize across diverse environments. In contrast, PP_{det} must operate on screenshots of arbitrary resolution while maintaining high accuracy across a wide spectrum of BMA campaigns. This challenge is further compounded by our goal of deploying the model as an on-device browser extension, necessitating not only strong detection performance but also low-latency operation on resource-constrained mobile hardware.

Dual-Modality Reasoning. Many BMA webpages share strong semantic similarities, even when their visual designs differ. To capture both superficial and latent patterns, PP_{det} processes both visual and textual information. By combining visual features from webpage screenshots with OCR-extracted text, the model is able to recognize attacks based on both visual appearance and semantic similarity, thus improving generalization against diverse BMA variants.

Resilience to HTML Obfuscation. Sophisticated attackers frequently obfuscate HTML content to bypass traditional DOM-based detection methods. A common tactic is embedding key phrases (e.g., "Click here to claim your prize") as images, rendering them invisible to systems that extract only text from the HTML. Beyond this, adversaries often use custom web fonts to remap visible characters to unrelated Unicode code points; although the user sees familiar phrases like "Allow" or "Download," the underlying HTML contains meaningless sequences that evade keyword-based detectors. Another prevalent technique involves canvas-based rendering, where critical text is drawn dynamically using JavaScript and HTML5 `<canvas>` or SVG elements, completely bypassing the DOM. These and similar strategies highlight the inadequacy of DOM inspection alone and underscore the need for OCR to recover the actual, user-visible semantics of a page. While OCR is more computationally demanding than HTML parsing, it remains essential for robust detection in adversarial settings.

2.2.2. Implementation

PP_{det} uses a dual-branch neural network (implemented using PyTorch) that processes visual and textual features in parallel before fusing them for final classification (see Figure 1). Specifically:

- The **visual branch** employs a MobileNetV3-Small [26] backbone pre-trained on ImageNet, with the classification head removed and replaced by an identity layer as a visual feature extractor. The output feature vector (576 dimensions) is passed through a dropout layer to reduce overfitting.
- The **text branch** uses a compact transformer encoder based on BERT Mini [27] (specifically, `prajjwal1/bert-mini`). The pooled output from the [CLS] token (with dimensionality 312) is processed through a dropout layer followed by a linear projection to a fixed 128-dimensional embedding.

The 576-dimensional visual and 128-dimensional textual embeddings are concatenated to form a unified 704-dimensional feature vector. This joint representation is passed through a feedforward classification head consisting of a 256-unit fully connected layer with ReLU activation, followed by a final linear layer that

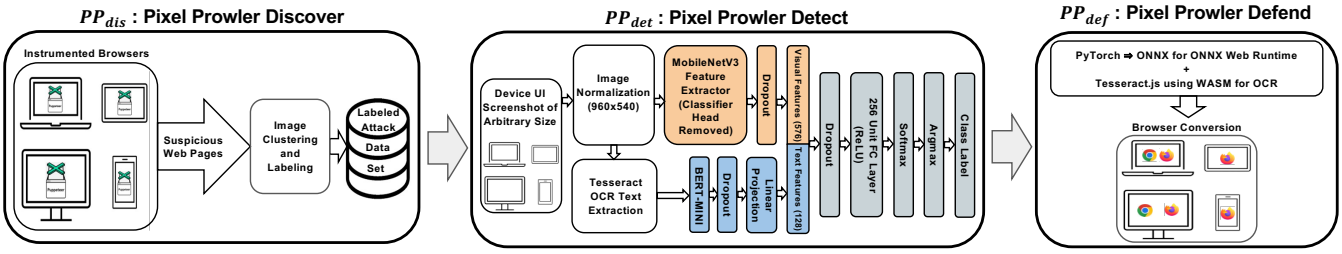


Figure 1: Overview of the *Pixel Patrol 3D* Framework. The PP_{det} module includes both preprocessing steps (e.g., image normalization and OCR) and the core detection model. The colored blocks within PP_{det} represent the core model components used for feature extraction and classification.

outputs logits for binary classification. Dropout regularization is applied before the classification head to mitigate overfitting. During inference, logits are passed through a softmax layer to produce class probabilities, with the predicted label selected via `argmax`. The overall model architecture is illustrated in the PP_{det} portion of Figure 1. This design allows PP_{det} to reason jointly over both visual and textual modalities while remaining efficient enough for real-time deployment on both desktop and mobile platforms.

To normalize screenshots of arbitrary screen resolutions (to accommodate different device screen formats), we first project each image onto a 1920×1080 frame. For oversized images—those where either dimension exceeds 1920 pixels in width or 1080 pixels in height—we apply isotropic scaling such that the larger dimension fits within the corresponding bound (1920 or 1080), preserving the aspect ratio. For undersized images—those that already fit within 1920×1080—no scaling is applied. In both cases, the scaled image is centered on a 1920×1080 canvas, and any unused margins are padded with zero-valued pixels (RGB [0, 0, 0]). The resulting canvas is then downsampled by a factor of 0.5, empirically chosen to balance classification accuracy, OCR effectiveness, and inference speed, resulting in a final input size of 960×540. A procedural outline is presented in Appendix I. This normalization process ensures consistent input dimensions across devices and supports efficient batch inference on the GPU.

To extract text from webpage screenshots, we use the Tesseract OCR engine [25]. Tesseract was selected for its strong tradeoff between recognition accuracy and performance, particularly in low-resource environments.

2.3. Pixel Patrol Defense

To demonstrate the practicality of the *PP3D* framework, we develop a proof-of-concept extension for the browser that allows for detecting BMA in real time.

Initialization. When the extension initializes—either at browser launch or when it is re-enabled—it first creates a single off-screen document, which in turn spawns a pool of dedicated Web Workers. One worker hosts the image-text classifier introduced in Section 2.2.2, which we implemented by converting the pre-trained model using ONNX [24]. The converted modules are the colored modules shown in the PP_{det} portion of Figure 1. Four additional workers run Tesseract, leveraging its native WebAssembly (WASM) support. This allows all OCR tasks to execute efficiently on the browser, ensuring user privacy (all inference runs locally to the browser with no data leaked to any external API) and delivering consistent performance across both desktop and mobile browsers. All workers are instantiated only once per session, so that model weights and tokenizer tables reside in memory, avoiding repeated loading delays and garbage-collection churn. With the worker pool active, the background extension script starts a non-blocking fixed 5 second page scan timer that drives every subsequent classification cycle.

Decision Pipeline. For each page scan, the extension applies a tiered filtering strategy that minimizes expensive inference. If the current domain is less or equal to rank 100K in the Tranco Top 1M list [28], the page is immediately labelled *benign*, as such high-ranking domains are rarely associated with malicious behavior. This threshold aims to strike a balance between efficiency and coverage, allowing the system to focus resources on less popular and more likely malicious domains. Otherwise, the extension captures a screenshot, computes an 8x8 block mean value-based perceptual hash (BMVBHash) [29], and measures its Hamming distance [30] to the hash from the previous scan.

This yields four mutually exclusive cases:

Case 1: Domain in whitelist → verdict of *benign*.
Case 2: No previous phash → run inference.
Case 3: Hamming distance ≥ 5 → run inference.
Case 4: Hamming distance < 5 → reuse last verdict.

Only Cases 2 and 3 invoke the heavy model, keeping the average per-page overhead low. The hamming distance threshold of 5 was empirically chosen to be conservative, so that an inference result from a previous scan is reused only if the two images are visually very similar, thus justifying the reuse of the previous verdict.

Inference. Whenever a screenshot needs to be analyzed (i.e., in Cases 2 or 3), the scanning loop is temporarily paused. The extension captures a screenshot of the current browser tab and normalizes the image as described in Section 2.2.2. This normalized image is then transferred—using zero-copy techniques—to the off-screen document for processing. Within the off-screen document, two types of Web Workers operate sequentially: *(i)* The OCR workers divide the image into four horizontal slices (1 column × 4 rows), a layout empirically shown to balance speed and accuracy effectively. Each slice is sent to a separate Tesseract Web Worker, and the resulting plain text outputs are concatenated from top to bottom. *(ii)* The ONNX worker then receives the resized image along with the extracted OCR text and feeds both

into the ONNX-converted model. The model performs inference on the dual inputs and returns a classification label—either *benign* or *malicious*—which is used to update local storage accordingly.

User Interaction. If the classifier returns *malicious*, the background script opens a notification window presenting a warning that the page is likely malicious, shows the page screenshot, and displays three action choices: *Return to Safety* (redirects the tab to a trusted site), *Ignore Warning*, and *Not Malicious* (records a false-positive override). Scanning pauses while the dialog is active and resumes as soon as the user selects a choice and the window closes.

Logging and Configuration. The extension also provides a popup UI that allows users to toggle on/off the logging of screenshots and performance metrics, as well as to view the latest classification results. Execution times, domain names, webpage screenshots, and classification outcomes are buffered in memory and flushed to a timestamped text file on the user's device every two minutes.

3. Experimental Setup

We now discuss the setup for evaluating *PP3D* framework.

3.1. Data Collection using Pixel Patrol Discover

To collect a diverse set of web pages for evaluating PP_{det}, we use PP_{dis} to crawl (i) seed URLs that use low-tier ad networks that are likely to lead to attack web pages and (ii) an unbiased set of benign URLs, using the setup explained in Appendix C.

Benign Webpage Collection. To account for variability in the benign webpage designs and build a representative dataset, we combined seed URLs from two distinct sources: the top 5,000 domains listed by Tranco [28], representing popular sites, and 10,000 URLs randomly selected from Web Extraction (WET) files spanning one month of the Common Crawl dataset [31], capturing less popular and potentially more diverse sites. Using the crawler to emulate multiple devices and resolutions, we obtained 396,255 distinct screenshots from the popular websites and 386,180 distinct screenshots from the Common Crawl URLs. While some webpages sampled from the Common Crawl dataset may not be explicitly benign, given the random sampling, we consider the likelihood of encountering BMA sites to be minimal, with any resulting noise likely having only minor impacts, such as slightly increased false positives during testing. Further discussion regarding ethical considerations and the negligible impact of our large-scale crawling on benign websites is discussed in Appendix D.

BMA Webpage Collection. To collect seed URLs that may provide a high-yield of BMAs, we follow an approach similar to previous studies [18], [19] and collect URLs of websites that are known to host ads from low-tier ad networks, as they have been observed to often lead to BMA pages. Overall, we collected 24,979 seed URLs associated with 10 low-tier ad networks (listed in Table 7(a), in the Appendix). In addition to this initial collection, we conducted a second BMA gathering effort several months later. This allowed us to evaluate the effectiveness and temporal robustness of our detection system across different collection periods.

Primary BMA Collection: In this initial collection, we began with *28,923* unfiltered BMA screenshots provided by the authors of [18] and expanded the dataset by crawling *24,979* seed URLs. This crawl resulted in *650,255* additional unique screenshots from 55,539 distinct webpages, leading to a total of *679,178* images. We then removed duplicates and applied clustering based on screen resolution and perceptual hashing—following techniques similar to those in [18], [22]—which yielded *8,107* images grouped into *318* clusters.

To identify BMA campaigns, we carried out a systematic labeling process involving three individuals familiar with BMA techniques. This process included multiple rounds of collaborative review followed by independent labeling of the *318* clusters. After independent labeling, we computed the inter-rate reliability score using Krippendorff's alpha score [32] among the labelers and obtained $\alpha = 0.82$, which indicates a high level of agreement [33]. As a result, we identified *245* clusters containing *7,012* BMA screenshots across 30 screen resolutions. We then manually grouped these 245 clusters into *74* meta-clusters based on visual and behavioral similarities in attack type, representing distinct BMA campaigns. Table 1 summarizes the number of campaigns and associated screen resolutions for each attack category in the primary dataset.

TABLE 1: Summary of BMA categories in the initial collection, including the number of campaigns and distinct screen resolutions observed in the collected webpage screenshots.

BMA Category	# Screenshots	# Resolutions	# Campaigns
Fake Software Download	4700	29	29
Notification Stealing	1130	27	7
Service Sign-up Scam	758	24	20
Scareware	213	8	9
Fake Lottery/Sweepstakes	194	4	6
Technical Support Scam	17	2	3
Total (distinct)	**7012**	**30**	**74**

Secondary BMA Collection: While the primary BMA webpage collection is used for most of our experiments, we collected a secondary set of BMA pages after several months to assess how well PP_{det} performs against newly observed BMA attack campaigns. By crawling 6,120 seed URLs, we captured 305,411 unique screenshots that led to 137 unique BMA webpage screenshots across 10 new distinct never-before-seen BMA campaigns.

3.2. Training Pixel Patrol Detect

To perform all PP_{det} experiments, we used a dedicated Dell PowerEdge C4140 with 512GB memory and 4x 32GB NVIDIA V100 GPUs, with CUDA 12.2 and PyTorch v2.6.0+cu124.

Dataset Generation. From the collection runs described above, we construct a master training dataset of *119,536* images comprising both benign and BMA samples.

The benign portion includes *100,000* unaugmented screenshots randomly sampled from our broader benign collection. We ensured proportional representation across all 30 screen resolutions (see Appendix Table 7(b)) and maintained a roughly even split between screenshots obtained from the Tranco Top 5K crawl and those from the Common Crawl.

The BMA portion is based on *6,512* unaugmented screenshot-text pairs from our primary BMA collection (with the remaining

labeled BMA samples reserved for testing). For each screenshot, we used Tesseract OCR [25] to extract visible on-page text, saving the result as a plain-text file to establish a one-to-one mapping between each image and its OCR output. We then applied both image and text augmentation to this set, resulting in a final BMA training set of *19,536* samples.

We intentionally preserve class imbalance in the training dataset, with a significantly larger number of benign examples. This decision reflects the real-world distribution of web content, where benign pages vastly outnumber malicious ones. Training under a class distribution that reflects the imbalance found in deployment conditions helps the model calibrate its decision boundary more effectively and reduces overfitting to rare malicious patterns. Prior work in anomaly detection and imbalanced learning [34], [35] supports this approach, showing that maintaining class imbalance, when combined with appropriate augmentation or feature learning techniques, can improve generalization and reduce false positives in deployment scenarios.

Test datasets are tailored for each experiment, depending on the evaluated scenario, for example, by excluding specific campaigns or resolutions.

Data Augmentation. To enhance the model's robustness, we applied data augmentation techniques to all training images, targeting both their visual and textual components. For image augmentation, two transformations were randomly selected from a predefined set, which included color inversion, grayscale conversion, random margin cropping, extreme hue shifts, and adjustments to brightness, contrast, saturation, and solarization. Each transformation was applied with randomized parameters to minimize duplication. All selected augmentations were designed to maintain text legibility and the overall visual structure of the webpage.

For text augmentation, we utilized Groq's LLaMA-3.3-70B-Versatile model via their cloud API to perform synonym replacement. This method increases lexical variety while preserving sentence structure and semantic accuracy. Complete prompting details and API settings are provided in Appendix F. As with the image transformations, we ensured that textual modifications did not impair readability or alter the original meaning of the webpage content.

Hyperparameter Tuning. Prior to final experimentation with the PP_{det} model, we performed extensive hyperparameter tuning using 10,000 randomly sampled benign train-test pairs and 2,000 randomly sampled BMA train-test pairs from the master training dataset, supplemented by a validation set comprising 500 benign and 500 BMA test images.

We began the tuning process using the Adam optimizer with an initial learning rate of 1×10^{-3}, and explored both higher and lower learning rates. We also evaluated various learning rate schedulers, including cosine decay, one-cycle, and step decay. To assess the impact of regularization, we varied the degree of backbone freezing—evaluating configurations where MobileNetV3 and BERT-Mini were either fully frozen, partially unfrozen, or fully trainable. We additionally tuned dropout rates on the visual branch, textual branch, and the fusion multilayer perceptron.

Other hyperparameters explored included multiple values for weight decay, batch size, and three loss formulations: standard cross-entropy, class-weighted cross-entropy, and focal loss. We also tested different pre-processing configurations, including image downscaling factors and maximum token lengths for the textual input.

The best-performing configuration left all layers fully trainable and used a fixed learning rate of 2×10^{-6} without any scheduler. It applied class-weighted cross-entropy loss, with weights proportional to the observed class imbalance, and a weight decay of 5×10^{-4}. Dropout was set to 0.3 for both the visual and textual branches, and 0.6 for the fusion layer. Optimal results were achieved using an image scaling factor of 0.5, a maximum token length of 512, and a batch size of 64.

Although exact composition of the training and test datasets varied across experiments (see Section 4), all models were trained using this optimized configuration for consistency.

3.3. Deploying Pixel Patrol Defend

To quantify both user-visible latency and device-side resource consumption, we evaluated PP_{def} on representative desktop, tablet, and mobile hardware. Achieving robust results required two complementary steps: (i) choosing test devices that mirror what everyday users own, and (ii) crafting a repeatable measurement protocol. We outline both below.

Device Selection. To evaluate the performance of our proof-of-concept PP_{def} browser extension across realistic user environments, we selected representative devices spanning mobile, tablet, and desktop platforms. The mobile and tablet categories are covered by widely available Android devices that balance affordability and performance, reflecting mainstream user hardware. For desktops, we chose a typical enterprise-grade laptop alongside a high-end prosumer device to assess scalability across both resource-constrained and high-performance environments. Detailed device specifications are provided in Table 2.

TABLE 2: Device specifications used in performance evaluation.

Device	Processor	Cores	RAM	OS
Samsung Galaxy A55	Exynos 1480	8	8 GB	Android 14
Samsung Galaxy Tab S9 FE	Exynos 1380	8	8 GB	Android 14
Dell Latitude 5420	Intel Core i7-1165G7	4	16 GB	Ubuntu 22.04
MacBook Pro	Apple M4 Max	16	128 GB	macOS Sequoia

Testing Strategy. As detailed in Subsection 2.3, the PP_{def} extension supports four execution paths, which give rise to three distinct performance profiles. Our testing strategy is designed to evaluate these three profiles, as well as evaluate the extension initialization step. Additionally, although we developed versions of the extension for both Chrome and Firefox, only the Firefox build is usable across desktop, tablet, and mobile platforms. Accordingly, all procedures described here refer to the Firefox variant.

To systematically evaluate the latency and system resource impact of each profile, we designed the following testing strategy. We selected 15 representative webpages: 5 whitelisted domains (ranked within the top 100,000 of the Tranco Top 1M list) and 10 non-whitelisted domains. The larger proportion of non-whitelisted domains reflects our focus on measuring the extension's most resource-intensive behavior—phash comparisons and model inference.

For each device under test, we shuffled the domain list and browsed each domain for two minutes using either Firefox (on desktop) or Firefox Nightly (on tablet and mobile). Throughout the experiment, we relied on our extension's built-in logging functionality to collect timestamps and latency data for each major page processing stage. Note that there is no added latency due to the logging functionality.

In parallel, we run a custom monitoring script to record system-level CPU and memory usage attributed to the Firefox process. On desktop and tablet environments, we used a Python script that sampled the process list at 500 ms intervals to extract CPU usage as a percentage of total logical cores (e.g., 800% = full use of 8 cores) and RAM usage (in MB) using `ps` and `/proc/meminfo`. On mobile devices, we used an analogous script that executed `adb shell top` at regular intervals to retrieve live CPU and memory usage of the Firefox Nightly app (Fenix). In both cases, the monitoring script logged timestamped data and usage as a percentage of total available CPU and RAM on the device.

To establish a performance baseline, we repeated each experiment with the extension uninstalled. This enabled direct comparison of resource utilization with and without PP_{def} active.

4. Experimental Results

Our evaluation aims to answer 5 major research questions (RQs) and provide a performance evaluation of our defense.

4.1. Pixel Patrol Detect Results

RQ1: Can PP_{det} accurately identify *new* instances of BMAs belonging to previously observed campaigns?

To answer this question, we evaluate the detection capabilities of our PP_{det} model. To this end, we randomly select 500 benign examples and 500 BMAs, which are completely separate from the training dataset. Notice that the related web page screenshots were selected across randomly chosen screen sizes and BMA campaigns, and that it is possible that a test BMA sample may be part (as a newly observed attack instance) of a BMA campaign previously seen during training. For the training datasets, we use the master dataset discussed in Section 3.2. In this simple test setting, PP_{det} achieves an AUROC score of 1.0 and a detection rate above 99% at 1% false positives.

4.2. Generalization to New Screen Sizes

RQ2: Can PP_{det} accurately identify instances of BMAs captured on a *new* screen size never seen during training?

To answer RQ2, we proceeded as follows. First, we selected 9 different screen resolutions from our master dataset discussed in Section 3.2. Among these 9 resolutions, 5 are landscape resolutions while the other 4 are portrait resolutions related to mobile devices. Let r_i indicate one of these 9 resolutions. Then, we formed 9 different training datasets, $\{T_1,...,T_9\}$, where dataset T_i was formed by selecting images from all resolutions except for r_i. The remaining images with resolution r_i were set aside as test dataset S_i. We then performed 9 training/test experiment by training on T_i and testing on S_i (i.e., images with the excluded resolution r_i).

Notice that, because PP_{dis} visits the same URLs using different browser viewport sizes, the same BMA campaigns can typically be observed under different resolutions. Therefore, T_i and S_i can include BMAs from the same campaigns, but the training and test screenshot images are taken on different device screen sizes. This allows us to focus on the effect of web page screenshots taken on never-before-seen screen sizes, rather than the effects of previously unseen BMA campaigns. Also, to obtain a roughly balanced test dataset, where a particular BMA campaign does not dominate the others, given resolution r_i we selected at most 10 images from each BMA campaign represented under that resolution and 500 random images from benign pages in the same resolution.

The results of this experiment are reported in Table 3. "Global" refers to the overall results computed by first combining the classification scores obtained during the 9 training/test experiments into a single scores set. It is notable that PP_{det} can generalize very well to previously unseen resolutions, with an AUROC score at or above 0.999 and detection rates at or above 99% at 1% false positives.

TABLE 3: Results for generalization to new screen sizes.

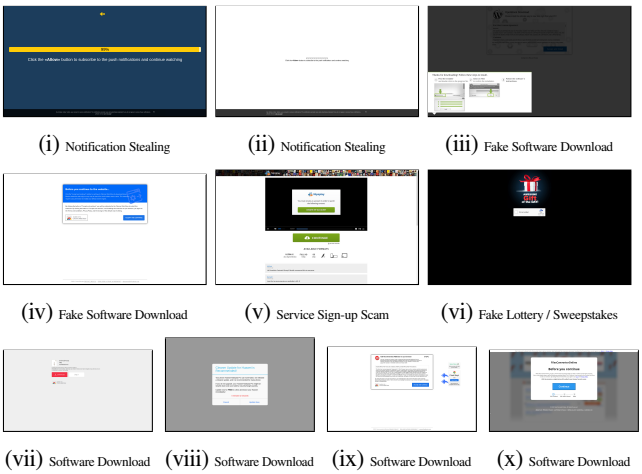
Test Res.	# Benign	# BMA	AUROC	DR at 1% FP
1366x768	500	54	1.0	1.0
800x1280	500	57	1.0	1.0
1920x998	500	41	1.0	1.0
414x896	500	12	1.0	1.0
1478x837	500	34	0.999	1.0
768x1024	500	33	1.0	1.0
1536x824	500	474	1.0	1.0
360x640	500	198	0.999	0.990
1366x728	500	354	1.0	1.0
Global	4500	1257	0.999	0.998

4.3. Generalization to New BMA Campaigns

RQ3: Can PP_{det} identify web pages belonging to never-before-seen BMA campaigns?

To setup this experiment, we randomly selected 10 BMA campaigns from our master dataset discussed in Section 3.2, $\{c_1,...,c_{10}\}$, to be excluded (in turn) from training and used for testing, and randomly selected 5,000 benign test pages (not seen during training) from randomly chosen screen resolutions. Example screenshots (one per campaign) for the 10 campaigns $\{c_1,...,c_{10}\}$ are shown in Figure 2. Notice that while popular BMA categories, such as Software Download and Notification Stealing are (by chance) represented multiple times, the specific campaigns within those categories are different (e.g., different fake software being distributed and different visual appearance of the attack pages).

We then trained 10 models, $\{T_1,...,T_{10}\}$. For each model, T_i, we excluded one BMA campaign, c_i, and 500 randomly chosen benign images from training. We then tested the model on the excluded BMA campaign and benign images. The results of this experiment are reported in Table 4.



(i) Notification Stealing (ii) Notification Stealing (iii) Fake Software Download

(iv) Fake Software Download (v) Service Sign-up Scam (vi) Fake Lottery / Sweepstakes

(vii) Software Download (viii) Software Download (ix) Software Download (x) Software Download

Figure 2: Examples of screenshots from each of the 10 randomly selected test campaigns.

We can see that PP_{det} is able to generalize very well to never-before-seen attack campaigns, with an AUROC score at or above 0.996 and detection rates at or above 99% at 1% false positives for all test campaigns.

TABLE 4: Results for the generalization to never-before-seen BMA campaigns.

Model name	# Benign	# BMA	AUROC	DR at 1% FP
Campaign 1	500	281	1.0	0.993
Campaign 2	500	229	1.0	1.0
Campaign 3	500	59	0.998	1.0
Campaign 4	500	67	1.0	1.0
Campaign 5	500	910	0.999	0.998
Campaign 6	500	15	0.996	1.0
Campaign 7	500	43	1.0	1.0
Campaign 8	500	83	1.0	1.0
Campaign 9	500	224	1.0	1.0
Campaign 10	500	128	1.0	1.0
Global	5000	2039	0.999	0.993

4.4. Generalization to Fresh BMA Campaigns

RQ4: Can PP_{det} accurately identify *fresh* BMA attacks that were collected well after the training data?

Real-world security systems must contend with the continuous evolution of attack content over time. Adversaries frequently modify or redesign the visual and textual elements of scam pages, rendering previously learned patterns less effective and introducing new features that may lie entirely *out of distribution* with respect to the training data. This phenomenon—known as temporal drift—makes generalization over time a particularly difficult challenge. In such cases, the model must correctly identify BMAs that appear days, weeks, or even months after the training process has concluded.

To evaluate this, we tested the temporal generalization capabilities of the same model used in **RQ1** (see Subsection 4.1). Specifically, we assessed its performance on a new BMA test set constructed from all data collected from our second BMA collection (see Section 3.1). To create the benign test set, we take 500 randomly selected images never seen in training from the benign data collection run.

Despite the difficulty of this setting, PP_{det} achieves an AUROC score above 0.999 and a detection rate of 97.8% at 1% false positives. These results suggest that PP_{det} can effectively generalize to previously unseen and temporally distant attack content.

4.5. Adversarial Examples Evaluation

RQ5: Can PP_{det} be strengthened against adversarial examples?

Because BMA pages are under the attacker's control, the attacker can attempt to craft attack pages that explicitly attempt to evade PP_{det} by injecting adversarial noise. At the same time, attackers must operate under two concurrent constraints: (i) attack pages must maintain their visual attack components, and (ii) the level of injected noise should be limited so not to be noticeable to users.

It is well known that adversarial examples forged by adding small perturbations can reduce deep learning models' classification accuracy [36] and that adversarial training is a popular approach to make models more robust to them [37], though perfectly defending against adversarial attacks is still a very open research question [38], [39].

Additionally, our model is unique in that it takes multimodal input in the form of images and their OCR extracted text. This opens another surface to the attacker. They have the ability to perturb the images to not only attempt to throw off the vision-based feature extractor, but also the OCR model. For instance, it has been shown that well-crafted image perturbations can cause OCR systems to misidentify, jumble, or even replace text with its semantic opposite [40], [41]. However, PP_{det} does not directly train an OCR model and instead uses a pre-trained one (Tesseract [25]). So, in order to achieve an effect similar to OCR-specific adversarial examples, we directly modify the textual input to our detection model, simulating the kinds of perturbations that may be introduced by adversarial changes to the image itself, according to previous work [40], [41].

In this section, we first test PP_{det} against both white-box adversarial attacks on the image side and black-box adversarial attacks on the text side, and then show that adversarial training can significantly increase its robustness to such attacks.

Visual Perturbation. For visual perturbation, we follow guidance from highly cited prior work [42] and use projected gradient descent (PGD) to construct adversarial images in a white-box attack setting. We implement these attacks using the Foolbox library [43], [44], targeting the visual feature extractor within our model.

The level of perturbation is controlled by the ϵ parameter. Larger values of ϵ produce more visible artifacts to the human eye and increase the likelihood of model evasion. We experiment with five levels of perturbation using $\epsilon \in \{2,4,8,16,32\}/255$. As shown in Figure 5 in Appendix G, noise is barely perceptible at $\epsilon = 2/255$, but becomes clearly visible by $\epsilon = 32/255$.

Textual Perturbation. For textual perturbation, we define five levels of increasing perturbation severity, drawing from estab-

lished techniques in adversarial NLP (e.g., TextAttack [45]) and recent work on adversarial OCR examples [40], [41]. Each level introduces progressively more aggressive distortions to the text:

- Level 1: Minor character-level noise that mimics OCR or typo artifacts.
- Level 2: Adds light paraphrasing or synonym substitution while preserving overall meaning.
- Level 3: Introduces misleading insertions or reordering that partially alters semantics.
- Level 4: Aggressively inverts meaning using antonyms, sentiment flips, or false claims.
- Level 5: Heavy semantic and character-level distortion that obscures or breaks meaning.

To perform these perturbations at scale, we use OpenAI's `gpt-4.1-mini` model via the API [46]. Given the original input text, we prompt the model to generate perturbations corresponding to each of the five defined levels. Additional details on the prompting strategy, system setup, and evaluation of the resulting perturbations are provided in Appendix G.

TABLE 5: Results of adversarial training experiments

		DR at 1% FP	
Level	ϵ	*Before adv. training* (*Exp.1*)	*After adv. training* (*Exp.2*)
clean	clean	1.0	1.0
1	$2/255$	0.968	0.998
2	$4/255$	0.816	0.998
3	$8/255$	0.840	0.998
4	$16/255$	0.564	0.982
5	$32/255$	0.040	0.994

The third column of Table 5 (*Exp.1*) reports the detection rate of our PP_{det} model when evaluated on adversarial examples, using the same evaluation setup described in **RQ1** in Section 4.1. These adversarial inputs consist of image perturbations generated using PGD with varying ϵ values, combined with text perturbations applied to the corresponding OCR-extracted content. As the attack strength increases—both in terms of visual distortion (ϵ) and textual manipulation level—the detection rate at a 1% false positive threshold drops significantly. Notably, at Level 4 and where $\epsilon = 16/255$, the detection rate falls to 0.564, despite the perturbations appearing only slightly noticeable to human users (see Figure 5, in the Appendix). At the highest attack level (Level 5), performance degrades drastically, with a detection rate of only 0.040, highlighting the model's vulnerability under coordinated multimodal adversarial conditions.

Adversarial Training. To harden PP_{det} against coordinated multimodal attacks, we retrain the model with an adversarial curriculum. Starting from the clean image–text pairs described in Section 4.1, we synthesize five tiers of adversarial examples by (i) applying PGD to the screenshots with $\epsilon \in \{2,4,8,16,32\}/255$ and (ii) injecting the corresponding Level 1-5 text perturbations. Each training epoch is built from 10,000 clean pairs augmented with 2,000 adversarial pairs from every tier, yielding a balanced mixture of benign, weak, and strong attacks.

The adversarially trained model (Table 5, *Exp.2*) maintains a detection rate above 98 % at 1 % FPR even under the most severe Level 5 attack, confirming that the curriculum substantially improves robustness while preserving performance on clean inputs.

4.6. Pixel Patrol Defend Evaluation

To evaluate PP_{def}, we examine its performance along two key dimensions: (1) user-facing latency, with a focus on the time required for inference during page analysis, and (2) device resource utilization, specifically CPU and RAM overhead. These metrics were measured using the setup described in Section 3.3 across representative hardware classes, including desktop, tablet, and mobile devices. Our goal is to assess whether PP_{def} can provide timely and accurate security verdicts without disrupting the user's browsing experience or overloading system resources. Results in this section pertain to the Firefox version of the extension to facilitate comparison across device types as mentioned in Section 3.3. [1]

Importantly, latency should be interpreted in the context of user reaction time. Recent studies show that users typically take several seconds, often more than eight [47], to cognitively process and make decisions about new or dynamically changing web pages. This provides a practical upper bound for acceptable detection delays and reinforces the feasibility of real-time inference within this window. Moreover, because the extension operates in a non-blocking manner, these background computations do not interfere with or delay the user's browsing experience in any perceptible way. In practice, our objective is to alert the user within few secods from the rendering of a BMA page, before they take potentially harmful actions, while remaining unobtrusive to typical browsing behavior.

Latency. We evaluate the latency experienced by users when interacting with PP_{def}, focusing on the portion of time attributable to the most computationally intensive component: the inference pipeline. This includes down-sampling the screenshot, executing four parallel OCR workers, running the fused ONNX vision-text model, and returning the final verdict to the background script.

Figure 3 presents the CDF of total inference latency across a range of representative devices. As expected, the Apple M4 Max outperforms all others with a median latency of only 388 ms and a 95th-percentile latency of only 618 ms. The Dell 5420 follows closely (median: 859 ms, 95th: 1482 ms), well below the default 5-second scan interval. For mid-range mobile-class devices, the Samsung A55 reaches the P50 and P95 thresholds at 1662 ms and 3162 ms, respectively. The slowest performance was observed on the Samsung Tab S9 FE (median: 2653 ms; 95th: 6972 ms). This is expected, as our extension implementation is a research prototype and the Tab S9 FE is a resource-constrained device with limited CPU and RAM compared to the others. A native implementation of the extension in the browser would likely improve performance on such devices very significantly. We leave this engineering optimization for future work.

It is also worth noting that these latency values are well aligned with human-computer interaction norms. For instance, Nielsen's foundational work [48] defines 1 second as the upper bound for seamless interaction, and 10 seconds as the limit of user attention. More recent research [49], [50] confirms that users begin to experience noticeable frustration only when delays exceed 7-10 seconds. Our measurements show that even on resource-constrained mobile and tablet devices, inference

1. Examples of the extension in action are available at https://pixelpatrol3d.github.io/.

typically completes well within these bounds. Additionally, user studies suggest that individuals typically do not make immediate navigational decisions following a page load or content change. For example, a recent industry study found that users take an average of 8.5 seconds to make their first click on live webpages [47]. Another large-scale analysis of Google user sessions reported an even longer average first click time of 14.6 seconds [51]. These findings indicate that users generally spend several seconds processing a page before interacting with it. Given that PP_{def} consistently completes inference well within this decision window on most devices, its latency is likely sufficient to warn users before they take potentially harmful actions, while remaining unobtrusive to typical browsing behavior.

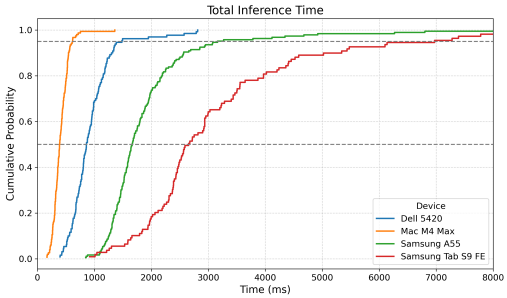


Figure 3: CDF of total inference latency for PP_{def}. Dashed lines indicate median (P50) and 95th-percentile (P95) thresholds.

Moreover, real-world usage further mitigates user impact. Empirical data shows that a small fraction of domains dominate browsing activity, with over one-third of all web traffic concentrated in just the top 116 sites [52]. For such high-confidence domains, our system bypasses full inference entirely, resulting in a worst-case latency of less than 100 ms. Even when encountering non-whitelisted pages, PP_{def} only triggers inference upon significant page changes. In steady states, it relies on relatively low-cost perceptual hash (phash) checks, which further reduces latency.

Taken together, these results demonstrate that PP_{def} delivers security-relevant classification with latencies that are both computationally efficient and behaviorally unobtrusive.

Device Resource Usage. To assess the runtime efficiency of PP_{def}, we also measured changes in device-side CPU and RAM utilization during operation relative to an idle browser baseline. These deltas were computed over the same test sessions described in Section 3.3, allowing us to quantify the additional computational burden incurred by the extension. CPU and RAM utilization deltas were computed by randomly sampling equal-length subsets of CPU and RAM usage values from both the idle (baseline) and extension-enabled (active) logs, and then taking element-wise differences. This ensured balanced comparisons even when log durations varied across sessions. Figure 4 summarizes these results. It should be noted again that ours is a research prototype implementation and that a native in-browser implementation of the detection model in the browser would likely improve performance on such devices very significantly. We leave more extensive engineering effort for future releases.

Overall, the increase in CPU usage across devices remained moderate and consistent with expectations for lightweight background inference. On the Dell 5420 laptop, CPU deltas were tightly bounded, with a median increase of 10.4% and an interquartile range (IQR) of 1.1% to 17.4%, indicating stable, moderate overhead. The Mac M4 Max exhibited comparable performance, with a slightly lower median of 7.0% but a wider IQR from –10.6% to 34.0%, reflecting greater variability likely due to background processes or dynamic power scaling.

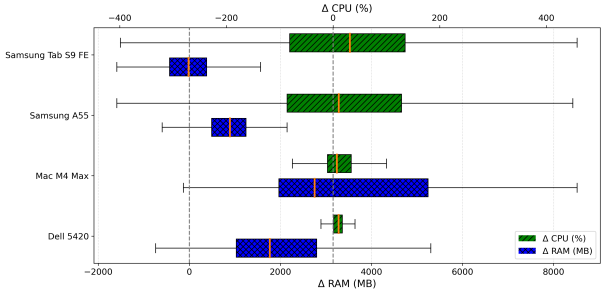


Figure 4: Difference in resource usage compared to the baseline during testing. CPU usage is measured as a percentage of total logical cores (e.g., 800% = full use of 8 cores).

Mobile-class devices showed greater variance, as expected given their constrained thermal envelopes and aggressive power management. The Samsung A55 had a median CPU delta of 10.75%, similar to desktop-class systems, but a much broader IQR of –86.1% to 128.6%, reflecting larger spikes from OCR or inference activity. The Tab S9 FE showed the highest central CPU usage, with a median increase of 31.5% and an IQR from –81.6% to 135.1%, again highlighting the bursty nature of on-device workloads. These spikes are short-lived, with inference completing quickly (see Section 4.6), and do not imply sustained load.

Memory usage followed a similar trend. On desktop-class devices, the extension incurred a moderate footprint, with median RAM increases of 1.77 GB (Dell 5420) and 2.75 GB (Mac M4 Max). The IQRs—1.03 GB to 2.80 GB for the Dell and 1.97 GB to 5.24 GB for the Mac—reflect differing levels of memory pressure on each device. In both cases, however, the overhead remained comfortably within the systems' available memory.

On mobile devices, memory usage was more constrained. The Samsung A55 showed a median RAM increase of 893 MB, with an IQR from 490 MB to 1.24 GB. In contrast, the Tab S9 FE had a median RAM delta of –17 MB and an IQR from –434 MB to 380 MB, suggesting frequent memory reclamation or reuse by the Android OS under pressure. This behavior is common on mobile systems where memory optimization is tightly integrated into runtime scheduling.

Taken together, these results show that while PP_{def} introduces measurable CPU and RAM overhead, the impact is transient, device-dependent, and consistent with what is expected from lightweight, on-device inference systems. Crucially, these resource demands do not persist across sessions and are tightly scoped to the moment of inference, ensuring minimal disruption to the user's broader browsing experience. Moreover, with additional engineering effort to integrate the detection system directly into the browser's codebase, its overhead could be reduced significantly.

5. Discussion and Limitations

In this section, we outline additional considerations and limitations of our framework.

Generalization Limits. While PP_{det} achieves high accuracy on new screen sizes, attack instances, and full campaigns (Sections 4.2 and 4.3), it may fail when presented with pages whose visual or textual features differ sharply from any seen during training. In practice, most BMAs rely on obvious visual cues or attack-related text designed to entice users [18], so completely novel campaigns are rare. Still, continuous crawling and periodic retraining are necessary to incorporate emerging styles into the training set and gradually close these blind spots.

Analysis of False Positives. Although uncommon, some benign pages can be flagged as malicious. For example, the screenshot in Table 9(1) (Appendix H) comes from a benign site (sariascosmetics.com) but was misclassified because its "Spin to Win" popup closely resembles the fake-lottery design used in many BMAs. Table 9(2) shows a legitimate lottery-style popup; the model has learned to associate this pattern with malicious content. Another case, Table 9(3), depicts a benign captcha page (ozbargain.com.au) whose layout is similar to "Notification Stealing" BMAs that use fake "I'm not a robot" challenges to trick users into clicking "Allow" on a notifications permission popup (the popup itself is rendered outside the viewport). In both examples, the model mistakenly treated harmless UI elements as BMA indicators.

Analysis of False Negatives. Conversely, some malicious pages may slip through as benign if their layout or text diverges significantly from the training examples. Tables 9(5) and 9(6) (Appendix) illustrate two such cases from the *Secondary BMA Collection* (Section 3.1): both use unconventional designs that the model had not encountered during training. Despite this, PP_{det} still detected 97.8% of BMAs at 1% false positive rate in the never-before-seen secondary campaigns collected months after training. Regular crawling and retraining remain the best strategy for reducing these blind spots over time.

6. Related Works

Most previous research on BMA type attacks focus on specific categories. For instance, Miramirkhani et al. [6] performed an analysis of TSS scams and reported websites and phone numbers used by scammers. Stone-Gross et al. [5] studied how users are persuaded to install fake AVs by employing scareware attacks. Kharraz et al. [7] employed an ML model to detect online survey scams that led victims to reveal personal information for fake prizes or content. Our work is different because we focus on creating an effective and practical in-browser system for detecting *BMAs* more broadly without limitation to a particular category.

Other studies have focused primarily on identifying and measuring the occurrence of BMA campaigns, without offering a detection solution. For instance, Subramani et al. [19] proposed a system called PushAdMiner to collect and discover web push notification messages that can deliver BMA type campaigns, whereas Vadrevu et al. [18] introduced a measurement system that automatically collects examples of BMAs and identifies previously unknown ad networks that promote BMA campaigns.

Recently, Yang et al. [22] proposed a first approach towards detecting and blocking BMA attacks in the browser. The proposed system, named TRIDENT, primarily targets malicious ads injected into publishers' webpages by low-reputation ad networks. Malicious ads are non-traditional ads that themselves utilize SE techniques, such as transparent overlays that perform clickjacking. TRIDENT is able *indirectly* detect these attacks by identifying malicious ads. However, not all BMAs are distributed via such ads. Unlike [22], our framework directly aims at detecting BMAs and by recognizing their visual and semantic traits.

There is also a large body of research that focuses on detecting IHAs. While these attacks can be considered a subclass of Social Engineering [53], they are characterized by different visual traits and attack tactics, as we discussed in Section 1. Recent research has focused on detecting IHA websites using visual cues. For instance, Abdelnabi et al. [11] used convolutional networks to detect IHA pages by visual similarity. Lin et al. [12] detect IHA pages by visually detecting abused company logos. Liu et al [13] presented a technique that is a combination of machine learning and browser instrumentation to detect an IHA page by not only using visual cues but also discovering the intention of an IHA webpage that perform credential stealing via web forms. Unlike the above solutions, our system is able to detect BMAs that differ from IHAs, even if no specific benign website/logo is abused or in absence of credential stealing attempts.

Hao et al.'s recent work [54] uses an adversarial diffusion model to morph logos in IHA pages, subtly altering them to successfully evade detection while maintaining their recognizability to an end user. However, applying such techniques to BMA pages that our work focuses on would require altering multiple elements, potentially raising user suspicion due to the extensive changes needed.

7. Conclusion

We presented *Pixel Patrol 3D*, a browser-based framework that detects and defends against behavior-manipulation attacks (BMAs) using large-scale discovery, a resolution-agnostic multimodal model, and a lightweight, privacy-preserving extension. *PP3D* achieves high accuracy on unseen screen sizes, new campaigns, and temporally distant data, with low latency and modest resource use. These results highlight its practicality and effectiveness against a largely overlooked class of social engineering threats.

References

[1] I. Mann, *Hacking the Human: Social Engineering Techniques and Security Countermeasures*. Ashgate Publishing, Limited, 2008. [Online]. Available: https://books.google.com/books?id=1veE2f9rPOgC

[2] K. Krombholz, H. Hobel, M. Huber, and E. Weippl, "Advanced social engineering attacks," *Journal of Information Security and Applications*, vol. 22, pp. 113–122, 2015, special Issue on Security of Information and Networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214212614001343

[3] P. Zhang, A. Oest, H. Cho, Z. Sun, R. Johnson, B. Wardman, S. Sarker, A. Kapravelos, T. Bao, R. Wang *et al.*, "Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1109–1124.

[4] K. Subramani, W. Melicher, O. Starov, P. Vadrevu, and R. Perdisci, "Phishinpatterns: measuring elicited user interactions at scale on phishing websites," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 589–604.

[5] B. Stone-Gross, R. Abman, R. A. Kemmerer, C. Kruegel, D. G. Steigerwald, and G. Vigna, "The underground economy of fake antivirus software," in *Economics of Information Security and Privacy III*, B. Schneier, Ed. New York, NY: Springer New York, 2013, pp. 55–78.

[6] N. Miramirkhani, O. Starov, and N. Nikiforakis, "Dial One for Scam: A Large-Scale Analysis of Technical Support Scams," in *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS)*, 2017.

[7] A. Kharraz, W. K. Robertson, and E. Kirda, "Surveylance: Automatically detecting online survey scams," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018, pp. 70–86. [Online]. Available: https://doi.org/10.1109/SP.2018.00044

[8] J. Liu, P. Pun, P. Vadrevu, and R. Perdisci, "Understanding, measuring, and detecting modern technical support scams," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 18–38.

[9] M. Khonji, Y. Iraqi, and A. Jones, "Phishing detection: A literature survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2091–2121, Fourth 2013.

[10] S. Afroz and R. Greenstadt, "Phishzoo: Detecting phishing websites by looking at them," in *Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC 2011), Palo Alto, CA, USA, September 18-21, 2011*. IEEE Computer Society, 2011, pp. 368–375. [Online]. Available: https://doi.org/10.1109/ICSC.2011.52

[11] S. Abdelnabi, K. Krombholz, and M. Fritz, "Visualphishnet: Zero-day phishing website detection by visual similarity," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, 2020, p. 1681–1698.

[12] Y. Lin, R. Liu, D. M. Divakaran, J. Y. Ng, Q. Z. Chan, Y. Lu, Y. Si, F. Zhang, and J. S. Dong, "Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages." in *USENIX Security Symposium*, 2021, pp. 3793–3810.

[13] 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)*, 2022, pp. 1633–1650.

[14] R. Liu, Y. Lin, Y. Zhang, P. H. Lee, and J. S. Dong, "Knowledge expansion and counterfactual interaction for {Reference-Based} phishing detection," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4139–4156.

[15] R. Liu, Y. Lin, X. Teoh, G. Liu, Z. Huang, and J. S. Dong, "Less defined knowledge and more true alarms: Reference-based phishing detection without a pre-defined reference list," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 523–540.

[16] Y. Li, C. Huang, S. Deng, M. L. Lock, T. Cao, N. Oo, B. Hooi, and H. W. Lim, "Knowphish: Large language models meet multimodal knowledge graphs for enhancing reference-based phishing detection," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[17] "FBI Releases Annual Internet Crime Report." [Online]. Available: https://www.fbi.gov/news/press-releases/fbi-releases-annual-internet-crime-report

[18] P. Vadrevu and Roberto Perdisci, "What you see is NOT what you get: Discovering and tracking social engineering attack campaigns," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC, 2019.

[19] K. Subramani, X. Yuan, O. Setayeshfar, P. Vadrevu, K. H. Lee, and R. Perdisci, "When push comes to ads: Measuring the rise of (malicious) push advertising," ser. IMC '20, 2020, p. 724–737.

[20] M. E. Blog and M. E. Team, "Stand up to scareware with scareware blocker, now available in preview in Microsoft Edge," Jan. 2025. [Online]. Available: https://blogs.windows.com/msedgedev/2025/01/27/stand-up-to-scareware-with-scareware-blocker/

[21] "Google safe browsing," https://safebrowsing.google.com/.

[22] Z. Yang, J. Allen, M. Landen, R. Perdisci, and W. Lee, "TRIDENT: Towards detecting and mitigating web-based social engineering attacks," in *USENIX Security Symposium*. USENIX, 2023.

[23] M. Zaoui, B. Yousra, S. Yassine, M. Yassine, and O. Karim, "A comprehensive taxonomy of social engineering attacks and defense mechanisms: Toward effective mitigation strategies," *IEEE Access*, vol. 12, pp. 72 224–72 241, 2024.

[24] "ONNX Runtime|Home." [Online]. Available: https://onnxruntime.ai/

[25] "Tesseract.js|Pure Javascript OCR for 100 Languages!" [Online]. Available: https://tesseract.projectnaptha.com/

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[27] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-Read Students Learn Better: On the Importance of Pre-training Compact Models," Sep. 2019, arXiv:1908.08962 [cs]. [Online]. Available: http://arxiv.org/abs/1908.08962

[28] "A research-oriented top sites ranking hardened against manipulation - Tranco — tranco-list.eu," https://tranco-list.eu, [Accessed 13-Jun-2023].

[29] "commonsmachinery/blockhash," Mar. 2025, original-date: 2014-09-02T17:46:34Z. [Online]. Available: https://github.com/commonsmachinery/blockhash

[30] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[31] "Common Crawl - Open Repository of Web Crawl Data." [Online]. Available: https://commoncrawl.org/

[32] K. Krippendorff, "Computing Krippendorff's Alpha-Reliability," no. 43, Jan. 2011. [Online]. Available: https://repository.upenn.edu/handle/20.500.14332/2089

[33] K. De Swert, "Calculating inter-coder reliability in media content analysis using krippendorff's alpha," *Center for Politics and Communication*, vol. 15, pp. 1–15, 2012.

[34] P. Perera and V. M. Patel, "Learning deep features for one-class classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 802–811.

[35] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[36] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

[37] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, "Recent advances in adversarial training for adversarial robustness," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2021, pp. 4312–4321, survey Track. [Online]. Available: https://doi.org/10.24963/ijcai.2021/591

[38] H. Zhu, S. Zhang, and K. Chen, "Ai-guardian: Defeating adversarial attacks using backdoors," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 701–718.

[39] N. Carlini, "A llm assisted exploitation of ai-guardian," 2023.

[40] N. Imam, V. Vasilakis, and D. Kolovos, "OCR post-correction for detecting adversarial text images."

[41] C. Song and V. Shmatikov, "Fooling OCR Systems with Adversarial Text Images," Feb. 2018, arXiv:1802.05385 [cs]. [Online]. Available: http://arxiv.org/abs/1802.05385

[42] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[43] J. Rauber, R. Zimmermann, M. Bethge, and W. Brendel, "Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax," *Journal of Open Source Software*, vol. 5, no. 53, p. 2607, 2020. [Online]. Available: https://doi.org/10.21105/joss.02607

[44] J. Rauber, W. Brendel, and M. Bethge, "Foolbox: A python toolbox to benchmark the robustness of machine learning models," in *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017. [Online]. Available: http://arxiv.org/abs/1707.04131

[45] "QData/TextAttack," May 2025, original-date: 2019-10-15T00:51:44Z. [Online]. Available: https://github.com/QData/TextAttack

[46] "Overview - OpenAI API." [Online]. Available: https://platform.openai.com

[47] J. S. PhD, W. S. PhD, E. Short, and J. L. PhD, "First Click Times on Websites Versus Images – MeasuringU." [Online]. Available: https://measuringu.com/first-click-times-on-websites-versus-images/

[48] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Oct. 1994.

[49] T. Abbas, U. Gadiraju, V.-J. Khan, and P. Markopoulos, "Understanding User Perceptions of Response Delays in Crowd-Powered Conversational Systems," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, no. CSCW2, pp. 345:1–345:42, Nov. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3555765

[50] I. Arapakis, S. Park, and M. Pielot, "Impact of Response Latency on User Behaviour in Mobile Web Search," Jan. 2021. [Online]. Available: https://arxiv.org/abs/2101.09086v1

[51] "How People Use Google Search (New User Behavior Study)," Aug. 2020. [Online]. Available: https://backlinko.com/google-user-behavior

[52] H. S. Xavier, "The Web unpacked: a quantitative analysis of global Web usage," Apr. 2024. [Online]. Available: https://arxiv.org/abs/2404.17095v2

[53] W. Syafitri, Z. Shukur, U. Asma'Mokhtar, R. Sulaiman, and M. A. Ibrahim, "Social engineering attacks prevention: A systematic literature review," *IEEE Access*, vol. 10, pp. 39 325–39 343, 2022.

[54] "It doesn't look like anything to me: Using diffusion model to subvert visual phishing detectors," https://qingyinghao.web.illinois.edu/files/USENIX24-visual-phish.pdf.

[55] "Phishtank: Phishing intelligence," https://phishtank.com/.

[56] "Openphish: Phishing intelligence," https://openphish.com/.

[57] "Puppeteer — Puppeteer — pptr.dev," https://pptr.dev, [Accessed 13-Jun-2023].

[58] "puppeteer-extra-plugin-stealth — npmjs.com," https://www.npmjs.com/package/puppeteer-extra-plugin-stealth, [Accessed 13-Jun-2023].

[59] StatCounter, "Desktop Screen Resolution Stats Worldwide, December 2020," https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide, [Accessed 21-Jun-2023].

[60] M. Z. Rafique, T. Van Goethem, W. Joosen, C. Huygens, and N. Nikiforakis, "It's free for a reason: Exploring the ecosystem of free live streaming services," in *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS 2016)*. Internet Society, 2016, pp. 1–15.

[61] "The best cpm ad networks for publishers," https://publishergrowth.com/category/ad-networks/cpm-ad-networks.

[62] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[63] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *EMNLP*, 2019.

[64] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *NeurIPS*, 2020.

[65] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

Appendix

A. BMA vs IHA Examples

Examples of BMA and IHA pages are shown in Table 6 to illustrate the differences between the two sub classes of social engineering attacks.

B. Pixel Patrol Discover Details

While research on phishing defenses can leverage URL feeds such as PhishTank [55] and OpenPhish [56] to collect ground truth data, to our best knowledge, there exist no analogous feeds or repositories that would allow us to readily collect fresh samples of BMA. Therefore, we had to build our own system, called *Pixel Patrol Discover* (PP_{dis}), for discovering and collecting recent examples of such attacks in the wild, by extending the BMA campaigns discovery approach proposed in [18]. Although PP_{dis} is not the main contribution of our work, we introduce important improvements that enable better data collection and more systematic labeling, which we describe below.

Briefly, the approach we use is based on a *crawler farm* that is seeded with URLs that are likely to lead to BMAs. A crawler consists of an instrumented browser that loads a seed URL and automatically interacts with web pages in an attempt to trigger a redirection to a BMA. A number of heuristics are used to pilot the browser to collect *landing pages* that have a high likelihood of being related to BMAs. As these landing pages are visited, the crawler records the related URL and a screenshot. Then, to discover BMA campaigns and filter out noise (i.e., non-BMA pages), a webpage screenshot clustering algorithm is used, which is based on the intuition that BMAs are typically orchestrated into BMA campaigns that distribute the same (or very similar) attacks across multiple malicious domain names.

We re-implement [18] using Puppeteer [57] and extended it to add the following improvements:

– *Improved crawler heuristics:* In [18], "click-worthy" page elements were prioritized by sorting them simply in decreasing element size order (i.e., based solely on the size of an element on the rendered webpage). To allow PP_{dis} to more quickly reach BMA content, we empirically derived additional heuristics. For instance, we prioritize interacting with page elements (e.g., buttons, images, links, etc.) that include keywords such as *update*, *download*, *play*, etc. (we used a total of 38 keywords often found on pages that lead to BMAs). Additionally, we added the use of [58] to improve the "stealthiness" of our crawlers (i.e., make it more difficult for pages to detect browser automation) and yield a larger variety of BMAs.

– *Screen size diversity:* Because one of our main goals is to develop a practical defense that can be deployed on devices with widely different screen resolutions, we significantly extended the number of devices and screen sizes emulated by our crawlers, from 5 to 30; To select the screen sizes and devices to be emulated, we relied on external statistics [59] as well as results from a prior IRB-approved user study (from an unrelated research project) involving 400 MTurk users that allowed us to derive the most popular viewport sizes and screen aspect ratios to be embedded in our PP_{dis} system.

– *Systematic labeling process:* Unlike in [18], we use a systematic labeling approach. Namely, we label potential BMA campaigns and remove non-BMA pages by using three different human

labelers with expertise on web security and social engineering attacks who were tasked with labeling each cluster of webpage screenshots as either *BMA Campaign* or *benign* (i.e., not BMA).

After the labelers independently completed their tasks, they met to discuss and attempt to resolve possible label disagreements. In the rare cases in which consensus on the label to be assigned could not be found, they marked the related cluster as *unknown* and excluded it from further consideration. This systematic data labeling process allowed us to collect higher quality BMA examples. We computed the inter-rate reliability score using Krippendorff's alpha score [32] among the labelers and obtained $\alpha = 0.82$, which indicates a high level of agreement [33]. Besides collecting and labeling examples of BMAs, we also used our PP_{dis} to collect benign webpages under 30 different screen resolutions by seeding it with urls as described in Section 3.1.

C. Crawler Setup

We use Docker to run many parallel instances of PP_{dis}'s instrumented browser. Each instance represents an isolated and clean environment that does not retain any session information related to crawling previous seed URLs. We deploy our crawlers on a server with 24 CPU cores running Ubuntu Linux, where we run 15 crawler instances at a time. For any given seed URL, we visit it using multiple instances of our instrumented browser, each configured to emulate one of 10 different browser/device combinations, including different smartphones, tablets, laptops, etc., and browsers that render pages with different viewport sizes (the browser/OS combinations we use include Firefox on Windows, Safari on Mac, Edge on Windows, Chrome on Windows, Chrome on Linux, Chrome on Mac, Chrome on an Android Phone, Safari on iPhone, Safari on iPad, and Chrome on different Android tablets). Additionally, to make our crawling as stealthy as possible and circumvent the anti-crawler mechanisms implemented by some ad networks, we used Puppeteer's "stealth" libraries [58].

D. Ethical considerations

PP_{dis} uses an approach similar to recent research [18], [19], [22], [60]. Like previous works, it is possible that our crawler inadvertently clicked on ads belonging to legitimate websites. As a result, genuine advertisers might face a small expense due to our system's data collection activity, since they are likely to compensate a third-party publisher and an ad network for their services (it is important to note that we do not profit financially from these activities). This scenario is not unique to our study but is also observed in other similar works. Consequently, we have explored the ethical implications of our research by drawing on established precedents [18], [19], [22].

To ensure that our actions do not adversely affect legitimate third parties, we assessed the financial impact our crawler might have on advertisers and determined that the effect is minimal. Specifically, we calculated the number of clicks that led our crawler to navigating to new domain names that were different from the visited domain where the click occurred. As it is difficult to determine if the navigation was related to clicking on an ad, we *overestimate* the number of ad-related navigations by considering every navigation to a third-party domain as an ad click. We then calculated the cost associated with visiting each landing domain by taking into account the number of times our system navigated to that domain, by applying the maximum Cost Per Mille (CPM) rate of $5 [61]. On average, each domain was visited 56 times by our system, resulting in an (overestimated) average expense of $0.28 per domain (or per advertiser). Given these minimal costs, we consider the impact on advertisers to be acceptable, especially in light of the research benefits: PP_{dis} enabled the collection of a large, in-the-wild BMA dataset and facilitated the development of a novel and effective in-browser defense. We will also share the collected dataset with the research community to foster further research.

E. Additional data

Table 7 presents additional details about our data collection. Table 7(a) lists the ad networks we used to find seed URLs, whereas Table 7(b) shows a detailed breakdown of our datasets in terms of number of images per screen resolution.

TABLE 7: Additional dataset details. Left: Seed URLs by ad network. Right: Samples per screen resolution for the master dataset.

(a)

Ad Network	Seed URLs
Adroll	21716
Adblade	2161
Adpushup	373
Adsupply	205
Pop Ads	178
Adcash	147
AdMaven	118
Ad4Game	42
AdReactor	24
Adsense	15

(b)

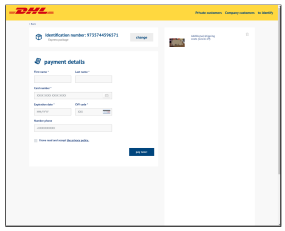
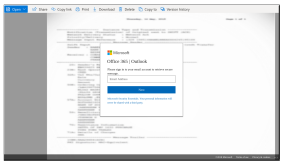
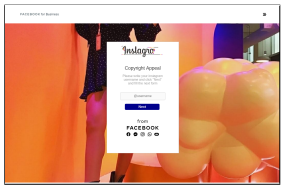
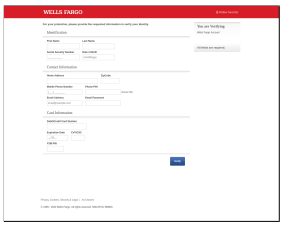
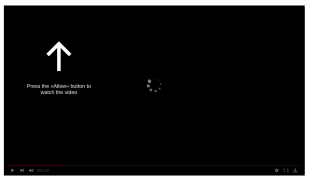
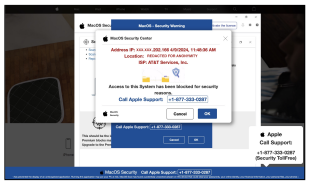
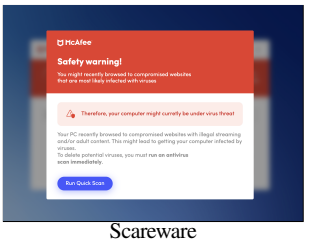
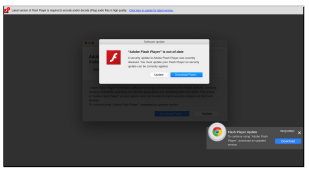
Resolution	Benign	BMA
360x640	8730	642
360x740	-	4062
414x896	8660	54
750x1334	1476	-
768x1024	8885	123
800x1280	8653	135
1024x768	840	54
1280x800	740	87
1200x803	1481	-
1366x677	52	21
1366x720	213	42
1366x724	60	12
1366x728	2800	891
1366x738	214	45
1366x741	96	21
1366x768	351	111
1478x837	167	75
1536x816	253	147
1536x824	2757	1200
1536x826	138	54
1536x834	394	198
1536x864	149	51
1785x993	-	8901
1858x1053	164	60
1858x1080	144	45
1920x998	384	90
1920x1032	292	111
1920x1040	3162	1251
1920x1050	473	156
1920x1052	145	36
1920x1080	47918	774
1920x1097	209	87
Total	**100000**	**19536**

F. Text Augmentation Details

To augment the OCR-extracted text data used in training, we employed Groq's cloud API with their LLaMA-3.3-70B-Versatile model to perform large-scale, batched synonym-based data augmentation. This approach aimed to increase the diversity of the training text while preserving the original intent and structure of each sample.

Batch Augmentation Pipeline. We implemented a fully asynchronous Python pipeline using the `asyncio` and `aiofiles` libraries to maximize throughput and minimize I/O bottlenecks. Text files were grouped into batches of ten and sent in parallel to Groq's API while respecting a rate limit of 100 requests per minute. A queue-based locking mechanism ensured compliance with Groq's rate-limiting policies. Each batch request was constructed by inserting up to ten input texts into a single structured prompt. The prompt instructed the model to perform

TABLE 6: Visual comparison between Information-Harvesting Attacks (IHA) and Behaviour-Manipulation Attacks (BMA). IHA pages mimic legitimate sites and solicit credentials, while BMA pages use visual deception to induce unsafe actions. We focus on developing a content-based defense against BMA pages, which are not addressed by existing defenses.

IHA				
BMA				
	Notification Permission Stealing	Technical Support Scam (TSS)	Scareware	Fake Software Download

keyword-based synonym replacement (focused on verbs, nouns, and adjectives) while explicitly avoiding cleanup of potentially nonsensical or adversarial content. The prompt also requested that outputs be labeled clearly as "Text 1:", "Text 2:", etc., to facilitate automated parsing of the response.

Prompt Format. The core prompt used in each batch was structured as follows:

```
Text 1: <original text 1>
Text 2: <original text 2>
...
Text 10: <original text 10>

Please perform data augmentation on the texts above.
Modify each text while preserving its overall meaning and
structure. Use keyword-based synonym replacement, focusing on
semantically important words (e.g., verbs, nouns, adjectives).
Do not clean up or correct any nonsensical phrases-leave them
unchanged. Each augmented response should be unique and writ-
ten in plain text. Please do not add any other information,
notes, or text other than what is described below. Clearly
label each output as 'Text #:' matching the input numbering.

Example: Text 1: This is an example response.
```

The model's response was parsed using regular expressions to extract the modified text for each numbered input. This method enabled scalable, semantic-preserving augmentation of BMA texts, significantly increasing lexical variability.

G. Adversarial Setup and Evaluation Details

Adversarial Text Generation. To generate adversarial textual examples, we use OpenAI's `gpt-4.1-mini` model via the official API. Our generation script reads raw OCR-extracted text files and produces five adversarial versions per input, each targeting a specific perturbation level. The five-tier perturbation scheme is designed to progressively degrade model performance while remaining plausible to a human reader.

We format each API prompt to include a detailed description of the five perturbation levels. Each batch includes one text sample (batch size = 1) to remain within the model's token limits and ensure precise control over the output. Responses are parsed and written to disk by level.

Prompt Format. The core prompt (split across 4 prompt boxes to conserve space) used in each batch was structured as follows:

```
You are an adversarial text generator.
Your task is to perturb
the input text to fool downstream models such as classifiers
or OCR engines. Generate five versions of the input text,
each with a different level of perturbation as defined below.
```

```
Perturbation Levels:
Level 1 - Minimal (light noise): Minor character-level
changes. Mostly readable and retains original meaning. Mimics
accidental typos or OCR noise. (e.g., swap adjacent letters,
lowercase/uppercase inconsistency, "O" to "0", "l" to "1")
Level 2 - Mild (readable but noisier): Combine character-
level noise with small paraphrases or word substitutions.
Still semantically faithful but noticeably altered.
```

```
Level 3 - Moderate (semantics shift): Introduce paraphrasing,
antonyms, misleading insertions, or reordered phrases.
Meaning is partially changed; models may misclassify it.
Level 4 - Strong (semantic inversion): Aggressively
alter meaning through word replacements, phrase reversals,
and misleading information. Still somewhat readable.
Level 5 - Extreme (highly adversarial): Severe character
and semantic distortion. Text is hard to read or nonsensical,
and original meaning is obscured or inverted completely.
```

```
Instructions:

1. For each
level, label the output clearly as Level 1:, Level 2:, etc.
2. Each version must be unique.
3. Do not explain your changes.
4. Return the output in plain text only.
```

This structure ensures that the model returns five labeled outputs corresponding to Levels 1 through 5, with each level representing an increasing degree of adversarial noise.

Evaluation Metrics. To quantify the perturbations, we compute three metrics between each adversarial version and its original text:

- Levenshtein Distance [62] — A character-level edit distance that measures how many insertions, deletions, or substitutions are required to transform the original text into the perturbed version.
- Semantic Similarity [63], [64] — Cosine similarity between sentence embeddings generated using the `all-MiniLM-L6-v2` model from `SentenceTransformers`, measuring semantic distortions between text variants.
- ROUGE-L F1 [65] — A token-level metric commonly used in summarization, capturing overlap between longest common subsequences.

Evaluation Pipeline. We sample 1,000 shared text examples across all perturbation levels. For each example, we compute the Levenshtein distance, semantic similarity, and ROUGE-L score between the original and each adversarial variant.

Quantitative Results. The table below summarizes the average perturbation metrics for each level:

TABLE 8: Average perturbation metrics across different perturbation levels. Higher Levenshtein Distance indicates greater character level changes. In contrast, lower scores for Semantic Similarity and ROUGE-L F1 indicate more significant semantic and structural deviations from the original input text.

Level	Levenshtein Distance	Semantic Similarity	ROUGE-L F1
1	33.96	0.931	0.940
2	150.31	0.934	0.753
3	194.96	0.884	0.684
4	211.80	0.815	0.600
5	273.61	0.264	0.124

These results confirm that perturbation strength increases consistently with each level. Level 1 introduces minor noise with minimal semantic degradation, while Level 5 causes substantial semantic drift and textual distortion. This quantitative validation supports the use of these adversarial levels for training and evaluation of robustness in PP_{det}.

Adversarial Image Generation Examples. Figure 5 shows examples of images perturbed using PGD at various levels of ϵ. At $\epsilon = 16/255$, the introduced noise becomes clearly perceptible to the human eye.

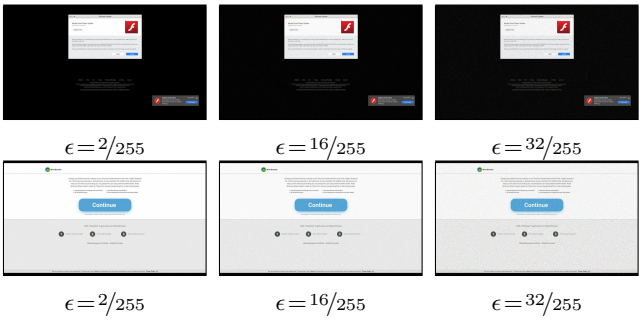
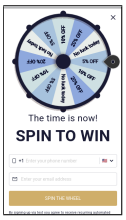


Figure 5: Adversarial examples generated using PGD at increasing perturbation strengths (ϵ).

H. Challenging BMA Examples

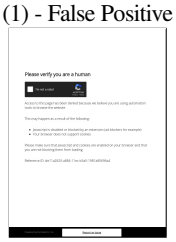
Table 9 presents examples of misclassified pages that illustrate the challenges of generalizing across evolving BMA attack strategies (discussed in Section 5).

TABLE 9: Examples of misclassified BMA pages.

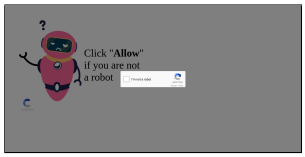


(1) - False Positive

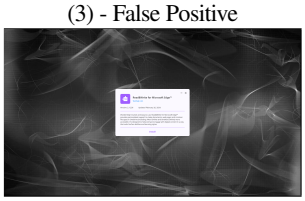
(2) - Malicious Case



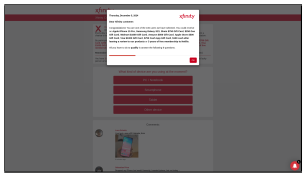
(3) - False Positive



(4) - Malicious Case



(5) - False Negative



(6) - False Negative

I. Screenshot Normalization Details

In Algorithm 1, we detail the screenshot normalization process used to standardize inputs across devices.

Algorithm 1 Normalizing Screenshots to Fixed Resolution

Input: I: Raw screenshot of arbitrary resolution
T: Target canvas size (1920×1080)
S: Downsampling factor (0.5)
Output: I': Normalized image of size 960×540

if I.width > 1920 **or** I.height > 1080 **then**
  scaleFactor = $\min(1920/I.\text{width}, 1080/I.\text{height})$
  I_{scaled} = Resize(I, scaleFactor)
else
  $I_{\text{scaled}} = I$
end if
C = Create zero-valued (RGB [0, 0, 0]) canvas of size 1920×1080
Center I_{scaled} on C
I' = Downsample(C, factor S)