You Only Perturb Once: Bypassing (Robust) Ad-Blockers Using Universal Adversarial Perturbations

Dongwon Shin
KAIST

Suyoung Lee
KAIST

Sanghyun Hong
Oregon State University

Sooel Son
KAIST

***Abstract*—Extensive academic effort has been put into the development of effective machine learning models that block advertising and tracking service (ATS) content. These ATS blockers leverage various features from websites, such as structural, content, flow, and JavaScript features, to develop accurate and robust models. However, establishing the robustness of these ATS blockers to evasion attacks is largely understudied, particularly in practical scenarios in which an adversary generates a single and cost-effective universal perturbation that renders ATS detection across websites ineffective at scale.**

In this paper, we show that recent ATS blockers using machine learning are not robust to a universal adversarial attack. Specifically, we propose an auditing framework (YOPO) that enables one to generate a single adversarial perturbation in a cost-effective manner. Our framework casts the generation of a universal perturbation into an optimization problem in a principled way; it enables an adversary to minimize the cost of manipulating various features in HTML content and to thwart ATS classification while constraining the perturbation size for each feature. We demonstrate that YOPO is capable of generating a universal perturbation that enables bypassing four seminal ATS blockers: AdGraph, WebGraph, AdFlush, and PageGraph, attaining success rates of up to 92.27%, 71.50%, 61.91%, and 85.81%, respectively. We also propose a practical and effective countermeasure against YOPO that only requires preprocessing training instances without large performance drops in ATS blocking.

1. Introduction

Advertising and tracking service (ATS) providers are renowned for tracking Internet users via various vectors [16], such as third-party cookies [21, 26, 34], first-party cookies [24, 50], device fingerprinting [47, 51], cache-based tracking [40, 59], and cross-device tracking [48, 62, 66]. They monetize users' browsing trajectories across websites by compiling their profiles to serve personalized ads [20].

For this privacy threat, ATS blocking services provide an effective way to prevent such ATS providers from tracking Internet users by blocking ATS resources. ATS blocking has contributed to improving the privacy of Internet users [18, 31, 33, 44, 46] and improving browsing performance [32, 55]. AdBlock, a notable ad-blocking extension, has over 65 million daily users, demonstrating its wide prevalence [3].

Recent studies have focused on developing ATS blockers by leveraging machine learning (ML) [30, 37, 38, 50, 56, 63, 64]. They have proposed effective features for ATS classification and improved robustness to evasion efforts. These improvements typically come from engineering new features that an adversary may not easily perturb and deploying ML models that leverage these robust features. For example, AdGraph [37] proposes features that encode how web resources are rendered in browsers by leveraging graph representations and achieves state-of-the-art ad-blocking performance. Siby *et al.* [56] present WebGraph, which improves the robustness of AdGraph against evasion attacks by employing information flow features. A recent work by Lee *et al.* [43] introduces AdFlush, a high-performing real-time ATS classifier.

However, these ATS blockers examine their robustness against adversaries with limited capabilities. WebGraph assumes a limited-capability adversary unable to easily manipulate the proposed flow features; AdFlush does not address the threat posed by adversarial examples artificially crafted through perturbing multiple features together.

In this paper, we challenge this assumption of the adversary's limitation and study the vulnerability of ATS blockers to evasion attacks. Specifically, we ask the question: *how can an adversary craft a single, cost-effective perturbation applicable to multiple websites for evading ATS blockers?* Unlike a few studies [56, 65] that test robustness to evasion attacks, by crafting adversarial perturbations tailored to individual websites, we focus on devising a practical attack scalable to millions of ads. This question is important from a security perspective, as successful attacks expose a common vulnerability in existing ATS blockers, even if their features are designed to provide better accuracy and robustness.

To answer the question above, we develop a novel framework (You Only Perturb Once: YOPO) that generates a universal adversarial perturbation (UAP) against a target ATS blocker. We manifest three challenges in generating a cost-effective UAP. (1) One cannot simply adapt the gradient-based techniques proposed by prior work [45], as ATS blockers employ discrete features, such as categorical and/or binary features. (2) In practical scenarios, an adversary should prioritize manipulating specific features to minimize the engineering cost of implementing perturbed features in HTML. Changing HTML tags, introducing extensively repetitive tags, or deleting existing DOM elements

2576-9103/24/$31.00 ©2024 IEEE
DOI 10.1109/ACSAC63791.2024.00031

can disrupt the functionality or proper display of a webpage. (3) The UAPs that our attacker exploits should be reflected in an HTML format while preserving the functionality of a target webpage.

To overcome these key challenges, we propose novel techniques for generating a single, cost-effective UAP. We formulate the UAP crafting process as an optimization problem and solve this problem using projected gradient descent (PGD) [45]. We devise a series of update functions that convert categorical and binary features into numerical vector representations, as well as reflect numerical feature changes in the discrete features. This enables numerical gradients to be reflected on each input feature through our update functions, thus addressing the first challenge. For the second challenge, we define a cost model that prioritizes which features to manipulate first in generating a UAP. This model is integrated into our objective function, which guides YOPO to compose a cost-effective and imperceptible UAP for the adversary to inject. Lastly, we implement a series of HTML update functions that apply a total of 46 feature changes to a target HTML webpage, thus addressing the third challenge.

We comprehensively evaluate our universal adversarial attacks against four seminal ATS blockers: AdGraph, WebGraph, AdFlush, and PageGraph. YOPO achieves high attack success rates of 92.27%, 71.50%, 61.91%, and 85.81% against these blockers, respectively. Our attack is practical and cost-effective, requiring only 32.42 seconds to generate a UAP, which is a one-time procedure. We also analyze 400 webpages manipulated by YOPO and demonstrate that our HTML manipulation incurs functionality disruptions against only 14 webpages. Moreover, we investigate the factors that result in this common vulnerability and find that it mainly stems from the imbalanced distribution of binary and categorical features in the training data.

Based on our findings, we finally propose two feature engineering-based mitigation strategies that exploit the imbalanced feature distribution in the training set. Employing our strategies reduces the attack success rates to 40.41%, 48.55%, 42.74%, and 64.51% against AdGraph, WebGraph, AdFlush, and PageGraph, respectively, while preserving the accuracy of these blockers.

2. Related Work

2.1. ATS Blocking

Blocklist-based ATS blocking. Leveraging blocklists is a classic approach to preventing ad service providers and publishers from tracking Internet users. These blocklists are manually maintained by trusted ATS blocking vendors. They contain a collection of regular expressions and keywords that match the URLs of network requests and HTML elements that fetch ATS resources. Once matched, browsers prevent fetching these resources [7]–[9]. However, this approach has limitations. It takes time to add new URL patterns to blocklists as their maintenance typically relies on crowdsourced reports [58]. Prior studies have also demonstrated a series of successful evasion attempts, such as domain rotation [2], domain generation [54], and collusion between first- and third-party websites [56, 65]. These limitations render the blocklist-based approach ineffective in keeping up with evolving ATS techniques [46].

ML-aided ATS blocking. Recent proposals have employed ML models for ATS blocking [14, 30, 37, 38, 50, 56, 63, 64]. Previous studies explored various ML algorithms that examine visual content, such as website screenshots [14, 30]. While they have shown effectiveness, focusing solely on visual (or perceptual) content may allow an attacker to add human-imperceptible perturbations to websites for evasion.

To address this shortcoming, the latest ATS blockers [37, 43, 56, 58] employ new features that are carefully engineered from the graph representation of a webpage. The hope is that it will be more difficult for an adversary to manipulate these new features to evade ATS blocking.

For instance, AdGraph [37] constructs a graph representation of a webpage. Graph nodes represent HTML elements or network requests, and edges encode the order in which a webpage loads these elements, such as parent-child relationships between the HTML elements or how the webpage initiates network requests (e.g., iframes, images, styles, and JavaScript code). For each network request node, AdGraph extracts structural and content features and trains a random forest classifier using the extracted features. During inference, the trained classifier identifies whether a given network request is related to ATS or not. PageGraph [58] improved upon AdGraph by leveraging both perceptual features extracted from an image classifier and contextual features extracted from the webpage's graph representation.

WebGraph [56] proposed using additional information flow features that encode the dynamic behavior of a webpage (e.g., reading/writing cookies), providing improved robustness to evasion attacks that manipulate structural and content features, such as mutating the graph representation of a webpage or the request URLs. Lee *et al.* [43] carefully analyzed the effectiveness of 883 features proposed in prior works [37, 56, 64] and selected 27 features for superior ATS classification over AdGraph and WebGraph. These approaches are highly effective, attaining high accuracy (92–96%) in ATS blocking.

2.2. (Universal) Adversarial Attacks

Deep neural networks (DNNs) are known to be sensitive to small input changes [60]. Numerous studies [15, 22, 23, 41, 45, 52, 60] have demonstrated that an adversary can exploit this sensitivity to fool a model's decision by adding human-imperceptible perturbations to a test-time sample. The objective of the adversarial example-crafting process is to find the human-imperceptible perturbation δ that, when added to the test-time sample x of an adversary's interest, maximizes the loss with respect to the correct label, formally as follows:

$$\arg\max_{|\delta| \leq \varepsilon} \mathcal{L}(\theta, x + \delta, y), \tag{1}$$

where θ is the model parameters of a target model f, y is the true label, $\mathcal{L}$ is the loss function (typically, the cross-entropy loss), and ϵ is the maximum perturbation magnitude.

Many optimization approaches have been proposed to find such δ [22, 45]. One representative approach is projected gradient descent (PGD) [45]. In PGD, the attacker crafts an adversarial example (i.e., $x + \delta$) as follows:

$$x^{t+1} = \Pi_\epsilon\Big(x^t + \alpha \cdot \mathtt{sign}\Big(\nabla_x \mathcal{L}\big(\theta, x^t, y\big)\Big)\Big), \qquad (2)$$

where x^t is the adversarial example generated at step t, α is the step size, and Π_ϵ is the projection operator that enforces the constraints on the perturbation size by clipping δ so that it falls within the ℓ_p ball around the original input x. We take the intuition from these studies and utilize PGD in our adversarial example-crafting process.

Universal adversarial perturbation (UAP). Moosavi-Dezfooli *et al.* introduced the concept of UAP [49]: an adversary crafts a *single* perturbation δ that can fool the target model's classification when added to *multiple* test-time samples. UAP is useful when a defender wants to audit whether a target classifier has a common vulnerability that an adversary can exploit to launch large-scale adversarial attacks. In particular, an adversary certainly prioritizes UAP over per-sample adversarial attacks in web-specific settings.

Unfortunately, it has *not* yet been shown whether an adversary can craft a universal manipulation of HTML elements that allows multiple ads to bypass ATS blockers. Prior work closest to ours is the UAP against perceptual ad-blockers, presented by Tramèr *et al.* [61]. However, this involved a straightforward adaptation of an existing UAP in image domains to perceptual ad-blocking, which is not applicable to crafting a UAP against state-of-the-art ATS blockers that leverage categorical and binary features. Per-sample adversarial attacks have been demonstrated against a single ATS blocker [65], while it still remains unknown whether other ATS blockers have a common vulnerability. Our work bridges this gap between the two lines of work by providing an auditing framework (YOPO) and exploitation in practical web settings.

3. Problem Statement

Recent advances in ATS blockers using ML have motivated ATS providers and publishers to seek ways of bypassing these blockers, thus securing their monetization channels. A few studies [43, 56, 65] have evaluated the robustness of recently proposed ATS blockers by performing adversarial attacks. However, we argue that these studies have not evaluated their robustness to the fullest extent in terms of the adversary's capability and scalability. Siby *et al.* [56] assume that the adversary is unable to manipulate flow features. Lee *et al.* [43] only evaluate whether applying random URL manipulations or JS obfuscation can bypass ATS blockers.

Zhu *et al.* [65] propose A^4 that generates adversarial examples in a per-sample manner, which is not scalable to real-world websites. In our preliminary study using 2,000 ATS network request nodes randomly sampled from Tranco's Top-10K websites, A^4 took 13.91 minutes on average to compute adversarial perturbation against one ATS-related request node and to reflect the computed perturbation onto the webpage. Note that A^4 is designed to optimize adversarial perturbation against a single network request node; thus, the computed perturbation cannot alter the classification result when applied to other request nodes. Considering that a webpage usually includes multiple ATS-related web resources (e.g., 8.60 ATS requests per webpage in our dataset) and that abusive ATS providers aim to deploy their service at scale, we believe that such an instance-specific attack is impractical and resource-intensive.

As an alternative, UAP is applicable to generating a single perturbation that enables ATS resources to be misclassified as non-ATS ones when applied to multiple network request nodes. In this regard, we seek to answer research questions regarding the generation of a single, cost-effective perturbation applicable to multiple websites. To the best of our knowledge, no previous study has investigated the feasibility of computing UAPs for either ATS classifiers or the critical threat that such perturbations pose.

3.1. Threat Model

Goal. We consider an adversary who crafts a UAP that enables multiple ads (or tracking services) to evade a target ATS blocker. These adversaries could be ATS providers—or even publishers—who are incentivized to evade the ATS blocker. Upon successful evasion, the attacker can render ads on multiple websites in their advertising networks. These attackers are also motivated to minimize their perturbations. By introducing such negligible changes, they can preserve the original functionalities as well as the visual elements of a target webpage [56].

Capabilities. Our adversary has *black-box* access to a target ATS blocker. We assume that the attacker does not know the ML algorithm that the ATS blocker employs or its training data. Because ATS blockers often utilize models that only provide hard labels, we assume that the attacker only has access to the hard labels of test-time instances, rendering the challenging *label-only* attack scenario [25, 35].

Our attacker also has query access to the target ATS blocker and uses this interface to construct data for training a surrogate model. The attacker uses (or alters) ATS resources available from the Internet for data construction. Note that we consider an ATS provider who colludes with an ATS publisher as our adversaries. These adversaries can manipulate all structural, content, flow, and JavaScript (JS) features in Table 1. For instance, since the URLs of ATS request nodes are under the ATS provider's control, the provider is able to modify content features, such as URL length or the presence of ad-related keywords in the URL. However, we do not allow our attacker to change the domain name to the first-party domain. Although such manipulation significantly eases bypassing ATS blockers, it requires CNAME cloaking [29], which is typically beyond the ATS provider's control; the number of domains using CNAME cloaking

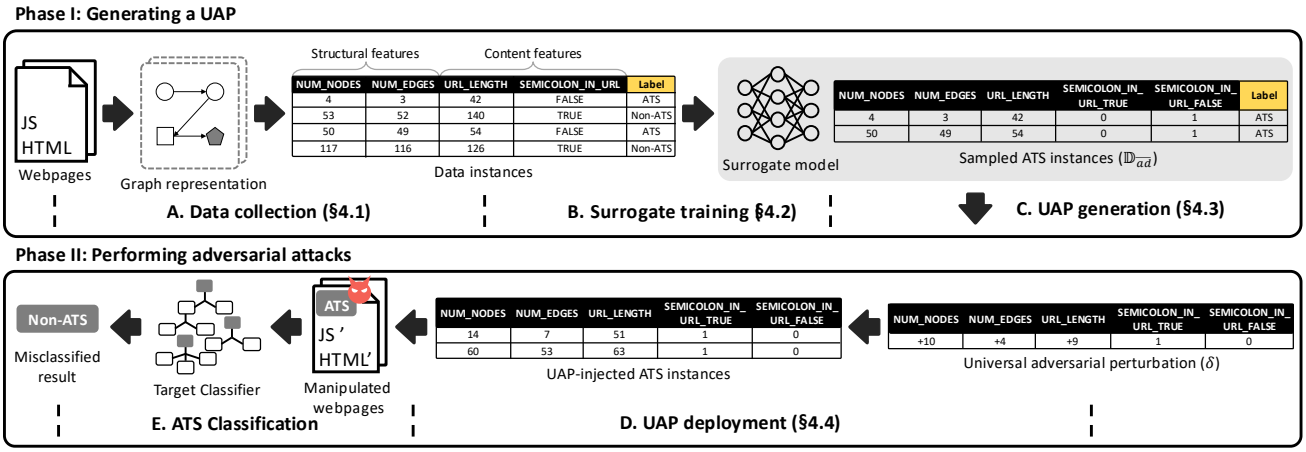


Figure 1: Illustration that shows how YOPO generates UAPs against a target ATS blocker.

accounted for only 0.58% of the Alexa Top-300K websites collected in 2020 [27]. To manipulate flow and JS features, the ATS provider can append a JS snippet that affects such features when providing ATS-related JS code.

3.2. Challenges in Crafting UAPs

We lay out three key challenges in generating UAPs.

Perturbing binary or categorical features. Many ATS blockers [36, 50, 56] leverage different types of features, such as binary and categorical features. Conventional adversarial attacks assume that the features they are perturbing are numerical, which allows a straightforward adoption of backward propagation for computing adversarial examples. However, such adversarial example-crafting techniques do not apply to discrete features, such as binary and categorical ones. To manipulate discrete features, we need a set of unique mapping functions that define how an adversary will reflect the numerical changes in feature values computed using our optimization approach to the actual binary and categorical feature values.

Injection cost. Our attacker needs to inject the perturbation that our framework generates; however, for some features, their manipulation, such as changing HTML tags or removing HTML elements, can impact users' browsing experience. If the attack affects the user experience or disrupts the original website's functionalities, this also reduces the adversary's revenue due to users leaving their websites [17, 28]. To minimize such shortcomings, it is important to consider how an adversary will prioritize perturbing each feature in UAP crafting. This prioritization will be performed based on a predefined cost model. We design a cost model based on how visually imperceptible a feature's perturbation is and how much the perturbation disrupts the original website's functionality.

Manipulating HTML elements. Ensuring that the target website correctly reflects the numerical perturbations that our framework computes poses a distinct challenge. Otherwise, our attack may not bypass the target ATS blocker when applied to the target website. It is also important for the attacker to achieve stealthiness as a victim may employ human curators who can detect and remove visually abnormal HTML manipulations before hosting [53]. We thus need an HTML manipulator that ensures our UAP is inconspicuous at the HTML level while retaining its effectiveness.

4. You Only Perturb Once

We present YOPO, a framework for auditing the vulnerability of a target ATS blocker to universal adversarial attacks. Figure 1 shows the YOPO workflow. Given a target ATS blocker, YOPO conducts Phase I, a one-time preparation step that crafts a UAP. In Phase II, YOPO applies the UAP to ATS requests on a given webpage.

Phase I: Generating a UAP. YOPO constructs training data for a surrogate model (§4.1), trains the surrogate on the constructed data (§4.2), and crafts a UAP (§4.3). Specifically, the framework starts by collecting websites by crawling the Internet. It then transforms these websites into the format used by a target classifier and feeds them to collect the classifier's predictions (i.e., hard labels). YOPO trains a surrogate model on the collected data. Finally, the framework crafts a UAP based on a subset of collected ATS samples. In crafting, YOPO leverages the cost model that we define in our objective function so that the adversary can minimize the cost of deploying the UAP.

Phase II: Performing adversarial attacks. Now, the adversary conducts (universal) adversarial attacks for each network request node responsible for fetching ATS resources on a publisher's webpage (§4.4). YOPO starts by feeding each ATS request node into YOPO along with the webpage. To each request on the webpage, YOPO applies the UAP by altering the DOM structure around the request. When this request fetches a JS file, YOPO also supports adding a JS snippet that introduces feature changes regarding the request. This manipulation step modifies the website's loading context of ad resources, thus enabling it to bypass the target ATS blocker.

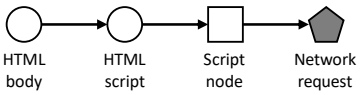
4.1. Generating Training Data for the Surrogate

YOPO's first step is to construct a dataset for training a surrogate model. Given a set of URLs, YOPO visits each one and converts the webpage into a set of features. In our attacks against AdGraph, WebGraph, AdFlush, and PageGraph, we first transform the webpage into a graph representation that captures the hierarchical loading context of various HTML/JS elements. To perform this conversion, YOPO leverages the revised Chromium binary [37] for AdGraph. It also uses OpenWPM [31] for WebGraph and AdFlush. For PageGraph, the native Brave browser supports this conversion process [6, 19].

Consider the following HTML snippet as an example:

```
<body>
  <script src="http://assets.adnetwork.com/js/ad_30x25.js">
</body>
```

This HTML snippet initiates a network request that fetches JS code from an ad network. Given this snippet, AdGraph generates the following graph representation:



The first and second HTML nodes of this representation correspond to the <body> and the <script> tags in the HTML snippet, respectively. The <script> tag then creates a script node, and it finally appends a network request node.

YOPO identifies such network request nodes (i.e., vertices) within each graph representation. These network request nodes are from image, script, and iframe HTML elements that fetch remote resources. For each request node, YOPO extracts features, which serve as an input of the target classifier. For instance, from the example above, YOPO extracts the following features (a subset of AdGraph's features):

NUM_NODES	NUM_EDGES	URL_LENGTH	SEMICOLON_IN_URL
4	3	42	FALSE

The first two structural features represent the number of nodes and edges in the entire graph, indicating that this graph has four nodes and three edges. The last two content features indicate that the URL length of this network request is 42, and the URL does not have a semicolon. YOPO extracts 65, 75, 27, and 15 features for AdGraph, WebGraph, AdFlush, and PageGraph, respectively.

Using these features, YOPO composes a query instance, queries this instance to the target model, and obtains its hard label, as shown in the following example:

NUM_NODES	NUM_EDGES	URL_LENGTH	SEMICOLON_IN_URL	Label
4	3	42	FALSE	ATS

Note that YOPO only requires black-box access to the target classifier (as discussed in §3.1). Against this black-box adversary with query access, ATS blockers may employ defense strategies (e.g., rate-limiting). However, most ATS blockers are offered as client-side browser extensions, which allows the attacker to send an arbitrary (or even unlimited) number of queries to them [3, 11, 13].

4.2. Training the Surrogate Model

The next step is to train a surrogate model on the training data collected in §4.1. We formally define this process as follows: Given a target classifier $f(\theta, x) \rightarrow \hat{y}$ (e.g., a random forest model) that produces a prediction vector $\hat{y}$ for a given query instance x, where θ represents the weight parameters of f. With black-box access to this target classifier, YOPO trains a surrogate model $f'(\theta', x') \rightarrow \hat{y'}$ that replicates the decision boundary of f as closely as possible. This step is similar to that in model extraction attacks [61], known to perform well with a model's confidence values. However, we are in a more challenging setting in which the adversary only has access to hard labels.

Through empirical trials, we find that a deep neural network (DNN) serves as a good surrogate. It achieves the highest attack success rates among other surrogates and enables the efficient computation of gradients in the UAP crafting process. Specifically, we use a four-layer feed-forward network for the architecture of our surrogate model.

The input vector x of f often includes binary and categorical features; thus, we cannot directly optimize across these discrete values. To ease the optimization process, we expand the dimension of the original input vector x when constructing the surrogate model f' by using one-hot encoding. Each binary feature thus becomes a two-dimensional vector, and each categorical feature becomes a k-dimensional vector, where k equals the number of its possible categories. For example, since the SEMICOLON_IN_URL feature is a binary feature, we expand this feature's dimension by splitting it into the SEMICOLON_IN_URL_TRUE and SEMICOLON_IN_URL_FALSE features, as follows:

NUM_NODES	NUM_EDGES	URL_LENGTH	SEMICOLON_IN_URL_TRUE	SEMICOLON_IN_URL_FALSE	Label
4	3	42	0	1	ATS

4.3. Crafting UAPs

YOPO computes a UAP δ, which induces the misclassification of multiple ATS request nodes into non-ATS requests against the surrogate model f'. For this, YOPO samples ATS data instances, compiling a set $\mathbb{D}_{\overline{ad}}$. Our framework optimizes δ across $\mathbb{D}_{\overline{ad}}$.

Objective function. The goal of the adversary is two-fold: (1) a UAP δ should elicit the classification of a given ATS request as non-ATS, and (2) δ should incur the minimum cost in introducing it to the target webpage. To achieve this goal, we define the objective function $F(\delta)$ and search for δ that minimizes this objective function while enforcing the feature constraint (i.e., $||\delta||_\infty \leq \epsilon$), as shown in Eq 3.

$$\min_{\delta} F(\delta) \; subject \; to \; ||\delta||_\infty \leq \epsilon \tag{3}$$

In Eq 4, the objective function F has two terms, each reflecting one of the adversary's two goals. λ is a hyper-parameter that strikes a balance between the two terms. The first term refers to an adversarial loss: the expected risk of cross entropy loss (i.e., $\mathcal{L}_{CE}$) between the model's prediction for each perturbed input $f'(\theta, x + \delta)$ and its

TABLE 1: List of features that YOPO perturbs and their perturbation injection costs against four ATS blockers: ADGRAPH (AG), WEBGRAPH (WG), ADFLUSH (AF), and PageGraph (PG). ✓ and ✗ denote whether each ATS blocker employs the corresponding feature for ATS classification.

Feature Type	Category	Perturbation	AG	WG	AF	PG	Assigned Cost
Structural	Numerical	Increase the graph size (NUM_NODES, NUM_EDGES)	✓	✓	✗	✗	0.1
		Increase the node degree (NUM_OUT, NUM_IN_OUT)	✓	✓	✗	✓	0.1
		Increase the inbound node degree (NUM_IN)	✗	✗	✗	✓	0.1
		Increase the number of siblings (NUM_SIBLINGS, NUM_PARENT_SIBLINGS)	✓	✗	✗	✗	0.1
		Increase the parent's node degree (NUM_PARENT_IN, NUM_PARENT_OUT, NUM_PARENT_IN_OUT)	✓	✗	✗	✓	2
		Increase / Decrease the average degree connectivity (AVG_DEGREE)	✓	✓	✗	✓	3
	Binary	Modify the attribute using a script (MOD_BY_SCRIPT)	✗	✗	✗	✓	2
		Add / Remove the parent's attribute (PARENT_ASYNC, PARENT_DEFER)	✓	✗	✗	✗	2
		Add the parent's attribute using a script (PARENT_ADD_BY_SCRIPT)	✓	✗	✗	✗	2
		Modify the parent's attribute using a script (PARENT_MOD_BY_SCRIPT)	✓	✗	✗	✓	2
		Remove the ascendant nodes' ad-related keywords (REMOVE_ASCENDANT_AD_KEYWORD)	✓	✓	✗	✗	3
		Remove the parent's sibling nodes' ad-related keywords (REMOVE_SIBLING_AD_KEYWORD)	✓	✗	✗	✗	3
	Categorical	Modify the first parent's tag name (PARENT_TAG_NAME)	✓	✗	✗	✗	3
Content	Numerical	Increase / Decrease the URL length (URL_LENGTH)	✓	✓	✓	✓	0.2
	Binary	Add / Remove a semicolon in the URL (SEMICOLON_IN_URL)	✓	✓	✗	✓	1
		Add / Remove a base domain in the URL query string (DOMAIN_IN_QS)	✓	✓	✗	✗	1
		Add / Remove screen dimension keywords in the URL query string (SCREEN_IN_QS)	✓	✓	✗	✗	2
		Make the URL valid / invalid (VALID_URL)	✓	✓	✗	✗	2
		Add / Remove ad-related keywords (e.g., 'ads' and 'banner') in the URL (AD_KEYWORD_IN_URL)	✓	✓	✓	✗	3
		Add / Remove special characters (e.g., '&', '/', and '=') in the URL (SPECIAL_CHAR_IN_URL)	✓	✓	✗	✗	3
		Add / Remove size keywords (SIZE_KEYWORD_IN_URL, SIZE_KEYWORD_IN_QS)	✓	✓	✗	✗	3
Flow	Numerical	Increase the flow graph's node degree (FLOW_OUT)	✗	✓	✗	✗	1
		Increase the number of sent requests (NUM_SENT_REQUESTS)	✗	✓	✓	✗	1
		Increase the number of received requests (NUM_RECEIVED_REQUESTS)	✗	✓	✗	✗	1
		Increase the flow graph's average node degree (AVG_FLOW_IN, AVG_FLOW_OUT)	✗	✓	✗	✗	1
		Increase the number of set cookie (NUM_SET_COOKIE)	✗	✓	✗	✗	2
		Increase the number of get cookie (NUM_GET_COOKIE)	✗	✓	✓	✗	2
		Increase the number of accesses to the storage (NUM_SET_STORAGE, NUM_GET_STORAGE)	✗	✓	✓	✗	2
JS	Numerical	Increase the frequency of 3-grams (3GRAM_FREQUENCY)	✗	✗	✓	✗	1
		Increase / Decrease the ratio of bracket notations to dot notations (RATIO_BRACKET_DOT)	✗	✗	✓	✗	1
		Increase / Decrease the average identifier length (AVG_ID_LENGTH)	✗	✗	✓	✗	2
		Increase / Decrease the number of average characters per line (AVG_CHARS_PER_LINE)	✗	✗	✓	✗	2

original label t_{ad} (i.e., ATS) across $\mathbb{D}_{\overline{ad}}$. That is, this term, $\mathbb{E}_{(x,y)\sim\mathbb{D}_{\overline{ad}}}[\mathcal{L}_{CE}(\cdot)]$, computes the average cross-entropy loss over data points (x, y) in $\mathbb{D}_{\overline{ad}}$. The second term computes the total injection cost of introducing δ to a webpage. Each element of our injection cost vector C defines the cost needed to manipulate each feature. Then, the total cost needed for injecting δ becomes the dot product of two vectors: the UAP δ and the cost vector C.

$$F(\delta) = -\lambda\ \mathbb{E}_{(x,y)\sim\mathbb{D}_{\overline{ad}}}[\mathcal{L}_{CE}\big(f'(\theta, x+\delta), t_{ad}\big)] + \delta \cdot C \quad (4)$$

Table 1 outlines the default injection cost vector used in our implementation. For each feature, we assign a numeric cost value that represents a relative risk of manipulating it. This relative risk reflects the likelihood that perturbing the corresponding feature could disrupt the webpage's functionality or compromise the stealthiness of the UAP.

The adversary is able to avoid perturbing specific features depending on the adversary's preferences by assigning higher costs to them. For instance, modifying the parent node of an ATS request could impact its sibling nodes, potentially disrupting their functionalities. Therefore, the adversary may want to assign higher costs (e.g., 2 or 3) to the features involving parent nodes to avoid such disruptions. The default cost (DC) model in Table 1 is developed from the adversary's perspective by two authors, considering the objectives aforementioned.

By minimizing F (Eq 4), YOPO finds a UAP δ that incurs a minimal injection cost while altering the classification of ATS to non-ATS when added to the original website. Note that YOPO crafts δ using the dataset sampled from $\mathbb{D}_{\overline{ad}}$, but the resulting UAP is effective on other data instances as well, not just those within $\mathbb{D}_{\overline{ad}}$. Moreover, YOPO can generate an effective UAP with only 40 samples, which are trivial for the adversary to collect (see Appendix C).

PGD-based optimization. YOPO runs the PGD optimization on the objective function F as follows:

$$\delta^{t+1} = \Pi\Big(\delta^t - \alpha \texttt{sign}\Big(\nabla_{\delta^t} F\big(\delta^t\big)\Big)\Big) \quad (5)$$

We perform gradient descent on $F\big(\delta^t\big)$, where δ^t is a UAP in the current optimization step t, and α is a step size. It

then updates the UAP and projects it onto the constrained space to enforce the feature constraints.

Note that YOPO only computes feature updates on those that our HTML manipulator addresses (see §4.4). For features that our HTML manipulator cannot address, YOPO assigns a fixed value of 0 to the corresponding field in δ and does not update them in the optimization process. YOPO perturbs 26, 24, 15, and 11 features out of 65, 75, 27, and 15 when attacking AdGraph, WebGraph, AdFlush, and PageGraph, respectively.

The features that YOPO covers in generating a perturbation are categorized into three types: numerical, binary, and categorical features. We now explain how we enforce constraints for each type of feature:

- **Numerical features.** YOPO obtains the minimum and maximum values from $\mathbb{D}_{\overline{ad}}$ and computes the difference d for each feature. YOPO then clips the updated perturbation value to ensure it falls within $[-d, d]$. At the same time, YOPO uses the parameter ϵ as a knob to control the maximum magnitude of the feature update in δ, limiting the perturbation value within $[-\epsilon, \epsilon]$ or $[0, \epsilon]$. Note that YOPO prioritizes the latter constraint on features highly likely to disrupt the functionality of a webpage if the attacker decreases the value. For instance, when ϵ is 10, YOPO enforces the constraint as follows:

NUM_NODES	NUM_EDGES	URL_LENGTH
+25	+4	+9

→

NUM_NODES	NUM_EDGES	URL_LENGTH
+10	+4	+9

  In this example, the original perturbation computed for the NUM_NODES feature indicates appending 25 nodes to the graph. However, since the constraint for this numerical feature is $[0, 10]$, YOPO clips the value, limiting the perturbation to appending ten nodes.
- **Binary features.** YOPO employs a one-hot vector representation (i.e., a two-dimensional vector) for each binary feature. Our framework feeds each binary feature vector into the SoftMax function in the projection step, transforming it into the probability that a data instance will belong to each class. When applying the binary feature to a webpage, our framework selects the class with a higher probability value.

SEMICOLON_IN_URL_TRUE	SEMICOLON_IN_URL_FALSE
0.9772	0.0228

→

SEMICOLON_IN_URL_TRUE	SEMICOLON_IN_URL_FALSE
1	0

  Consider the left-side figure as an example output of a SoftMax function for the two-dimensional vector of the SEMICOLON_IN_URL feature. In this case, YOPO takes the left-side feature with the higher value, which indicates adding a semicolon to the URL.
- **Categorical features.** YOPO takes the same approach as it does for binary features; the only difference is that YOPO expands the dimension of categorical features to the number of their possible categories instead of two.

4.4. HTML/JS Manipulation

In Phase II, YOPO performs adversarial attacks by mapping each feature update in the computed UAP back to the adversary's chosen webpage. We significantly expand an existing HTML manipulator [65] to include flow and JS features and to support more structural and content features. Note that none of the prior studies [43, 56, 65] consider flow and JS features when conducting adversarial attacks. The design objective of our HTML/JS manipulator is to correctly reflect the UAP on a webpage while preserving the original functionalities and visibility of the webpage. It also aims to keep the changes introduced to the webpage stealthy at the HTML level. Note that when applying a single UAP to each ATS request node, the altered form of the request varies depending on the surrounding context of the ATS request. We summarize the features that YOPO perturbs in Table 1.

Applying structural features. Removing HTML elements can disrupt the webpage layout and functionality. Therefore, our HTML manipulator only adopts the strategy of adding new HTML elements rather than removing existing ones when perturbing structural features. Note that this approach ensures that the modified webpage remains functional and is correctly rendered. For example, when the UAP indicates an increase in the NUM_SIBLINGS feature, the HTML manipulator adds sibling nodes using HTML tags with the "hidden" attribute, effectively introducing new elements without compromising the webpage's visual elements or functionality.

Note that A^4 [65] takes a similar approach to increasing structural feature values. In our experiment, A^4 inserted 547 <p> tags with the hidden attribute to increase the number of nodes (see Appendix A.1). However, repeatedly adding the same tags can be easily filtered by the defender [53]. To avoid this, we randomly sample existing HTML elements from the same webpage and append them as sibling nodes, thus rendering them indistinguishable from other benign HTML elements. Specifically, we sample HTML elements that have <p>, , and tags, as adding these tags does not incur any side effects (e.g., triggering outgoing requests or involving JS execution). We assign the hidden attribute to these tags when injecting them into the webpage to render them invisible to users.

When the HTML manipulator changes the parent node to a specific tag to reflect perturbations in the categorical feature, such as PARENT_TAG_NAME, YOPO wraps the current node with the specified tag. In this way, we maintain the overall structure and functionality of the webpage while reflecting the changes that the UAP requires.

Applying content features. Note that all perturbable content features are associated with modifying the target URL of a given ATS request node. The HTML manipulator reflects these changes by modifying the URL.

When the UAP increases the URL length, the HTML manipulator randomly extracts query strings from the URLs of other network request nodes on the same webpage and then appends them to the target URL. This increases the stealthiness of the manipulated URL. For example, to apply the UAP to the URL_LENGTH and SEMICOLON_IN_URL features, YOPO modifies the URL as follows:

```
<body>
  <script src="http://assets.adnetwork.com/js/ad_30x25.js
    ↪ #ver3.15;">
</body>
```

YOPO appended nine characters to the URL. These characters include (1) a # symbol, which makes the appended characters a URI fragment, (2) `ver3.15`, extracted from a query string randomly sampled from URLs on the same page, and (3) a semicolon. On the other hand, when the UAP requires a decrease in the URL length, the HTML manipulator selectively deletes URL elements while ensuring that the domain name remains unchanged.

Applying flow features. Flow features describe the information flow in a webpage, such as the number of read-/write accesses to cookies. Since these flow features capture the dynamic behavior of a webpage, static manipulation of the HTML elements cannot alter the information flow features of a target network request node. To manipulate flow features, we assume that an abusive ATS provider can alter ATS-related JS files, thereby introducing arbitrary information flows (e.g., reading/writing cookies).

Specifically, to perturb the flow feature of an ATS request fetching a JS file, YOPO appends a JS snippet that introduces additional information flows. For example, when the UAP requires an increase in the number of cookie access, it appends a JS code snippet, such as `document.cookie`.

Applying JS features. YOPO supports perturbing nine features extracted from JS code, such as `AVG_ID_LENGTH` or `AVG_CHARS_PER_LINE`. YOPO manipulates these features by adding JS snippets. For example, when a UAP demands a decrease in the feature values of `AVG_ID_LENGTH` and `AVG_CHARS_PER_LINE`, YOPO appends JS statements that declare short variables and empty lines, respectively.

YOPO also supports manipulating the 3-gram frequencies extracted from JS abstract syntax trees, which AdFlush leverages for ATS classification. Note that when AdFlush classifies a request fetching a JS file, it fetches this JS file, parses it into an abstract syntax tree, and then extracts 3-grams of node types by traversing the tree in a depth-first order. AdFlush then extracts the frequencies of the extracted 3-grams for an input feature. In YOPO, to increase the frequency of the 3-grams (e.g., *Statement*, *CatchClause*, and *Statement*) that the computed UAP indicates to manipulate, YOPO appends JS snippets that match the 3-grams that the UAP demands to change. For this, we prepare six JS snippets, each of which corresponds to a 3-gram that AdFlush leverages for its classification (see Appendix G).

YOPO has significantly expanded its feature support compared to that of A^4. While A^4 was designed to support only 19 features from AdGraph, YOPO supports 46 features from AdGraph, WebGraph, AdFlush, and PageGraph, covering information flow and JS code features. We also note that YOPO supports injecting HTML/JS code in a stealthy manner, seamlessly blending the injected code into the target webpage, which A^4 has not considered.

5. Evaluation

We audit the robustness of existing ATS blockers to our universal adversarial attacks. We implement YOPO in 10K+ LoC using Python 3.8 and PyTorch 1.13.0. We also employ a man-in-the-middle proxy to monitor requests to target websites and redirect the responses to our local server to test the efficacy of the UAPs that YOPO generates. We release our code at https://github.com/WSP-LAB/YOPO.

5.1. Experimental Setup

All our experiments are conducted on a machine equipped with two Intel Xeon Gold 6348 (2.7 GHz) CPUs, four NVIDIA RTX 3090 GPUs, and 768 GB of DRAM. The machine runs Ubuntu 22.04 (64-bit).

Datasets. To construct our datasets, we crawl 7,907 webpages from Tranco's Top-10K websites as of December 2023. We transform each webpage into the graph representations used by our target ATS blockers. We extract the features for each network request node. We then label each data instance as ATS or non-ATS based on the recipient domain of the corresponding network request. When the domain matches any of the eight ad blocklists [1, 4, 5, 7]–[10, 12], its data instance is labeled as ATS; otherwise, it is labeled as non-ATS. We compile 477,559 data instances for AdGraph, 459,995 data instances for WebGraph and AdFlush, and 405,270 data instances for PageGraph.

For each target ATS blocker, we randomly sample 100K data instances to train its surrogate model. We assign hard labels to those instances based on their classification results from the target ATS blocker. We then randomly select 40K instances, which are labeled as ATS and not used for training the surrogate model. We then use these 40K samples for $\mathbb{D}_{\overline{ad}}$ to generate a UAP. To test the efficacy of YOPO, we choose 2K target requests randomly. Note that these are all ATS requests and are not used for training the surrogate or generating a UAP. We do not include any ATS requests made after dynamic JS execution as target nodes because our HTML manipulator perturbs ATS requests before rendering occurs. In Appendix C, we also consider an adversary using a different number of samples for training a surrogate model and UAP generation.

Target ATS blockers. We select AdGraph [37], WebGraph [56], AdFlush [43], and PageGraph [58] as our target ATS blockers. Note that AdFlush offers a Chrome extension [42] to facilitate its wide deployment. Since PageGraph [58] only blocks ATS images, we re-implemented its classifier to support the classification of other types of ATS resources beyond images, such as iframes, styles, and JS code. To train this classifier, we excluded perceptual features obtained from image resources and retained the remaining features extracted from the PageGraph representation provided by the native Brave browser [6].

These ATS classifiers leverage a wide range of HTML and JS features from webpages, ranging from primitive to sophisticated features, including information flow features. These features serve as well-received raw input that follow-up ATS classification studies will first consider.

Note that the target ATS blockers (AdGraph, WebGraph, AdFlush, and PageGraph) use random forest classifiers, which we implemented using the scikit-learn library. To evaluate the performance of each classifier, we

TABLE 2: Reproduced (Ours) and reported performance of the four target ATS blockers: AdGraph (AG), WebGraph (WG), AdFlush (AF), and PageGraph (PG).

Target Model	Accuracy (%)		Precision (%)		Recall (%)	
	Ours	Reported	Ours	Reported	Ours	Reported
AG	92.64	95.33	89.90	89.10	85.30	86.60
WG	95.68	94.32	93.26	92.24	92.27	94.14
AF	95.93	98.00	95.34	98.00	90.87	96.00
PG	95.89	97.60	92.90	92.00	93.45	75.00

conducted stratified 10-fold cross-validation on the data instances aforementioned and reported the average performance metrics across the ten test folds. To determine the best spatial ratio for each data folder, we varied the ratio and selected the best-performing spatial ratio of ATS data instances, which was an ATS ratio of 30%. When training each model, we used all the features supported by the model to maximize its performance. Table 2 summarizes the performance of these target ATS blockers trained on our dataset, along with their reported performances from the respective papers [37, 43, 56, 58].

Surrogate models. Our architectural choice of the surrogate model for each ATS blocker is a four-layer feed-forward neural network. We train each surrogate model for 30 epochs using 100K randomly sampled data instances, without considering the spatial ratio when compiling these training instances. We assume that our black-box adversary trains the surrogate model with a different network architecture and different data instances labeled by the target model, resulting in performance differences compared to its original model. The accuracy of the surrogate model serves as a proxy for how closely it resembles the decision boundary of the target model; a higher accuracy indicates that a surrogate model is better for attacking. The surrogate model for AdGraph has an accuracy of 95.49%, and those for WebGraph, AdFlush, and PageGraph show accuracies of 85.36%, 87.38%, and 83.45%, respectively.

Metrics. We employ two metrics for evaluating our attacks: the attack success rate (ASR) and the total cost (TC) of UAP injection. To compute ASR, among the 2,000 target requests, we only perturbed ATS requests that the target classifier correctly identified as ATS. The ASR is the ratio of the perturbed ATS requests that the target classifier misclassifies as non-ATS to the total number of perturbed ATS requests. TC represents the average cost of reflecting a UAP across the 2K webpages that initiate the target requests. A higher ASR and a lower TC indicate better efficacy in generating effective UAPs.

Injection costs. YOPO requires a cost model (i.e., cost vector C) for each feature set, where we assign an abstract cost to each feature that represents the relative risk of manipulating it. To demonstrate the flexibility of YOPO in generating UAPs with different cost models, we define four cost models: default cost (DC), high manipulation costs for HTML structural features (HSC), high manipulation costs for content features (HCC), and high manipulation costs for JS features (HJC). DC maintains a balanced cost setting, while HSC assigns 10 times higher costs to changing structural features, directing YOPO to avoid perturbing HTML structural features. Similarly, HCC guides YOPO to avoid perturbing content features. When attacking AdFlush, we adopt the HJC cost model instead of HSC since AdFlush does not use structural features; HJC guides YOPO to avoid perturbing JS features. ϵ denotes the maximum perturbation value that YOPO applies to numerical features and does not affect the binary or categorical features, which only take values of 0 or 1.

TABLE 3: Attack success rates (ASRs) and total costs (TC) of YOPO against AdGraph (AG), WebGraph (WG), AdFlush (AF), and PageGraph (PG).

Target Model	Cost Type	ϵ = 5		ϵ = 10		ϵ = 20		ϵ = 40	
		ASR (%)	TC	ASR (%)	TC	ASR (%)	TC	ASR (%)	TC
AG	DC	87.70	32	89.27	43	90.38	65	92.27	89
	HSC	87.93	67	89.59	68	89.91	70	92.19	75
	HCC	77.68	73	80.44	92	80.20	124	82.26	165
WG	DC	67.03	36	71.21	57	71.50	89	71.48	150
	HSC	66.74	33	69.38	50	69.23	83	69.60	170
	HCC	67.11	89	67.91	105	67.77	128	68.35	186
AF	DC	59.71	11	61.91	19	61.69	34	60.73	65
	HJC	59.85	51	61.25	62	61.62	72	60.37	108
	HCC	57.06	21	58.24	27	58.46	41	58.38	72
PG†	DC	82.42	5	84.16	8	85.35	13	85.53	22
	HSC	71.43	10	76.10	11	81.50	13	82.51	17
	HCC	80.31	18	84.98	20	85.81	22	85.47	28
WG‡	DC	22.47	17	27.64	32	28.06	63	27.99	125

† PageGraph without perceptual features.
‡ WebGraph without content features.

5.2. Effectiveness of Our UAPs

Table 3 summarizes the effectiveness of our UAP against the target ATS blockers. Rows 3–5 show our attacks against AdGraph. We achieve high ASRs, ranging from 77.68% to 92.27% across the different ϵ values and the three cost models. Notably, YOPO achieved a 92.27% ASR against AdGraph when ϵ was set to 40 in the DC model. When applying the HCC and HSC models, the overall ASRs slightly decreased, as these cost models restrict the feature search space for finding UAPs. Despite these decreases in ASR, YOPO still generates effective UAPs, maintaining ASRs above 77.68%.

Rows 6–8 show that WebGraph is more robust than AdGraph. The ASRs vary between 66.74% and 71.50%. We attribute the reduction in ASR for WebGraph to its use of information flow features. Manipulating flow features requires additional effort than modifying HTML elements, such as redirecting the received request to other servers. Currently, YOPO only supports manipulating flow features

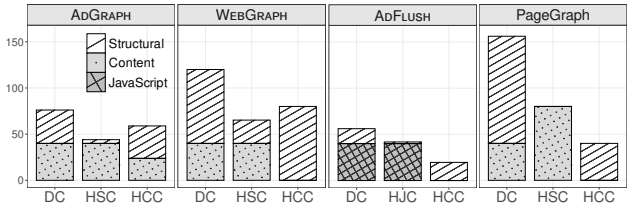


Figure 2: Comparison of perturbation magnitudes in UAPs' numerical features. UAPs are generated with ϵ of 40 using four injection cost models: DC, HSC, HCC, and HJC.

for network request nodes that fetch JS snippets. While YOPO only allows perturbing a subset of flow features, it attains a high ASR of up to 71.50% against WebGraph.

Rows 9–11 show our attack results against AdFlush. YOPO achieves ASRs of up to 61.91%, exhibiting the lowest ASR among the four ATS blockers. Note that AdFlush leverages character embedding features extracted from webpages (e.g., URLs and domain names). Perturbing these embedding features is technically challenging since directly perturbing embedding vectors and finding characters of which embeddings are similar to these updated embeddings often results in ineffective adversarial characters. However, YOPO still exhibits ASRs above 57%.

Rows 12–14 present the attack results against PageGraph. YOPO bypassed PageGraph with ASRs ranging from 71.43% to 85.53%, demonstrating its effectiveness. In Row 15, we also report YOPO's efficacy against WebGraph without content features. Although this version of WebGraph becomes more resistant to attacks, with a maximum ASR of 28.06%, it suffers from a 7.39% drop in classification accuracy [56], making it less practical for real-world deployments.

When comparing cost models, YOPO exhibits relatively higher ASRs with HSC (HJC for AdFlush) than HCC. This result provides an insight into the features that our target ATS blockers rely on: they depend more on the content features than on structural and JS features when making decisions. This finding highlights the importance of content-related features in ATS classification, which exacerbates the adversary's threat since URL-related content features are trivial to manipulate.

Per-sample adversarial attacks. In our experiment in §3, A^4 [65] achieved an ASR of 80.83% against AdGraph by conducting per-sample evasion attacks. We note that YOPO shows a higher ASR of 92.27% with only a single perturbation. We attribute this result to the fact that A^4 perturbs only 19 features, while YOPO exploits 26 features. We also emphasize that YOPO generates a UAP that can be applied to multiple ATS requests (refer to Appendix E). Moreover, YOPO takes 32.42 seconds for UAP generation, which is a one-time cost. Such small computational demands significantly reduce the resources attackers need, enabling them to conduct the attack at scale.

Injection cost comparison. Figure 2 shows the sum of numerical feature values in the generated UAP for three target ATS blockers under different injection cost models. For AdGraph, WebGraph, AdFlush, and PageGraph, we examine 11, 15, 10, and 8 numerical features in the UAP, respectively, and separate them into the content, structural, and JS feature groups. In HSC, YOPO introduces smaller changes in the numerical features associated with the HTML structures. This indicates that when manipulating structural features, YOPO is more conservative due to the higher cost associated with changing the underlying HTML structure. Similarly, in HJC, YOPO tries to generate smaller changes in the numerical features within the JS features. In contrast, in HCC, YOPO exhibits a tendency to introduce small changes in the numerical features within the content feature group. Overall, these results demonstrate the flexibility of YOPO in generating UAPs across different cost models and its ability to adapt a UAP based on the target features and their associated manipulation costs. In Appendix D, we further evaluate the impact of attacking ML models other than a random forest classifier.

5.3. Characterization of Vulnerability

We conduct an ablation study to investigate the impact of each perturbed feature on the ASRs against AdGraph, WebGraph, AdFlush, and PageGraph. For this, we choose a feature to nullify and compute the ASR difference between the original UAP and the UAP without that feature. We set ϵ to 10 for this experiment.

Results. Table 4 shows the top five most important features for a successful universal attack. We find that binary features play a crucial role in generating effective UAPs against all classifiers. For example, in AdGraph, binary features, such as removing/adding an *async/defer* attribute from the parent node, contribute the most to changing classification results. If the attacker cannot control one of the binary features, the ASR decreases by at most 23.44%. Perturbations to numerical features (e.g., increasing the URL length or increasing the 3-gram frequency in an AST) are also effective in successful attacks. Especially, in AdFlush, features regarding the frequencies of 3-grams in an AST play a crucial role. Three out of the top five most influential features are attributed to the frequencies of 3-gram sequences. In WebGraph, the top three most crucial features are all the content features. This result highlights that the vulnerability of WebGraph to our attacks stems from its dependence on content features in classification.

Recall that YOPO overwrites binary features when injecting the UAP into a target ATS instance (§4.3) and that the most influential features for attack success are mostly binary features (Table 4). Hence, we further investigate whether high ASRs (the target ad-blocker misclassifies the majority of ATS instances as non-ATS) occur when we overwrite these binary features. To validate our hypothesis, we analyze the distribution of these binary features in our dataset. We first select the top five most influential binary features contributing to the attack. For AdFlush and PageGraph, as YOPO only perturbs one and three binary features, respectively, we select those binary features. Then, for the selected binary features, we analyze network requests

TABLE 4: Top five most influential features in UAPs against AdGraph, WebGraph, AdFlush, and PageGraph.

Target	Description of Perturbation	Category	Feature type	ASR (↓)
AdGraph	Set PARENT_ASYNC to TRUE	Structural	Binary	-19.87%
	Set SEMICOLON_IN_URL to TRUE	Content	Binary	-8.28%
	Set PARENT_DEFER to FALSE	Structural	Binary	-5.83%
	Set DOMAIN_IN_QS to FALSE	Content	Binary	-5.52%
	Increase URL_LENGTH	Content	Numerical	-1.89%
WebGraph	Set AD_KEYWORD_IN_URL to FALSE	Content	Binary	-23.44%
	Set SIZE_KEYWORD_IN_URL to TRUE	Content	Binary	-10.48%
	Set SPECIAL_CHAR_IN_URL to FALSE	Content	Binary	-4.32%
	Increase NUM_SET_STORAGE	Flow	Numerical	-3.64%
	Increase NUM_RECEIVED_REQUESTS	Flow	Numerical	-3.08%
AdFlush	Increase 3GRAM_FREQUENCY (*Identifier*, *Expression*, *Identifier*)	JS	Numerical	-9.41%
	Increase 3GRAM_FREQUENCY (*Identifier*, *Expression*, *Literal*)	JS	Numerical	-8.53%
	Increase URL_LENGTH	Content	Numerical	-3.38%
	Set SPECIAL_CHAR_IN_URL to FALSE	Content	Binary	-3.16%
	Increase 3GRAM_FREQUENCY (*Expression*, *Identifier*, *Identifier*)	JS	Numerical	-1.4%
PageGraph	Set SEMICOLON_IN_URL to TRUE	Content	Binary	-12.64%
	Increase NUM_IN_OUT	Structural	Numerical	-9.89%
	Increase AVG_DEGREE	Structural	Numerical	-7.24%
	Set MOD_BY_SCRIPT to FALSE	Content	Binary	-5.13%
	Set PARENT_MOD_BY_SCRIPT to FALSE	Structural	Binary	-4.86%

that have a specific combination of values that the UAP overwrites with. 98.48%, 97.55%, 81.67%, and 75.71% of such network request nodes in our dataset are non-ATS nodes in AdGraph, WebGraph, AdFlush, and PageGraph, respectively. We believe that this imbalanced distribution causes the model to regard specific combinations of binary features as a strong sign of non-ATS instances. Consequently, the adversary is able to alter the classification of ATS instances from ATS to non-ATS by overwriting these binary features. We leverage this intuition to present potential countermeasures in §6.

5.4. Preserving Website Functionalities

YOPO manipulates HTML files when injecting the generated UAP into webpages. However, this HTML manipulation may affect the classification results of other non-ATS resources or break the original webpage functionality.

We analyze the unintended side effects caused by HTML manipulation. To this end, for each target ATS blocker, we measure the *collateral damage* [56] in 100 webpages filtered by that blocker. Collateral damage is the ratio of non-ATS nodes classified as ATS after the attack, quantifying the extent to which the ATS blockers block non-ATS resources after the attack. Greater collateral damage can negatively impact the victim's browsing experience.

Results. We observe the collateral damage of 2.97% ± 5.20% (median: 0.00), 1.51% ± 3.76% (median: 0.00), 2.97% ± 5.45% (median: 0.00), and 2.45% ± 4.41% (median: 0.00) in AdGraph, WebGraph, AdFlush, and PageGraph, respectively. In numbers, our attacks cause collateral damage in only 39, 26, 44, and 47 webpages in AdGraph, WebGraph, AdFlush, and PageGraph, respectively.

We further analyze whether collateral damage changes the original functionality of a webpage. Following prior work [56], we manually inspect the functionality of the 156 webpages with non-zero collateral damage. To conduct this analysis, we install an extension that blocks the URLs classified as ATS on the vanilla Firefox browser. We then load the original and manipulated versions of the webpage and compare whether YOPO introduces any functionality breakage. We find that only 14 webpages out of the 400 instances exhibit functionality disruption (e.g., sign-in functionality or apparent visual breakages). We further elaborate on the impact of applying the UAP generated by YOPO with a concrete example in §A.2.

6. Countermeasures

We now discuss potential countermeasures against our adversarial attacks. We initially evaluated the effectiveness of a standard countermeasure against adversarial input perturbations, *adversarial training* [45], which involves training ATS blockers with perturbed samples. However, we observed that this method is ineffective in reducing the attacker's success rate (see Appendix B). We thus introduce two alternative mitigation strategies to enhance the ATS blocker's robustness against our attacks.

We first examine whether making the ATS classifiers less sensitive to binary features is effective in decreasing the attacker's chance of performing successful attacks. We additionally present a strategy designed to mislead YOPO into computing perturbations that are difficult to implement in HTML. By combining these countermeasures, we propose a way of preprocessing input features that restricts the adversary's options on perturbable input features, thereby decreasing the likelihood of successful evasion attacks.

6.1. Nullifying Binary Features

Recall from §5.3 that the existing ATS blockers heavily rely on binary features due to their imbalanced distribution in the training set crawled from the Internet. YOPO exploits this imbalanced distribution to generate a UAP. Hence, we hypothesize that addressing this natural distribution imbalance will make the ATS blockers less dependent on these binary features, thereby enhancing the classifier's robustness.

Methodology. To this end, we manipulate the values of the top five most important binary features used by our target ATS blockers (recall §5.3). Specifically, we set these feature values to zero for all our training samples and train the AdGraph, WebGraph, AdFlush, and PageGraph models on this revised training set, ensuring that the resulting models do not use these feature values for classification. For

TABLE 5: Comparison of ASRs and performance of ATS blockers when applying our mitigation strategies.

Target Model	Mitigation	ASR	Accuracy	Precision	Recall
AdGraph	-	89.27	92.64	89.90	85.30
	#1	61.75	92.15	89.11	84.43
	#1 + #2	40.41	91.59	85.49	87.05
WebGraph	-	71.21	95.68	93.26	92.27
	#1	63.90	95.39	92.52	92.08
	#1 + #2	48.55	95.19	91.64	92.38
AdFlush	-	61.91	95.93	95.34	90.87
	#1	49.82	95.79	94.99	90.77
	#1 + #2	42.74	95.68	95.00	90.34
PageGraph	-	84.16	95.89	92.90	93.45
	#1	70.28	95.78	92.66	93.06
	#1 + #2	64.51	95.74	92.59	93.26

example, we set the value of the `PARENT_ASYNC` feature, which examines if the parent node has an *async* attribute, to zero.

Results. Table 5 summarizes the results of applying our mitigation strategy. The last three columns represent the ATS blocker's performance when applying the mitigation in the second column. The third column represents the ASR when conducting YOPO attacks against the corresponding model.

As the table shows, after applying the mitigation, all three classifiers achieve improved robustness against our UAP attacks. We successfully reduce the ASR of AdGraph, WebGraph, AdFlush, and PageGraph from 89.27%, 71.21%, 61.91%, and 84.16% to 61.75%, 63.90%, 49.82%, and 70.28%, respectively. Moreover, even if we remove the models' reliance on the top five binary features, we still preserve the performance of the ATS classifiers.

6.2. Misleading Perturbation Directions

Our second strategy involves misleading the optimization procedure of YOPO to generate UAPs that are difficult to implement in HTML. For this, we preprocess numerical input features and train an ATS blocker using these features. This preprocessing step induces our attack against the new ATS blocker to reflect decreases in numerical features within webpages. These perturbations entail removing HTML elements, which often results in functionality breakage or layout disruption. Consequently, this countermeasure restricts the adversary to perturbing only a limited set of features.

Methodology. For a set of numerical features, we manipulate the vast majority of feature values of which non-ATS samples have small values, thus intentionally introducing an imbalanced distribution into those features. Such feature engineering provides a strong signal to the target classifier that most non-ATS samples have small values for the selected features, while most ATS samples have large values. Consequently, this approach guides the UAP optimization process toward decreasing the selected features because the adversary seeks to alter the model's prediction for them to the non-ATS class.

However, preprocessing all numerical features for non-ATS instances to have small values certainly leads to a notable performance drop in ATS classification. Therefore, we select a subset of features among the numerical features that YOPO perturbs. Each feature in this subset meets the criterion where the proportion of non-ATS samples falling below a specific threshold exceeds that of ATS samples below the same threshold by at least 20%. Given that these features already have imbalanced distributions, with a significant portion of non-ATS samples having smaller feature values compared to those of ATS samples, there is room for preprocessing input features.

For instance, 54.56% (33.90%) of non-ATS (ATS) request nodes in our dataset have no siblings. We thus include the `NUM_SIBLINGS` feature in our feature set to preprocess. As a result, we choose five, eight, one, and two numerical features for AdGraph, WebGraph, AdFlush, and PageGraph, respectively. We then add an additional 10% of non-ATS samples that initially had values greater than the aforementioned threshold for these features to have random values ranging from zero to the threshold. After this preprocessing, the proportion of non-ATS samples falls below that of ATS samples by at least 30%. Finally, we train the ATS classifiers using these adjusted data instances.

Results. Table 5 shows the effectiveness of our mitigation. We consider a scenario where a defender applies both mitigation methods at the same time, expecting a further increase in the model's robustness against our attacks. Our approach results in an additional ASR drop of 21.34%, 15.35%, 7.08%, and 5.77% for AdGraph, WebGraph, AdFlush, and PageGraph, respectively, without notable performance decreases.

7. Conclusion

We present a novel auditing framework (YOPO) designed to generate a single adversarial perturbation cost-effectively. YOPO formulates UAP generation as an optimization problem, incorporates the cost of manipulating various features into the objective function, and optimizes a UAP on this objective function. We show that our attack bypasses the ATS blockers with a success rate of up to 92.27% without compromising the website's functionality or visual elements, demonstrating its critical threat. We also observe that imbalanced feature distribution leads to the common vulnerability of the ATS classifiers and discuss two feature engineering-based mitigation strategies.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work was supported by (1) the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2020-II200153) and (2) Korea Internet & Security Agency (KISA) grant funded by the PIPC (No. 2780000003).

References

[1] "Blockzilla," https://raw.githubusercontent.com/annon79/Blockzilla/master/Blockzilla.txt, 2016.

[2] "Who is stealing my power III: An adnetwork company case study," http://blog.netlab.360.com/who-is-stealing-my-power-iii-an-adnetwork-company-case-study-en/, 2018.

[3] "AdBlock," https://getadblock.com/, 2024.

[4] "Adblock Warning Removal List," https://easylist-downloads.adblockplus.org/antiadblockfilters.txt, 2024.

[5] "Anti-adblock killer," https://raw.githubusercontent.com/reek/anti-adblock-killer/master/anti-adblock-killer-filters.txt, 2024.

[6] "Brave browser," https://brave.com/, 2024.

[7] "EasyList," https://easylist.to/easylist/easylist.txt, 2024.

[8] "EasyPrivacy," https://easylist.to/easylist/easyprivacy.txt, 2024.

[9] "Fanboy's Annoyance List," https://easylist.to/easylist/fanboy-annoyance.txt, 2024.

[10] "Fanboy's Social Blocking List," https://easylist.to/easylist/fanboy-social.txt, 2024.

[11] "Ghostery," https://www.ghostery.com, 2024.

[12] "Peter Lowe," https://pgl.yoyo.org/adservers/serverlist.php?hostformat=adblockplus&mimetype=plaintext, 2024.

[13] "uBlock Origin," https://ublockorigin.com, 2024.

[14] AdblockPlus, "Sentinel is online," https://blog.adblockplus.org/blog/sentinel-is-online, 2018.

[15] A. Al-Dujaili and U.-M. O'Reilly, "Sign bits are all you need for black-box attacks," in *Proceedings of the International Conference on Learning Representations*, 2020.

[16] M. M. Ali, B. Chitale, M. Ghasemisharif, C. Kanich, N. Nikiforakis, and J. Polakis, "Navigating murky waters: Automated browser feature testing for uncovering tracking vectors," in *Proceedings of the Network and Distributed System Security Symposium*, 2023.

[17] D. An, "Find out how you stack up to new industry benchmarks for mobile page speed," https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/, 2018.

[18] G. Anthes, "Data brokers are watching you," *Communications of the ACM*, vol. 58, no. 1, pp. 28–30, 2015.

[19] B. browser, "Pagegraph," https://github.com/brave/brave-browser/wiki/PageGraph.

[20] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, "A survey on web tracking: Mechanisms, implications, and defenses," *Proceedings of the IEEE*, vol. 105, no. 8, pp. 1476–1510, 2017.

[21] A. Cahn, S. Alfeld, P. Barford, and S. Muthukrishnan, "An empirical study of web cookies," in *Proceedings of the International Conference on World Wide Web*, 2016.

[22] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2017, pp. 39–57.

[23] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 15–26.

[24] Q. Chen, P. Ilia, M. Polychronakis, and A. Kapravelos, "Cookie Swap Party: Abusing first-party cookies for web tracking," in *Proceedings of the Web Conference*, 2021, pp. 2117–2129.

[25] M. Cheng, S. Singh, P. H. Chen, P.-Y. Chen, S. Liu, and C.-J. Hsieh, "Sign-opt: A query-efficient hard-label adversarial attack," in *Proceedings of the International Conference on Learning Representations*, 2020.

[26] S. Dambra, I. Sanchez-Rola, L. Bilge, and D. Balzarotti, "When sally met trackers: Web tracking from the userśperspective," in *Proceedings of the USENIX Security Symposium*, 2022, pp. 2189–2206.

[27] H. Dao, J. Mazel, and K. Fukuda, "CNAME cloaking-based tracking on the web: Characterization, detection, and protection," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3873–3888, 2021.

[28] G. Data, "Anonymized google analytics data from a sample of mWeb sites opted into sharing benchmark data," https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/mobile-site-load-time-statistics/, 2016.

[29] Y. Dimova, G. Acar, L. Olejnik, W. Joosen, and T. V. Goethem, "The CNAME of the Game: Large-scale analysis of dns-based tracking evasion," in *Proceedings on Privacy Enhancing Technologies*, 2021, pp. 394–412.

[30] Z. A. Din, P. Tigas, S. T. King, and B. Livshits., "PERCIVAL: Making in-browser perceptual ad blocking practical with deep learning," in *Proceedings of the USENIX Annual Technical Conference*, 2020, pp. 387–400.

[31] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2016, pp. 1388–1401.

[32] K. Garimella, O. Kostakis, and M. Mathioudakis, "Ad-blocking: A study on performance, privacy and counter-measures," in *Proceedings of the Web Science Conference*, 2017, pp. 259–262.

[33] A. Gervais, A. Filios, V. Lenders, and S. Capkun, "Quantifying web adblocker privacy," in *Proceedings of the European Symposium on Research in Computer Security*, 2017, pp. 21–42.

[34] X. Hu and N. Sastry, "Characterising third party cookie usage in the EU after GDPR," in *Proceedings of the Web Science Conference*, 2019, pp. 137–141.

[35] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 2137–2146.

[36] U. Iqbal, Z. Shafiq, and Z. Qian, "The ad wars: Retrospective measurement and analysis of anti-adblock filter lists," in *Proceedings of the Internet Measurement Conference*, 2017, pp. 171–183.

[37] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq, "ADGRAPH: A graph-based approach to ad and tracker blocking," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2020, pp. 895–904.

[38] A. J. Kaizer and M. Gupta, "Towards automatic identification of javascript-oriented machine-based tracking," in *Proceedings of the ACM International Workshop on Security and Privacy Analytics*, 2015, pp. 33–40.

[39] V. Khrulkov and I. Oseledets, "Art of singular vectors and universal adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8562–8570.

[40] A. Klein and B. Pinkas, "Dns cache-based user tracking," in *Proceedings of the Network and Distributed System Security Symposium*, 2019.

[41] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *CoRR*, vol. abs/1607.02533, 2017.

[42] K. Lee, C. Lim, B. Jin, T. Kim, and H. Kim, "Adflush chrome extension," https://github.com/SKKU-SecLab/AdFlush.

[43] ——, "AdFlush: A real-world deployable machine learning solution for effective advertisement and web tracker prevention," in *Proceedings of the Web Conference*, 2024, pp. 1902–1913.

[44] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, "Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016," in *Proceedings of the USENIX Security Symposium*, 2016, pp. 997–1013.

[45] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proceedings of the International Conference on Learning Representations*, 2018.

[46] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. Weippl, "Block me if you can: A large-scale study of tracker-blocking tools," in *Proceedings of the IEEE European Symposium on Security and Privacy*, 2017, pp. 319–333.

[47] J. V. Monaco, "Device fingerprinting with peripheral timestamps," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2022, pp. 1018–1033.

[48] V. Moonsamy, M. Alazab, and L. Batten, "Towards an understanding of the impact of advertising on data leaks," *International Journal of Security and Networks*, vol. 7, no. 3, pp. 181–193, 2012.

[49] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1765–1773.

[50] S. Munir, S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, and C. Troncoso, "Cookiegraph: Understanding and detecting first-party tracking cookies," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2023, pp. 3490–3504.

[51] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless Monster: Exploring the ecosystem of web-based device fingerprinting," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2013, pp. 541–555.

[52] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings of the IEEE European Symposium on Security and Privacy*, 2016, pp. 372–387.

[53] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2020, pp. 1332–1349.

[54] D. Plohmann, K. Yakdan, M. Klatt, and J. Bader, "A comprehensive measurement study of domain generating malware," in *Proceedings of the USENIX Security Symposium*, 2016, pp. 263–278.

[55] E. Pujol, O. Hohlfeld, and A. Feldmann, "Annoyed users: Ads and ad-block usage in the wild," in *Proceedings of the Internet Measurement Conference*, 2015, pp. 93–106.

[56] S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, and C. Troncoso, "WebGraph: Capturing advertising and tracking information flows for robust blocking," in *Proceedings of the USENIX Security Symposium*, 2022, pp. 2875–2892.

[57] C. Sitawarin, F. Tramèr, and N. Carlini, "Preprocessors matter! realistic decision-based attacks on machine learning systems," in *Proceedings of the International Conference on Machine Learning*, 2023.

[58] A. Sjösten, P. Snyder, A. Pastor, P. Papadopoulos, and B. Livshits, "Filter list generation for underserved regions," in *Proceedings of the Web Conference*, 2020.

[59] K. Solomos, J. Kristoff, C. Kanich, and J. Polakis, "Tales of favicons and caches: Persistent tracking in modern browsers," in *Proceedings of the Network and Distributed System Security Symposium*, 2021.

[60] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of the International Conference on Learning Representations*, 2014.

[61] F. Tramèr, P. Duprè, G. Rusak, G. Pellegrino, and D. Boneh, "AdVersarial: Perceptual ad blocking meets adversarial machine learning," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2019, pp. 2005–2021.

[62] B. Wang, T. Zhou, S. Li, Y. Cao, and N. Gong, "GraphTrack: A graph-based cross-device tracking framework," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, 2022, pp. 82–96.

[63] Q. Wu, Q. Liu, Y. Zhang, P. Liu, and G. Wen, "A machine learning approach for detecting third-party trackers on the web," in *Proceedings of the European Symposium on Research in Computer Security*, 2016, pp. 238–258.

[64] Z. Yang, W. Pei, M. Chen, and C. Yue, "WtaGraph: Web tracking and advertising detection using graph neural networks," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2022, pp. 1540–1557.

[65] S. Zhu, Z. Wang, X. Chen, S. Li, K. Man, U. Iqbal, Z. Qian, K. S. Chan, S. V. Krishnamurthy, Z. Shafiq, Y. Hao, G. Li, Z. Zhang, and X. Zou, "Eluding ML-based adblockers with actionable adversarial examples," in *Proceedings of the Annual Computer Security Applications Conference*, 2021, pp. 541–553.

[66] S. Zimmeck, J. S. Li, H. Kim, S. M. Bellovin, and T. Jebara, "A privacy analysis of cross-device tracking," in *Proceedings of the USENIX Security Symposium*, 2017, pp. 1391–1408.

A. Case Study

A.1. Per-sample adversarial attacks (A^4)

```
1 <span>
2   <span>
3     <span>
4       <link href="//securepubads.g.doubleclick.net?918019=
          ↪ people.com#?track##...#\&amp;screenwidth=q"/>
5     </span>
6     <p hidden=""></p><p hidden=""></p>...<p hidden=""></p>
7   </span>
8 </span>
```

Figure 3: A website (`people.com`) manipulated by A^4.

Figure 3 illustrates an HTML snippet of `people.com` after performing a per-sample adversarial attack with A^4 [65]. A^4 attempted to bypass the detection of a network request that fetches an ATS-related JS snippet initiated by Line 4. As highlighted, A^4 placed aggressive modifications on the target HTML document; A^4 added 128,849 fragment identifiers (#) to the URL and 547 paragraph tags (<p>) with a hidden attribute.

A.2. Universal adversarial attacks (YOPO)

Figure 4 shows the impact of our attacks on a website in a real-world hosting scenario. In the original webpage shown in Figure 4a, AdGraph blocks a network request that fetches a JS code snippet responsible for placing an ad at the top of this webpage. When we apply our UAP to the website, the ad bypasses AdGraph and is rendered without any side effects. Figure 5 shows all HTML manipulations made by YOPO on a network request node initiated by Line 8. Here, YOPO wrapped the <script> tag with tags to remove ad-related keywords in the ascendant nodes'

(a) Original webpage

(b) Perturbed webpage

Figure 4: Visual comparison of the original and perturbed webpages of `dezeen.com`. The figure on the right shows that our attack bypasses AdGraph with human-imperceptible perturbations.

```
1  <div>
2    <div>
3      <div class="carousel-true-title" hidden=""></div>
4      <picture hidden="1">
5        ...
6        <span hidden=""></span>
7        <div class="category-type-background" hidden=""></div>
8        <script defer src="https://securepubads.g.doubleclick
           ↪ .net/tag/js/gpt.js#ver3.15.#;#3617x4553">
9        </script>
10       ...
11     </picture>
12   </div>
13 </div>
```

Figure 5: HTML manipulations that YOPO applies to the `dezeen.com` webpage (shown in Figure 4b).

attributes (in Lines 1 and 2). To alter the parent tag name, YOPO wrapped the `<script>` tag with a `<picture>` tag (in Line 4). YOPO also added a semicolon and a size-related expression to the request URL (in Line 8). We highlight all tags and strings that YOPO extracted from the same webpage and reused to reflect the UAP, showing that they are visually indistinguishable from other benign nodes.

B. Efficacy of Adversarial Training

Adversarial training [45] (AT) is a seminal approach to improve the robustness of a DNN model to human-imperceptible adversarial perturbations.

Methodology. To evaluate the effectiveness of AT against YOPO, we adversarially trained target ATS blockers. Since ATS classifiers typically employ ML models such as random forests, implementing adversarial training with these models is less straightforward. To address this challenge, we augmented the training data with a UAP generated by YOPO and then retrained the models on the augmented training set. Specifically, we randomly selected 20K, 40K, and 60K samples from the training set, applied the UAP to these samples, and added them back into the training data. We assess performance using two metrics: the classification accuracy of the defended model and the ASR. Note that we generated a new UAP while setting the perturbation bound ϵ to 10 and used it to attack the defended model.

TABLE 6: Comparison of attack success rates and performance of ATS blockers when applying adversarial training.

Target Model	# of adv. examples	ASR	Accuracy	Precision	Recall
AdGraph	20K	88.59 (0.68↓)	90.93 (1.71↓)	88.53 (1.37↓)	73.45 (11.85↓)
	40K	83.75 (5.52↓)	91.88 (0.76↓)	88.34 (1.56↓)	71.71 (13.59↓)
	60K	81.07 (8.20↓)	92.61 (0.03↓)	87.89 (2.01↓)	70.33 (14.97↓)
WebGraph	20K	70.79 (0.42↓)	95.87 (0.19↑)	92.78 (0.48↓)	91.67 (0.60↓)
	40K	67.18 (4.03↓)	96.07 (0.39↑)	92.59 (0.67↓)	90.14 (2.13↓)
	60K	63.59 (7.62↓)	96.31 (0.63↑)	92.56 (0.70↓)	87.61 (4.66↓)
AdFlush	20K	60.73 (1.18↓)	96.09 (0.16↑)	94.96 (0.38↓)	90.08 (0.79↓)
	40K	56.25 (5.66↓)	96.24 (0.31↑)	94.59 (0.75↓)	88.56 (2.31↓)
	60K	54.05 (7.86↓)	96.23 (0.30↑)	93.74 (1.60↓)	85.41 (5.46↓)
PageGraph	20K	83.74 (0.42↓)	95.65 (0.24↓)	93.43 (0.53↑)	91.64 (1.81↓)
	40K	76.81 (7.35↓)	96.06 (0.17↑)	92.61 (0.29↓)	90.24 (3.21↓)
	60K	75.06 (9.10↓)	96.31 (0.42↑)	91.56 (1.34↓)	86.26 (7.19↓)

Results. We find that AT is ineffective as a countermeasure against our attacks. Table 6 presents the performance of the defended models and their corresponding ASRs. As shown in the third column, the ASR decreased by less than 10% across all ATS blockers, indicating that this approach provides no significant defense. Moreover, this retraining led to a critical performance drop, with AdGraph experiencing a recall decrease of up to 14.97%.

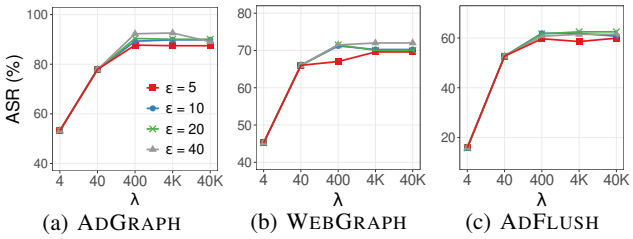


Figure 6: Impact of the attack hyperparameter λ on ASR.

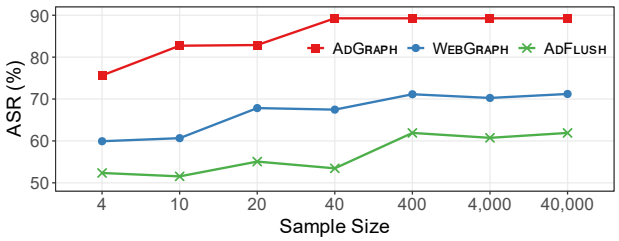


Figure 7: Impact of sample sizes in crafting a UAP on ASR.

We attribute the ineffectiveness of AT against YOPO to the way it perturbs binary and categorical features. YOPO flips the values of these features (i.e., 0→1 or 1→0), which significantly alters the input. While AT is effective in making models robust to *small numerical perturbations*, it becomes ineffective when feature values are drastically changed. This observation aligns with recent findings [57], which also showed that AT struggles to defend against adversarial attacks that generate discrete-valued perturbations.

C. Impact of Attack Configurations

Impact of attack hyperparameters. YOPO uses our optimization function to generate a UAP; the function employs an attack hyperparameter λ. A larger λ indicates a stronger focus on changing the ATS label to the non-ATS label, while a lower value of λ places emphasis on minimizing the total cost of injecting a UAP.

Figure 6 illustrates the variation in ASRs for AdGraph, WebGraph, and AdFlush when we vary λ. We observe that the ASR increases as λ increases. The attack success saturates when λ is 400 in AdGraph and 40 in WebGraph. The results show that it is crucial for our evasion attack to emphasize misclassification (i.e., a higher λ). However, after a certain point, attack success begins to saturate. We note that the adversary can perform this profiling offline on their surrogate models to find an optimal value of λ.

Impact of sample sizes. We now measure the ASRs while varying the size of the samples the adversary is able to use in crafting a UAP. Figure 7 illustrates the impact of sample sizes used for crafting a UAP on ASR. We also set ϵ to 10. We hypothesize that the ASR will decrease as the attacker uses a smaller dataset to generate UAPs. However, even when the attacker uses only four instances for computing a UAP, the ASR remains at 50–70%. This result demonstrates

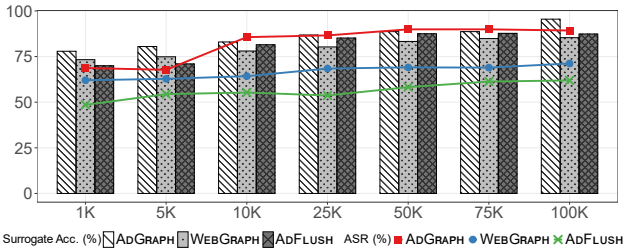


Figure 8: Impact of the training set sizes for the surrogate model on ASR.

that the adversary can still generate an effective UAP even with a small number of samples. This finding is consistent with previous studies in the image domain, where even a small number of sampling instances can lead to high ASRs in adversarial attacks. Khrulkov *et al.* showed that using only 16 sampled images can enable the generation of per-sample adversarial perturbations with a 56% ASR [39].

Impact of dataset sizes in training surrogate models. We hypothesize that if the number of queries an adversary sends to the target ATS classifier reduces, our attack success can decrease as the surrogate models may have degraded performance. Figure 8 shows our results against AdGraph, WebGraph, and AdFlush. The bar graphs represent the test accuracy of different surrogate models, each of which is trained on a corresponding number of queries to the target ATS classifier. The line plots show the ASR when the attacker uses each surrogate model.

As shown in Figure 8, YOPO can yield accurate surrogate models when the number of queries to the target ATS classifiers increases. The improved accuracy results in increased ASRs. We also observe that a number of queries lower than 5K leads to an ASR of less than 50%. However, increasing the number to 5K or more can improve the ASR by up to 90%. In WebGraph, we observe that the ASR increases. However, compared to our results in AdGraph, the ASR is relatively stable. Since ASR is highly dependent on how well the surrogate model mimics the target classifier, we attribute the stability of the ASRs against WebGraph to marginal improvement in surrogate model performance while using more queries.

D. Attacking Different ML Algorithms

We extended the evaluation of YOPO to consider two additional tree-based classifiers, XGBoost and LightGBM, as potential alternatives to the random forest classifiers that our target ATS blockers use. Table 7 presents a summary of the performance of YOPO against these new target models. The first column indicates the ML training algorithms of the target models, and the second column shows the feature sets that the corresponding algorithms leverage.

The third to fifth columns display the overall performance metrics of the corresponding target models. Notably, the performance of the ML models using XGBoost and LightGBM was slightly better than that of the original models.

TABLE 7: ASRs and the performance of three target ML classifiers, using the XGBoost and LightGBM algorithms, are measured using three evaluation metrics: accuracy (Acc.), precision (Prec.), and recall (Rec.).

Target Model	Feature Type	Performance (%)			ASR (%)			
		Acc.	Prec.	Rec.	ϵ = 5	ϵ = 10	ϵ = 20	ϵ = 40
XGBoost	AdGraph	92.55	89.10	85.94	87.22	88.41	89.83	91.01
	WebGraph	96.33	94.66	93.02	70.70	70.62	71.14	71.79
	AdFlush	96.77	95.62	93.50	58.90	60.96	61.04	60.57
LGBM	AdGraph	92.82	89.53	86.41	86.91	88.27	88.49	89.75
	WebGraph	95.75	93.61	92.15	66.23	65.71	67.47	66.89
	AdFlush	96.89	95.82	93.74	60.37	61.18	62.87	62.21

TABLE 8: Average ASRs when attacking multiple ATS requests on a single webpage. For target webpages, we randomly selected 100 webpages, each of which initiated at least five ATS requests.

Target Model	# of avg. ATS requests	ASR (%)			
		ϵ = 5	ϵ = 10	ϵ = 20	ϵ = 40
AdGraph	17.36	82.53	83.68	85.44	85.21
WebGraph	15.36	68.28	68.01	71.58	70.53
AdFlush	15.48	56.12	57.14	68.37	70.41

The sixth to ninth columns show the ASRs with different ϵ values. The experimental results show that regardless of the target ATS classifiers and the ML training algorithm, YOPO consistently achieved high ASRs that were comparable to those observed against AdGraph, WebGraph, and AdFlush using the original random forest algorithm. These results demonstrate the robustness and effectiveness of YOPO as an attack framework against various ATS classifiers employing different training algorithms.

E. Attacking Multiple ATS Requests

We evaluated the effectiveness of YOPO in attacking multiple ATS requests on a single webpage. Table 8 presents the attack performance in this setting.

The second column in Table 8 represents the average number of ATS requests on the selected webpages, and the third to sixth columns show the ASRs with different ϵ values. The experimental results demonstrate that YOPO is capable of successfully attacking multiple ATS requests within a single webpage, achieving average ASRs of greater than 82%, 68%, and 56% for AdGraph, WebGraph, and AdFlush, respectively.

These results highlight the capability of the UAP that YOPO generates because the UAP can be applied to misclassify multiple ATS requests on a single webpage. This ability to target multiple ATS requests significantly reduces the attack cost for the adversary, posing a critical threat to the effectiveness of ATS classifiers.

TABLE 9: Performance of target ATS blockers trained with a corrupted dataset.

Target Model	Accuracy (%)	Precision (%)	Recall (%)
AdGraph	91.97 (0.67↓)	88.94 (0.96↓)	83.94 (1.36↓)
WebGraph	95.17 (0.51↓)	92.54 (0.72↓)	91.26 (1.01↓)
AdFlush	95.53 (0.40↓)	94.85 (0.49↓)	89.98 (0.89↓)

F. Effect of Datasets

Bias in dataset. We built our dataset using Tranco's Top-10K websites. However, this dataset may lack network requests to less popular ad networks, potentially limiting its diversity. To evaluate the impact of this bias, we trained all target ATS classifiers on a new dataset of 10K websites, which consists of Tranco's Top 1K sites and 9K randomly sampled from the 1K to 100K rank range. We observed that YOPO still achieved comparable ASRs to those from the original dataset: 84.11%, 70.26%, and 62.65% for AdGraph, WebGraph, and AdFlush, respectively.

Label corruptions. When constructing a training set, prior studies [37, 43, 56, 58] labeled each network request based on filter lists, which may introduce noisy labels [37]. To evaluate the impact of label corruption on our attack results, we intentionally assigned incorrect labels to 2,000 instances in our training set and $\mathbb{D}_{\overline{ad}}$.

Table 9 shows the performance of the ATS blockers trained on this corrupted dataset. Against these ATS blockers, YOPO attained ASRs of 87.26%, 69.77%, and 63.40% for AdGraph, WebGraph, and AdFlush, respectively. We observed a slight performance drop of less than 2% across the three target ATS blockers. The ASRs showed minimal changes, with decreases of 2.01% for AdGraph and 1.44% for WebGraph, and a 1.49% increase for AdFlush. These findings indicate that label corruption has a minimal effect on our attacks.

G. JS Snippets for 3-gram Frequency

Table 10 presents six types of 3-grams AdFlush leverages for ATS classification and the corresponding JS snippets YOPO appends to increase their frequency.

TABLE 10: Six JS snippets YOPO appends to increase the 3-gram frequency in the AST. Each gram consists of one of the following: Expression (`Expr`), Identifier (`Id`), Statement (`Stmt`), CatchClause (`Catch`), and Literal (`Lit`).

3-gram types	JS snippet
(Expr, Id, Id)	`yopo = [btoa('1'), open, open];`
(Expr, Expr, Stmt)	`yopo = [btoa('1'), btoa('1'), ];`
(Id, Id, Id)	`yopo = [open, open, open];`
(Stmt, Catch, Stmt)	`try { yopo = 1; } catch (e) { console.log(e); }`
(Id, Expr, Lit)	`function yFunc() {}; yopo = [open, yFunc(), 'A'];`
(Id, Expr, Id)	`function yFunc() {}; yopo = [open, yFunc(), open];`