Practical Visual Detection of Social Engineering Attack Campaigns

by

Irfan Ozen

(Under the Direction of Roberto Perdisci)

Abstract

Social engineering attacks threaten users' and organizations' security and privacy. Existing solutions for social engineering attacks are limited in scope and reactiveness, lacking a comprehensive approach to tackle the fundamental traits of these attacks and address emerging threats. We present SEShield, a framework to detect web-based social engineering attacks beyond phishing to fill this gap. SEShield consists of three components: SECrawler for identifying new attack campaigns; SENet, a fully convolutional neural network utilizing federated learning for visual detection; and SEGuard, a real-time browser extension for classification and user alerts. The research primarily evaluated SENet, demonstrating its effectiveness in detecting new instances of social engineering attacks. SENet yielded a detection rate of 99.6% at 1% false positive rate when tested on new SE instances. The proposed framework strengthens defenses against generic social engineering attacks.

Index words: Web-based Social Engineering Attack Campaigns, Deep Learning, Transfer Learning, Federated Learning, Fully Convolutional Neural Networks, Malicious Ads, Browser Plugin, Web Crawler, Computer Vision

Practical Visual Detection of Social Engineering Attack Campaigns

by

Irfan Ozen

B.E., Erciyes University,Türkiye, 2017

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the
Degree.

Master of Science

Athens, Georgia

2023

Practical Visual Detection of Social Engineering Attack Campaigns

by

Irfan Ozen

Major Professor: Roberto Perdisci

Committee: Kyu Hyung Lee
Jin Sun

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
August 2023

Dedication

I dedicate this work to my family members: my father, Hakim Ozen; my mother, Sukruye Ozen; and my sister, Emine Ozen. The unwavering affection, faith, and direction you have bestowed upon me have served as the primary motivation behind my achievements. I am deeply grateful for your consistent presence and support throughout my journey.

Acknowledgments

I would like to express my deep gratitude to Dr. Roberto Perdisci for his unwavering assistance and extraordinary guidance. His support and motivation were instrumental in helping me overcome challenges and discover my way forward.

I would like to express my sincere gratitude to the individuals in my NIS-Lab: Dr. Karthika Subramani, Dr. Phani Vadrevu, and Spencer King. I am truly thankful for their priceless mentorship, knowledge, and constant encouragement during my academic pursuit. Their dedication to excellence and willingness to invest their time in reviewing my work and offering constructive feedback has played an indispensable role in my research accomplishments. I deeply appreciate the insightful discussions, valuable suggestions, and contributions that were made throughout our weekly meetings, as well as their involvement in shaping the development of my thesis.

I thank my committee members, Dr. Kyu Hyung Lee, and Dr. Jin Sun, for their kind assistance and support, which greatly facilitated my progress in obtaining my degree. Furthermore, I thank the School of Computing staff members, including Anne Steward, Samantha Varghese, and Evette Dunbar, for their helpfulness and support.

I genuinely thank the Turkish Government for awarding me this scholarship and providing me with the remarkable opportunity to pursue my studies in the United States. I am truly grateful for this invaluable chance to further my education and expand my horizons.

Lastly, I would like to express my heartfelt recognition to my parents for their unwavering commitment to nurturing and raising me and their boundless love and support for our family. I am profoundly grateful for all the experiences and opportunities they have provided me, which have significantly shaped me into the individual I am today. Their guidance and encouragement have been invaluable in my personal growth and development.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Social engineering (SE) encompasses a broad spectrum of attacks aimed at deceiving users into performing actions that may have important negative consequences for the security and privacy of the users themselves or their organizations (Mann, 2008). These threats exploit weaknesses in humans' decision-making process by using tactics such as pretexts, baiting, phishing, etc. (Krombholz et al., 2015). On the web, SE attacks include attack classes such as scareware (Stone-Gross et al., 2013), tech support scams (Miramirkhani et al., 2017), survey scams (Kharraz et al., 2018), and phishing (Hong, 2012), which result in sensitive data leaks, malware infections, and monetary loss. For instance, US consumers lose billions of dollars annually due to various SE attacks ("New Data Shows FTC Received 2.2 Million Fraud Reports from Consumers in 2020", n.d.).

Unfortunately, social engineering attacks remain understudied, compared to other important threats, such as software vulnerabilities and exploitation, network intrusions, malicious software, and botnets. While phishing attacks, which can be thought of as a specific subclass of SE attacks, have received significant attention (Khonji et al., 2013), to the best of our knowledge no comprehensive defense framework against generic SE attacks has been proposed or evaluated.

Previous studies on SE attacks are either limited to discovering and measuring SE attack campaigns (Subramani et al., 2020; Vadrevu & Roberto Perdisci, 2019) or to studying specific subclasses of SE attacks (Hong, 2012; Kharraz et al., 2018; Miramirkhani et al., 2017; Stone-Gross et al., 2013). Furthermore, existing practical defenses against malicious web pages mostly rely on reactive approaches that do not focus on the fundamental traits of SE attacks and tend to lag behind new threats, thus leaving many users exposed to the latest attack iterations. For instance, URL blocklist services (e.g., Google Safe Browsing ("Google

Safe Browsing", n.d.)) focus on *where* malicious content is hosted, rather than detecting and blocking the malicious content itself. Therefore, attackers evade blocking by simply relocating their attacks to a different hosting location.

A recent work by Yang et al. (Yang et al., 2023) has started to address the problem of detecting and blocking web-based SE attacks by proposing an in-browser defense system named TRIDENT. However, TRIDENT narrowly focuses on detecting JavaScript code served by low-reputation ad networks that often use SE techniques such as transparent overlays to hijack users' clicks and redirect them to a potentially malicious pages. Therefore, TRIDENT is able to detect SE webpages only indirectly, because low-tier ad network code often (but not always (Vadrevu & Roberto Perdisci, 2019)) redirects to SE attacks. Therefore, there is a need for more generic solutions that are able to directly detect generic web-based SE attacks, even when they are not linked specifically to ad network code.

1.1 Problem Definition and Motivation

In this paper, we aim to build an in-browser defense against generic web-based Social Engineering (SE) attacks, such as *fake software downloads*, *sweepstake scams*, *scareware*, *tech support scams* (TSS), etc., but not including phishing websites. The reasons for focusing on SE attacks other than phishing are:

SE attacks differ from phishing: Phishing attacks typically attempt to (i) mimic a specific benign target website (e.g., an online banking site, ecommerce site, etc.) and (ii) directly ask users to submit personal information, such as login credentials, credit card or social security numbers, etc. On the other hand SE attacks do not need to mimic a specific website (though in some cases they may abuse a popular brand/logo) and do not need to directly (or immediately) ask users to submit personal information. For instance, fake software download attacks trick the user into downloading malicious software, TSS attacks convince the user to call a fake support phone number, etc.

Defenses against SE attacks are critically understudied: While there exists an extensive body of work that studies how to detect phishing web pages using visual classification (Abdelnabi et al., 2020; Y. Lin et al., 2021; Liu et al., 2022), SE attacks understudied. Only few previous works have addressed the problem of defending against web-based SE attacks, with a limited focus on collecting/measuring SE campaigns (Subramani et al., 2020; Vadrevu & Roberto Perdisci, 2019) or on detecting ad network code that often leads to SE attack pages (Yang et al., 2023).

Unlike previous work, we aim to build a SE attack detection framework that allows us to (i) discover and collect examples of SE attacks served by different SE campaigns, (ii) train a deep learning-based system to visually detect future instances of such SE campaigns, and (iii) deploy this SE attack detection model into the browser to enable real-time detection of new SE attack pages belonging to previously observed campaigns (see Figure 2.1). Additionally, an important objective of our work is to build a *practical* solution that can be deployed into web browsers running on a variety of devices, including laptop and desktop computers and mobile devices.

Tackling the problem of detecting web-based SE attacks using a deep learning-based visual approach is motivated by the following main observations:

- SE attacks aim to trick users by presenting them with misleading content, and thus tend to have a significant visual component that must be present for such attacks to succeed;
- Large-scale SE attacks are often organized into *attack campaigns*, whereby attacks belonging to the same campaign share a common theme and thus carry similar content and visual traits;
- Recent advancements in deep learning have drastically improved the accuracy of image classification tasks, with popular deep learning frameworks now also available for languages such as JavaScript, making it feasible to build complex in-browser AI systems.

1.2 Contributions

Motivated by the observations outlined above, in this paper we make the following main contributions:

– We propose SEShield, a framework to enable the detection of generic web-based SE attacks. SEShield consists of three components: (i) a system for discovering and collecting examples of SE attack campaigns, named SECrawler, (ii) a deep-learning based visual classifier called SENet, and (iii) a browser extension named SEGuard that classifies web pages in real time and alerts the user when an SE attack is detected. Among these three components, most of our research efforts and contributions are dedicated to designing, implementing and evaluating the SENet classifier. At the same time, we also build a prototype of SECrawler and SENet to enable the collection of SE attacks needed to train SENet and to demonstrate the viability of the SEShield defense framework.

– We implement SECrawler by extending previous work (Vadrevu & Roberto Perdisci, 2019) to discover and collect fresh examples of SE attack campaigns.

We then label the data collected by SECrawler by employing three different human labelers and use the resulting SE attacks dataset to train and evaluate the SENet classifier.

– We design SENet as a novel deep learning-based visual detection model based on *fully convolutional networks* (FCNs), to enable the classification of images of arbitrary sizes. SENet can be deployed in the browser and is able to visually detect SE attack webpages independently of the size of the screen (or browser window) where the page is rendered.

– We conduct an extensive evaluation of SENet to demonstrate its ability to detect new instances of SE attacks linked to campaigns observed by SECrawler. We show that SENet can accurately detect new SE attack instances with a 99.6% detection rate at 1% false positive rate, that it is able to generalize to input images (i.e., webpage screenshots) related to previously unseen screen sizes, and that it is also able to generalize to detecting a number of never-before-seen SE campaigns.

– We implement a prototype of SEGuard as a browser extension using TensorFlowJS ("TensorFlow.js | Machine Learning for JavaScript Developers", n.d.) to demonstrate the utility of the entire SEShield framework as a practical defense against SE attack webpages.

– We will release all components of our SEShield framework as open-source software. Additionally, we will release the dataset of labeled SE attack campaigns collected with SECrawler, which consists of 30 campaigns and more than 3000 SE attack screenshots and 263 metaclusters taken on 30 different emulated devices and screen sizes. To the best of our knowledge, this will be the largest dataset of real-world SE attack webpages available to the research community.

Chapter 2

Framework Overview

In this section, we present an overview of our SE defense framework, named SEShield, which aims to offer a practical defense against generic web-based SE attacks. Figure 2.1 shows the three components that make up our framework: SECrawler, SENet, and SEGuard (we introduce each component in more detail below). In this paper, we focus most of our research efforts on SENet, and build an early prototype of SECrawler and SENet to enable the collection of SE attack examples and to demonstrate the viability of SENet and of the entire SEShield defense framework.

As motivated, we focus on detecting SE attacks beyond phishing, such as *fake software downloads*, *scareware*, *sweepstakes*, *tech support scams*, etc. To detect SE attacks, we take a deep learning-based computer vision approach to leverage the often glaring visual cues exhibited by SE attack campaigns. Additionally, to enable a practical deployment on devices with a variety of screens and browser window sizes, we design SENet to be able to classify images (webpage screenshots) of arbitrary sizes.

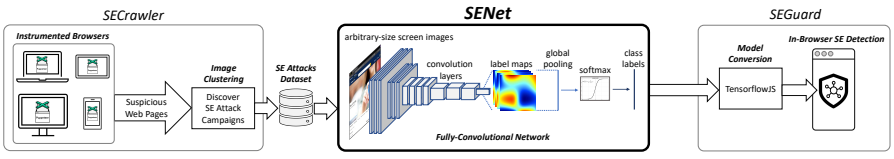


Figure 2.1: SEShield framework overview. SENet represents the primary contribution and main research focus of this paper. SECrawler and SENet are implemented as early prototypes that we use for training data collection and to demonstrate the viability of SENet and of the entire SEShield defense framework.

At a high level, the three components of our SEShield framework work as follows:

2.1 SECrawler

As we aim to learn to automatically identify web-based SE attacks, we first need to be able to continuously collect fresh examples of web pages related to SE attacks campaigns. To this end, we reimplement and extend the SE attack discovery approach proposed by Vadrevu et al. (Vadrevu & Roberto Perdisci, 2019). Specifically, we extend (Vadrevu & Roberto Perdisci, 2019) by significantly increasing the number of different supported devices and screen resolutions (from 5 to 30) and by keeping track of the path followed by the crawler to reach a potential SE attack page. We then use these improvements over (Vadrevu & Roberto Perdisci, 2019) to collect a larger and more detailed datasets of SE attack campaigns, which we use for training and evaluating our SENet module. Our data collection and labeling process will be presented in Framework Details chapter.

2.2 SENet

The SENet module represents the primary contribution and main focus of our research. Using examples of SE attack campaigns collected with our SECrawler, as well as negative examples consisting of benign web pages, we train a novel deep learning-based visual SE attack classifier. As mentioned earlier, we aim to deploy our SE attack classifier on a variety of devices, including laptops, desktops, and mobile devices, which can have widely different screen resolutions. Therefore, we want our classifier to support input images (screenshots) of arbitrary sizes. However, this is challenging and difficult to accomplished with traditional convolutional neural networks (e.g., VGG16, ResNet50, etc.), which are designed to expect images of fixed size in input. To solve this issue, we follow an approach inspired by *fully convolutional networks* (FCNs), which have been previously proposed for image segmentation tasks (Shelhamer et al., 2017), and adapt FCNs to instead build an image classifier that can generalize to classifying web page screenshots of any screen size with high accuracy. We will present the details of the architecture and implementation of SENet in Framework Details chapter.

2.3 SEGuard

Once our SENet classifier is trained, we demonstrate the practicality of our SE defense framework by developing a prototype Chrome extension that deploys the *SENet* model into the browser. Our browser extension, named *SEGuard*,

is programmed to detect SE attacks in near real time. It does so by capturing a screenshot of each webpage on which a meaningful user interaction is observed, such as a mouse click or key press, and by using *SENet* to classify the screenshot image as either *SE Attack* or *Not SE*. Note that our system could also be integrated natively into the browser's code, similar to Chrome's built-in visual phishing detector ("Chromium Blog: Faster and more efficient phishing detection in M92", 2023). More details on the implementation of *SEGuard* are discussed in Framework Details chapter.

In the following sections, we describe the components of our system in more detail and present a comprehensive evaluation of the entire SEShield framework, with particular focus on measuring the detection performance and generalization abilities of our SENet classifier.

Chapter 3

Framework Details

In this section, we provide more details on each of the components of the SEShield framework, with particular focus on SENet, which represents our main contribution.

3.1 SECrawler

While research on phishing defenses can leverage URL feeds such as PhishTank ("PhishTank: Phishing Intelligence", n.d.) and OpenPhish ("OpenPhish: Phishing Intelligence", n.d.) to collect ground truth data, to our best knowledge there exist no analogous feeds or repositories that would allow us to readily collect fresh samples of generic SE attacks. Therefore, we had to build our own system for discovering and collecting recent examples of such attacks in the wild. To this end, we extended the SE attack campaigns discovery approach proposed in (Vadrevu & Roberto Perdisci, 2019). Briefly, the approach is based on a *crawler farm* that is seeded with URLs that are likely to lead to SE attacks. A crawler consists of an instrumented browser that loads a seed URL and automatically interacts with web pages in an attempt to trigger a redirection to an SE attack. A number of heuristics are used to pilot the browser (please refer to (Vadrevu & Roberto Perdisci, 2019) for details) to collect *landing pages* that have a high likelihood of being related to SE attacks. As these landing pages are visited, the crawler records the related URL and a screenshot. Then, to discover SE attack campaigns and filter out noise (i.e., non-SE pages), a webpage screenshot clustering algorithm is used, which is based on the intuition that SE attacks are typically orchestrated into SE campaigns that distribute the same (or very similar) attacks across multiple malicious domain names (Vadrevu & Roberto Perdisci, 2019).

To enable our data collection, we reimplemented the instrumented browsers proposed in (Vadrevu & Roberto Perdisci, 2019) using Puppeteer ("Puppeteer | Puppeteer — pptr.dev", n.d.) and extended it to add the following improvements:

3.1.1 Improved crawler heuristics

In (Vadrevu & Roberto Perdisci, 2019), "click-worthy" page elements were prioritized by sorting them simply in decreasing element size order (i.e., based solely on the size of an element on the rendered webpage). To allow SECrawler to more quickly reach SE attack content, we empirically derived additional heuristics. For instance, we prioritize interacting with page elements (e.g., buttons, images, links, etc.) that include keywords such as *update*, *download*, *play*, *claim*, etc. (we used a total of 38 keywords often found on pages that lead to SE attacks). This small improvement over (Vadrevu & Roberto Perdisci, 2019) allowed our crawler to reach more attack content, when compared to the prior work. Additionally, compared to (Vadrevu & Roberto Perdisci, 2019), we added the use of ("puppeteer-extra-plugin-stealth — npmjs.com", n.d.) to improve the "stealthiness" of our crawlers (i.e., make it more difficult for pages to detect browser automation) and yield a larger set of SE attacks.

3.1.2 Improved data variety

More importantly, compared to (Vadrevu & Roberto Perdisci, 2019), we collect finer-grained contextual information around SE attacks with two improvements: (1) We implemented our SECrawler to track all steps that a crawler followed to reach a given page, which enables *attack traceback*; In later sections, we will discuss how this additional traceback capability helped us develop more effective SE attack detection models. (2) Because one of our main goals is to develop a practical defense that can be deployed on devices with widely different screen resolutions, we significantly extended the number of devices and screen sizes emulated by our crawlers, from 5 to 30; To select the screen sizes and devices to be emulated, we relied on external statistics (StatCounter, n.d.) as well as results from a prior IRB-approved user study (from an unrelated research project) involving 400 MTurk users that allowed us to derive the most popular viewport sizes and screen aspect ratios to be embedded in our SECrawler system.

3.1.3 Systematic labeling process

As in (Vadrevu & Roberto Perdisci, 2019), after suspected SE attacks are clustered into potential campaigns and noise is removed using conservative heuristics, there is a need for manual intervention to confirm and label SE attack campaigns. Unlike in (Vadrevu & Roberto Perdisci, 2019), we systematically label potential SE attack campaigns and remove non-SE pages by using three different human labelers with expertise on web security and social engineering attacks who were tasked to label each cluster of webpage screenshots assigned to them as either *SE Campaign* or *Other*.

After the labelers independently completed their tasks, they met to discuss and attempt to resolve possible label disagreements. In the rare cases in which consensus on the label to be assigned could not be found, they marked the related cluster as *unknown* and excluded it from further consideration. This systematic data labeling process allowed us to collect higher quality SE attack examples.

Besides collecting and labeling examples of SE attacks, we also used our SECrawler to collect benign webpages under 30 different screen resolutions by seeding it with a list of popular websites (the top 600 websites according to Tranco ("A research-oriented top sites ranking hardened against manipulation - Tranco — tranco-list.eu", n.d.)).

3.2 SENet

We build this research around developing SENet, a practical and deployable detection system for Social Engineering attacks. Given its importance, we attempt to explain the reasoning for various design decisions of SENet module. During design phase, We envisioned SENet to be capable of identifying web based SE Attacks tailored by adversaries to target users across a wide range of devices, including mobiles, laptops, tablets, and desktops. To that end, we identify and address the following design questions and explain the steps taken to address them as follows.

DQ1: What traits of SE attacks are unique enough to aid its detection?
DQ2: How to make SENet adaptable in order to detect SE attacks across multiple devices?
DQ3: How to implement it efficiently given limited resources?

First, regardless of the device, once a SE attack is visually rendered in a client's browser, they are known to exhibit certain visual traits that are pertinent to the attack's success. Therefore, we consider these visual cues to be reliable indicators of SE attacks that forms the basis for our decision to build a visual based detection system, *SENet*. In light of this, we opt to approach our attack classification task in a manner similar to an Image classification task, in turn, leading us to adopt deep learning techniques that are state-of-the-art in Image classification tasks.

Second, it should be noted that it is not as straightforward as training traditional convolutional neural networks(CNNs) on SE screenshots to obtain the desired output. One of the benefits of deep learning techniques is that we could use transfer learning to tailor a pre-trained model to suit our current task instead of training a model from scratch thus saving more time and resources. However, some caveats exists in using such pre-trained models; Few traditional CNNs(e.g., VGG16, ResNet50, etc.), are limited by their design to accept images of fixed input size(e.g., 224x224) and fixed aspect ratio. This, in turn, limits us from designing a detection system that can handle images of arbitrary sizes with largely varying aspect ratios. Further, resizing images to fit the input size would easily distort its important features. To address this significant issue, we leverage the advantages offered by Fully Convolutional Networks(FCNs) that were proposed for image segmentation tasks.

3.2.1 FCN Benefits

FCNs, or Fully Convolutional Networks, offer several advantages for tasks like semantic segmentation including handling images of variable sizes and faster training time. Semantic segmentation is a computer vision task that aims to provide a detailed understanding of the image by assigning class labels to individual pixels, unlike other forms of image classification that assign a single label to the entire image. Further, FCN incorporates multiple design features that results in accurate segmentation, even after down sampling, while still preserving fine-grained details within images. These benefits prove to offer the required solution for the shortcomings of some traditional CNN network that requires images of fixed size.

3.2.2 Adopting FCN

To convert a traditional CNN into a Fully Convolutional Network, we included a GlobalMaxPooling2D layer after the last Conv2D layer. This additional layer takes the feature maps produced by the preceding Conv2D layer, identifies the

most significant features, and applies max pooling across the entire feature map. As a result, a single value is obtained for each channel in the feature map. Finally, we included a dense layer in our binary classification task, consisting of two neurons and utilizing the Softmax activation function for classification. Third, another significant challenge is to ensure that we are able to train the customized deep learning model on large dataset of images without running into issues such as memory exhaustion as noticed during the initial experimentation stages. To overcome this challenge, we devised a customized version of the federated learning approach.

3.2.3 Implementation of Federated Learning

While conducting training, we obtained approximately 30,000 images from 60 different resolutions after applying data augmentation techniques. However, attempting to train such a large image dataset on a single GPU, specifically the NVIDIA A40 48GB GPU, and the Dell PowerEdge R750xa with 256GB CPU memory, led to memory exhaustion. To overcome this challenge, we devised a customized version of the federated learning approach. Federated learning (McMahan & Ramage, 2017) is a technique in machine learning where numerous devices or entities collaborate to train a shared model without sharing their raw data. Instead, each device performs local training and only shares model updates with a central server. This approach prioritizes data privacy and minimizes data transfer requirements, making it well-suited for domains such as healthcare and finance.

Custom Fully Convolutional Federated Learning Implementation

As privacy is not a concern in our specific implementation, the clients and server function as a single entity during training. Algorithm 1 summarizes our custom Federated Learning approach

This implementation trains one client at a time using its specific data, which prevents resource exhaustion and provides the benefit of averaging the entire models along with their learned representations. In scenarios involving larger datasets or GPUs/CPUs with limited memory, we can adjust the number of clients to avoid overburdening the resources.

TensorFlow Federated Simulation

Initially, we attempted to utilize the pre-existing Federated Simulation environment provided by the TensorFlow library (accessible at "High-performance simulations with TensorFlow", n.d.) before exploring our custom approach.

Algorithm 1 SENet: Custom Fully Conv. Federated Learning Training

Input:
$SENet$: FCN VGG19 model with Imagenet weights
GE : Global epoch count
$CE = 5$: Client epoch count
CC : Client count,
SWL : List to store scaled client weights
Output: A trained FCN VGG19 model

1: **for all** $j = 1$ to GE **do**
2: **for all** $i = 1$ to CC **do**
3: Set $clientModel.weights = SENet.weights$
4: Create a dataset for ith client by preprocessing an exact number of images from all resolutions per class
5: Train the client model for CE epochs
6: Scale the weights of $clientModel$ based on CC
7: SWL.insert($clientModel.weights$)
8: **end for**
9: Set $averageWeights$ = average weights in SWL
10: Set $SENet.weights = averagedClientWeights$
11: Evaluate $SENet$ on validation data
12: **end for**

However, during experimentation, we encountered a limitation in the simulation environment. The framework loads the entire preprocessed client datasets into GPU memory just before training begins, leading to GPU memory exhaustion.

3.3 SEGuard

To further demonstrate the practicality of our system, we develop an extension for the Chrome browser that allows for detecting SE attacks in real time. As the user browses the web, every time a new non-allowlisted page (i.e., a new URL) is visited, the extension waits to check if the user has started to interact with the page, then takes a screenshots, and passes the related image to our SENet classifier to determine if an SE attack is being visited.

More precisely, SEGuard takes the following steps. (i) When a new page is visited, we can first check if the associated domain name is present in an allowlist (e.g., a list of the top most popular websites according to Tranco ("A research-oriented top sites ranking hardened against manipulation - Tranco — tranco-list.eu", n.d.)), in which case no action is taken. (ii) If the page is not in the allowlist, SEGuard waits for a meaningful user interaction with the page, such as a *click*, *mousedown*, etc., event. If one of such events occurs, a screenshot of the browser viewport is captured. Waiting for a meaningful user interaction with the page is motivated by the fact that, unlike drive-by browser exploits, SE attacks often require the user to interact with the page to click on a button, enter data in a form, etc., for the attack to succeed (notice that other heuristics may also be used to determine if/when to take a screenshot). (iii) Once a screenshot is taken, it is fed into our SENet model, and if it is classified as an SE attack the user is informed via an UI alert that will be shown over the page by the extension.

One important factor in implementing such an extension is performance, especially in terms of classification latency. Initially, we implemented SEGuard by using a large deep learning model based on the VGG19 architecture that we found to have the best detection accuracy. To use the model within our SEGuard extension, we first converted the model using the TensorflowJS ("TensorFlow.js | Machine Learning for JavaScript Developers", n.d.) tool set, so that the model could be loaded and used locally in the context of our browser extension. However, large models tend to have poor performance on regular devices (e.g., a normal laptop) without GPU or other high-performance hardware. To reduce the latency related to obtaining the classification decision for a given webpage screenshot, we therefore implemented a more light-weight version of SENet

based on the MobileNet (Howard et al., 2017) architecture, which provided significant latency reduction with a limited impact on accuracy. We will present more details on SEGuard's evaluation with both models in Experimental Setup and Experimental Results sections.

Chapter 4

Experimental Setup

In this section, we discuss model selection for SENet module, and setting up its prerequisites and test environment.

4.1 SECrawler Setup

Our intent with the added improvements in SECrawler is to reach more SE attack content, by visiting a given seed URL and processing its elements within a limited amount of time. In order to achieve that, it is important that we supply SECrawler with seed URLs that yield a higher number of leads to SE attack websites. We discuss the targeted approach made for collecting Seed URLs, SE websites and benign websites as follows

4.1.1 Environment Setup

We use Docker to run instances of SECrawler so that 1) we could run multiple instances of the crawler simultaneously; 2) each instance would essentially be an isolated and uncompromised environment that does not retain any session information related to crawling other URLs. We set up our crawlers on a server running Ubuntu OS containing 24 CPU cores and configure it to run 15 docker containers at a time. For any given seed URL, we run multiple instances of docker image, each configured for one of the 10 user agents, representing combinations of devices and browsers. We allow each crawling session a maximum time of 15 minutes.

4.1.2 SE Seed URLs

To collect seed URLs that may provide a high-yield of SE attacks, we follow a similar approach used in previous studies (Subramani et al., 2020; Vadrevu &

Table 4.1: Distribution of Seed URLs across Low-Tier Ad Networks

Ad Network	Seed URLs Count	Filtered Seed URLs Count
Adroll	21716	5785
Adblade	2161	293
Adpushup	373	117
Adsupply	205	63
Pop Ads	178	178
Adcash	147	61
AdMaven	118	45
Ad4Game	42	29
AdReactor	24	18
Adsense	15	13

Roberto Perdisci, 2019) and collect URLs of websites that are known to host ads from low-tier ad networks, as they have been observed to lead to SE and malicious content. Similar to the previous works, we compiled a list of such ad networks and keywords associated with each ad network and leverage PublicWWW ("PublicWWW - Source Code Search Engine", n.d.), a reverse code search engine, to find URLs of websites that contained the keywords associated with each ad network. As a result, we collected a total of *24,979* seed URLs associated with *10* low-tier ad networks. In addition, we apply a filtering step on the seed URLs and preprocess the data to improve our efficiency.

4.1.3 Filtering SE Seed URLs

Owing to an extensive and time-consuming crawling process, we apply a filtering step to eliminate seed URLs that may not lead to any SE attacks. As such, we use a simplified version of our crawler to visit each of these seed URLs and to perform clicks faster while recording URLs of all navigations. We use the simplified crawler to fasten the filtering process by avoiding expensive crawling steps such as capturing screenshots and analyzing all visited URLs. Next, we only select those seed URLs that led to any third-party domains during the fast crawling. After such filtering, we found *6,602* URLs, distributed across the *10* ad networks(refer to Table 4.1), that led to one or more third-party domains. Finally, we ran the full version of the SE Crawler only on these selected URLs.

4.1.4 SE Webpage Collection

On visiting these filtered seed URLs, our crawler had collected web pages related to *55,539* distinct URLs and *650,000* distinct screenshots. In addition, to

make our dataset more diverse, we combine *5,030* SE images collected through a previous study (Vadrevu & Roberto Perdisci, 2019) that was collected at a different period of time. We further preprocess the screenshots to remove duplicates, and cluster images of same campaigns and resolution, filter the clusters to find suspicious clusters which are then labeled manually through our systematic labeling process to identify SE attacks. As a result, we found *319* meta clusters consisting of *7,484* images related to SE attacks belonging to *65* different SE attack campaigns. Prior to labeling, the number of images were *7,675* and labellers found 97.51% of these images to be SE related.

4.1.5 Benign Webpage Collection

Alternatively, to collect seed URLs for benign cases, we rely on tranco popularity list and form seed URLs for benign cases with domains containing $rank <= 5000$. On feeding these URLs to SECrawler, even if an interaction with one of these popular domains lead to a URL with domain of $rank > 5000$, the crawler stops processing those navigated URLs. As a result, our crawler was able to collect around *396,000* distinct screenshots from *5,401* distinct benign URLs.

4.1.6 Data Augmentation and Combined Dataset Creation

Before supplying the dataset to our SENet model, we performed the following data augmentations and created a "combined" dataset for additional experiments.

Data Augmentation

To increase the deep learning model's generalization power, we randomly select images to be augmented, which we augment with an augmentation method chosen at random among the ones listed below. This allowed us to increase the size of the dataset to 15.000 benign and 15.000 SE (500 screenshots per resolution). Because of the nature of the webpage content, we made sure not to corrupt either the readability of the text content or the orientation of the webpage object when we chose the augmentation functions. We apply random parameters for the image augmentation functions to reduce the possibility of obtaining duplicate images. Every image undergoes the following image augmentation methods:

Applying Color inversion: We apply it at most once to avoid duplication because there is no randomized parameter.

Applying Gray Scaling: We apply it at most once to avoid duplication because there is no randomized parameter.

Scaling and Random Cropping: We apply it as one operation, images are scaled from random edges, not from the center, between %10 and %20, and then We crop them to their original resolution.

Changing hue: We change the hue randomly between -0.5 and +0.5 of the original hue.

Changing saturation: We change the saturation randomly between %10 and %250 of the original saturation.

Changing contrast: We change the contrast randomly between %10 and %250 of the original contrast.

Changing brightness: We change the brightness randomly between %10 and %250 of the original brightness.

Randomly Solarize: We solarize the pixels of the image if their value is above the randomly defined threshold specified between 0 and 255.

Combined Dataset Creation

We wanted to create an additional dataset to see if we could get an increase in the detection performance of the model. For this purpose, we used the collected traceback logs during the crawling sessions and traced the labeled SE screenshots back to the landing page leading to those attacks. For the benign combined set, we just paired benign screenshots coming from different second-level domains. We followed this path for the benign screenshot because we saw that in almost all cases, the social engineering attacks are hosted on different domains than the publisher landing pages. After that, we concatenated the landing page and attack page pairs side by side and augmented this set for training in the same way we did for the single-image dataset and called it the combined dataset.

4.2 SENet Setup

We performed a series of experiments to choose one of the many pre-trained traditional CNN models that are known for Image classification tasks. Since these models are already trained on large dataset of images, such as ImageNet (Deng et al., 2009; Russakovsky et al., 2015), we leverage "Transfer Learning" technique to reuse the model with pre-trained knowledge and generalizations so that it could be applied to our custom image dataset. To perform the experiments, we disabled all random initializations and tested various models including, Inception-ResNet-v2 (Szegedy et al., 2017), ResNet50V2 (He et al., 2016), VGG-16, VGG-19 (Simonyan & Zisserman, 2015), and Xception (Chollet, 2017)

on a fixed dataset and used a deterministic approach ("TensorFlow determinism", n.d.). On comparing the results of these models, we found VGG-19 to perform better than other models in terms of false positives and false negatives. As a result, we opted VGG-19 to be the core CNN model of SENet. As like other considered CNN models, VGG-19 only accepts images of fixed size and aspect ratio as its input. We customize VGG19's architecture according to the steps discussed in Framework Details chapter to convert it to a Fully Convolutional Network in order for it to handle images of varying aspect sizes.

The VGG19 architecture, represents a deep convolutional neural network (CNN) model comprising 19 layers. This includes 16 convolutional layers and 3 fully connected layers. With a stride of 1, the convolutional layers employ 3x3 filters and are accompanied by rectified linear unit (ReLU) activations. 2x2 max-pooling layers with a stride of 2 are utilized to down-sample the feature maps. The fully connected layers are responsible for the final classification stage, typically employing softmax activation for multi-class classification. To prepare the VGG-19 model for our pipeline, we loaded the VGG-19 model trained on the Imagenet dataset. Subsequently, We removed the classification head and dense layers while adding additional layers to make it a FCN.

4.2.1 Server Setup and Hyper-parameters Tuning

To perform all experiments related to SENet, we used a dedicated Dell PowerEdge R750xa server with 256GB memory and two 48GB NVIDIA A40 GPUs setup to run CUDA 11.8 and TensorFlow.

Before training the model, we fine-tune the model's hyperparameters to extract the best performance results on our dataset. We began by freezing the entire network and only training the added layers. Later, we unfroze selected convolutional layers and fine-tuned the network for a few epochs. Fine-tuning plays a crucial role in transfer learning, allowing a pre-trained model to be adjusted for a new task or domain.

At the beginning, we initiated training using a learning rate of 0.001 and tested lower and higher learning rates. Concurrently, we unfroze either all or a subset of the 4 block5 and 4 block4 convolutional layers. Through experimentation, we discovered that setting the learning rate to 0.00001 and unfreezing all block5 convolutional layers yielded the best outcomes for our SENet model. We chose "Adam" for the optimizer after comparing the results with "Nadam". On experimenting with different scaling factors(ranging from 2 to 8) for the training and testing images, we found that scaling factor of 4 for desktop resolutions and 2 for mobile resolutions gave us an optimal trade-off between training time and detection accuracy. To extract features from images, we chose the im-

age preprocessing functions of TensorFlow after experimenting with imaging libraries from TensorFlow and OpenCV. For the federated aspect of SENet, we found that using 5 local federated clients each training for 5 epochs was optimal while the global server trains for 50 epochs.

4.3 SEGuard Setup

For building SEGuard as a Chrome browser extension that could be easily deployed in the real world, we take essential steps to optimize its speed while not compromising the detection accuracy. Considering that the SENet model based on VGG19 is of larger size, with a high parameter count of 144 million, it may not be the most suitable model for implementing at browser-level. Therefore, we built SEGuard based on a light-weight CNN model, mobilenetv2 (Howard et al., 2017), known for its faster and accurate detection. In terms of detection performance, it is quite comparable to VGG19, as both models achieved a top-1 accuracy of 71.3% on the ImageNet validation datasets (Team, n.d.). However, similar to the traditional VGG19 model, mobilenetV2 expects the input images to be of fixed size and aspect ratio. Therefore, we first transform mobilenetV2 to a FCN by modifying ts architecture to reflect that of SENet, thus forming a light-weight version of SENet called, "MobileSENet". Then, we include an additional Conv2D layer of 128 filters with a stride of 1 before the GlobalMAxPooling2D layer.The objective of this layer is to reduce the feature maps of various sizes derived from the last convolutional layer. (M. Lin et al., 2014) showed that those 1x1 convolutions introduce additional non-linearity into the network, resulting in a decrease in computational workload while also enhancing the model's ability to acquire more robust representations and it is referred to as "Network in Network". Finally, we convert the TensorFlow based MobileSENet to its Javascript counterpart(TensorFlowJS) so that it could be integrated into our browser extension. In Experimental Results section, we report on the performance of SEGuard while using JS versions of SENet model versus MobileSENet model.

Chapter 5

Experimental Results

We evaluated our SENet classifier to answer the following research questions.

R1: Can SENet successfully identify new instances of SEA campaigns and benign screenshots it has seen during the training?
R2: Can SENet successfully identify the same instances of SEA campaigns and benign screenshots when they have completely different resolutions than the ones it was trained on?
R3: Can our SENet be able to detect the examples of SEA campaigns and benign screenshots it has never seen during training?
R5: How SEGuard performs in real-world web browsing?

Our models were trained using a consistent setup across all cases during the training process. This involved having 5 clients and a dataset of 15,000 benign images and 15,000 SEA images, approximately 500 images per resolution after augmentation. The training process included 5 client epochs and 50 global epochs, utilizing the VGG19 model with a batch size of 32. We saved the model at every 5 epochs, and showed the results for the best performing one on the testing dataset fixed for that model. The training was performed using the Adam optimizer with a learning rate of 1×10^{-5}. When showing the test results, we calculated the true positive rate at 1 percent false negative rate and called it the detection rate. Because when we browse the web, wrongly classifying more than a certain percentage of legit benign websites would not give a user a good experience. Therefore, three experts that labeled the dataset in our experiments decided to calculate the detection rate at the 1 percent false negatives and we put that score at the center of our evaluation. Another reason we consider the detection rate at 1 percent false negative rate is that the F1,recall, precision and accuracy scores we shared are calculated at a 0.5 detection threshold and are not

optimal. As a result, optimizing the detection threshold to consider the true positive rate at a maximum false negative rate yields a better evaluation of the model's detection performance.

5.1 SENet Detection Results

Our objective was to assess the detection capabilities of our SENet model on randomly excluded testing dataset. We tested the SENet on 500 benign examples irrespective of the URLs come from, 500 SEA examples across randomly chosen screen sizes. Because we chose those images randomly for testing, the model would see some of them first time, some of them under different resolutions during training. It is important to acknowledge that within a metacluster, the images possess minor differences in their designs. When we remove certain images from a metacluster for testing purposes, the model does not encounter images with the exact same composition during the testing phase. The outcome of the experiment is depicted in Table 5.1, and its zoomed Receiver Operating Characteristic curve (ROC) can be seen in Figure 5.1.

Table 5.1: SENet Detection Results

F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
0.994	0.992	0.996	0.994	TN: 498 FN: 4 FP: 2 TP: 496	0.999	0.996

The SENet demonstrates 99.4% accuracy and 99.4% F1 and the detection rate at 1 percent false positives is 99.6% As a result, we have strong confidence in asserting that when the SENet model is implemented in real-world web browsing applications, it will effectively distinguish new instances of benign webpages and SE attacks with slight changes and some examples with completely different composition and resolution by using only visual cues.

5.2 Generalization to Never-Before-Seen Screen Size

We wanted to see if our model could successfully generalize to the same examples when they have never-before-seen screen sizes or resolutions. Because, in real-world applications, our model can be deployed in devices with different resolutions and viewports than the 30 resolutions we utilized during the training. With that objective in mind, we performed the following experiments.

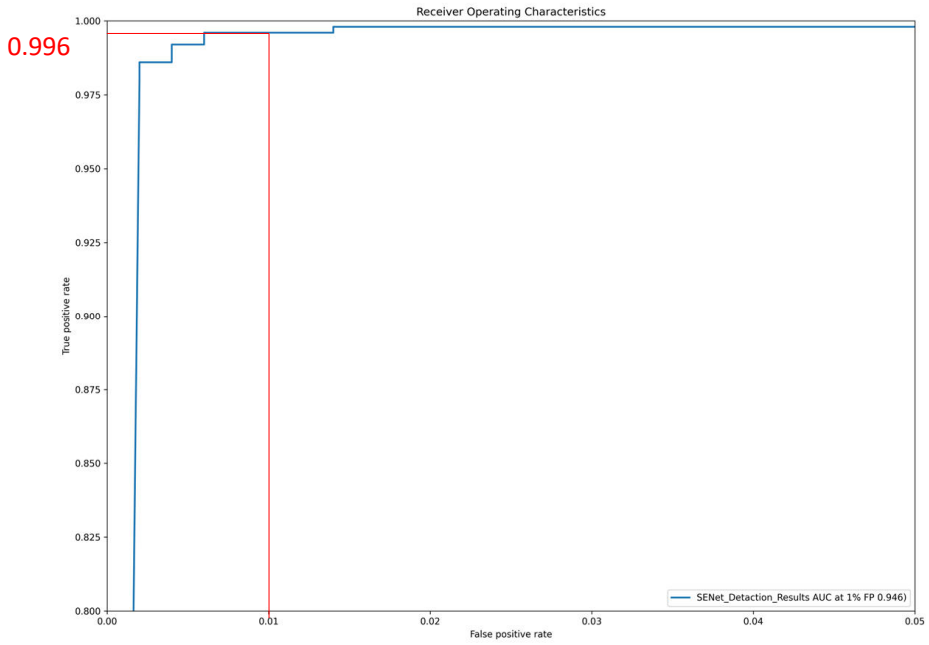


Figure 5.1: ROC for SENet Detection Results

5.2.1 Mixed Mode Experiments

We excluded all benign and SEA campaigns from 9 resolutions in this experiment. Then, we trained 9 models and excluded benign images and SEA campaigns under one resolution at each SENet model's training. When we excluded one resolution for testing, we ensured the SEA campaigns and benign images in the excluded resolution were seen during the training under different resolutions. Because our crawlers crawled the same URLs using different viewports and user agents, we have the same SEA campaigns under different resolutions, and we made sure to have the same SEA campaigns in the testing set to test only the impact of new resolutions on detection performance. In our dataset, we have 25 landscape resolutions having images with wider widths than height, while the other 5 are portrait resolutions having images with smaller widths than height. Out of the 9 total excluded resolutions for testing, 5 are landscape resolutions, while the other 4 are portrait resolutions. Because the images we used during the training have both landscape and portrait resolutions, we called this experiment "Mixed Mode." We got at most 10 images from SEA campaigns and 1000 images per benign resolution from the excluded screen sizes during testing. We chose the resolutions to test the model randomly via a script because

we didn't want to bias and chose some examples that may or may not be easy for the model to classify. We can see the results for those models in Table 5.2. and the ROC in Figure 5.2. Additionally, we combined the results for all the experiments, called it "Global," and added its results to the ROC and the result tables.

Table 5.2: Results for Generalization to Unseen Resolutions

Tested Res.	F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
1.800x1280	0.795	1.0	0.66	0.984	TN:983 FN: 0 FP:17 TP:33	0.998	1.0
2.414x896	0.645	1.0	0.476	0.99	TN:989 FN: 0 FP:11 TP: 10	1.0	1.0
3.768x1024	0.565	1.0	0.394	0.98	TN:980 FN: 0 FP:20 TP:13	0.997	1.0
4.360x640	0.825	0.94	0.734	0.981	TN:983 FN: 3 FP:17 TP:47	0.992	0.88
5.1366x768	0.94	1.0	0.887	0.994	TN:994 FN: 0 FP:6 TP:47	0.999	1.0
6.1920x998	0.943	1.0	0.892	0.996	TN:996 FN: 0 FP:4 TP:33	0.999	1.0
7.1478x837	0.941	1.0	0.889	0.995	TN:753 FN: 0 FP:4 TP:32	0.999	1.0
8.1536x824	0.976	0.959	0.993	0.994	TN:999 FN:6 FP:1 TP:140	0.998	0.959
9.1366x728	0.98	0.984	0.977	0.996	TN:997 FN:2 FP:3 TP:125	0.999	0.984
Global	0.911	0.978	0.853	0.99	TN: 8674 FN:11 FP: 83 TP:480	0.997	0.978

The accuracy for those models is between 98% and 99.6% percent with AUC scores between 99.2% and 100%. During the evaluation of portrait screen sizes using models 1, 2, 3, and 4, it becomes evident that their F1 scores are inferior to those achieved in landscape screen sizes. Specifically, at a resolution of 768x1024, the F1 score reaches 56.5%, which marks the lowest value recorded in the table. However, this particular model exhibits a recall of 100%, an accuracy rate of 98%, and a detection rate of 100% at a 1 percent false positive rate. In regard to the other results obtained from portrait resolutions, the detection rate score ranges between 88% and 100% at the false positive rate of 1 percent. During the evaluation of landscape screen sizes using models 5, 6, 7, 8, and 9, F1 score reaches a minimum of 94%, the recall achieves a minimum of 95.9%, and the

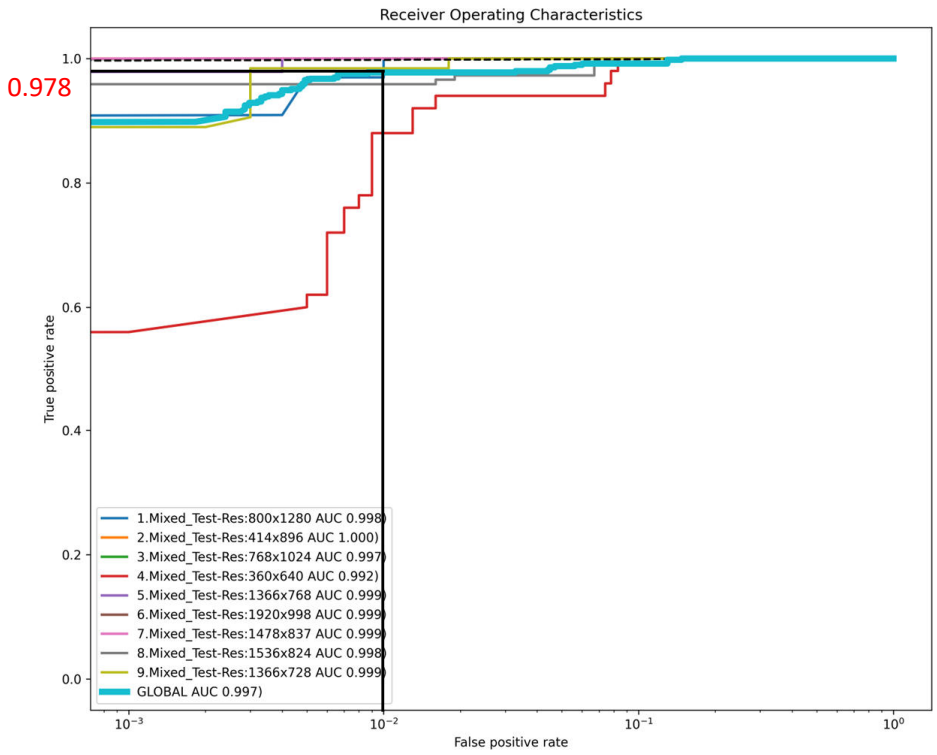


Figure 5.2: ROC Curve for Mixed Mode Experiments

detection rate at a 1 percent false positive rate is as low as 95.9%. The precision score for all landscape resolutions falls within the range of 88.7% to 99.3%. The overall detection performance for the Global Model is very promising, with no more than 11 False negatives out of 491 SEA images and 83 false positives out of 8757 benign images with a detection rate of 97.8% at 1% false positives.

5.2.2 Landscape and Portrait Mode Experiments

This experiment aims to see what happens when we use the same orientation both for testing and training and if we can improve on the results, we obtained in the Mixed Mode experiments. For this purpose, we separated the images under portrait and landscape resolutions and trained them separately. We performed testing on the models using landscape resolutions for the models trained on landscape resolutions and portrait resolutions for the models trained on portrait resolutions. During the landscape training phase, the model saw benign images across 24 landscape resolutions and SEA campaigns across the 24 landscape resolutions. The evaluation was conducted using 1 benign landscape resolution

and campaigns from 1 SEA landscape resolution. Similarly, during the portrait training phase, the model saw benign images across 4 portrait resolutions and SEA campaigns across the 4 portrait resolutions. In this case, the evaluation involved 1 benign portrait resolution and campaigns from 1 SEA portrait resolution. The number of screenshots per resolution during the training and other parameters were the same with the "Mixed mode" experiments to compare them on the same basis. We can see the results for those models in Table 5.3 and the Receiver Operating Characteristic curve (ROC) in Figure 5.3.

Table 5.3: Results for Generalization to Unseen Resolutions for Landscape and Portrait Mode

Tested Res.	F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
1.800x1280	0.892	1.0	0.805	0.992	TN:992 FN:0 FP:8 TP:33	0.999	1.0
2.414x896	0.77	1.0	0.625	0.994	TN:994 FN:0 FP:6 TP:10	1.0	1.0
3.768x1024	0.867	1.0	0.765	0.996	TN:996 FN:0 FP:4 TP:13	1.0	1.0
4.360x640	0.613	0.84	0.483	0.95	TN:955 FN:8 FP:45 TP:42	0.975	0.66
5.1366x768	0.94	1.0	0.887	0.994	TN:994 FN:0 FP:6 TP:47	0.999	1.0
6.1920x998	0.957	1.0	0.917	0.998	TN:997 FN:0 FP:3 TP:33	0.999	1.0
7.1478x837	1.0	1.0	1.0	1.0	TN:757 FN:0 FP:0 TP:32	1.0	1.0
8.1536x824	0.983	1.0	0.967	0.996	TN:995 FN:0 FP:5 TP:146	1.0	1.0
9.1366x728	0.984	0.976	0.992	0.997	TN:999 FN:3 FP:1 TP:124	0.999	1.0
Global Model	0.915	0.978	0.86	0.99	TN:8679 FN:11 FP:78 TP:480	0.999	0.98

We generally observe an improvement when comparing the detection performance of portrait models to mixed-mode experiments. However, there is an exception for the specific resolution of 360x640, where performance is reduced. The detection rate at 1 percent false positive rate decreased from %88 to %66, and the F1 score decreased from %82.5 to %61.3 for this particular resolution. The most noticeable increase in portrait resolutions is for the resolution 768x1024, where the F1 score reached 86.7% from 56.6%.

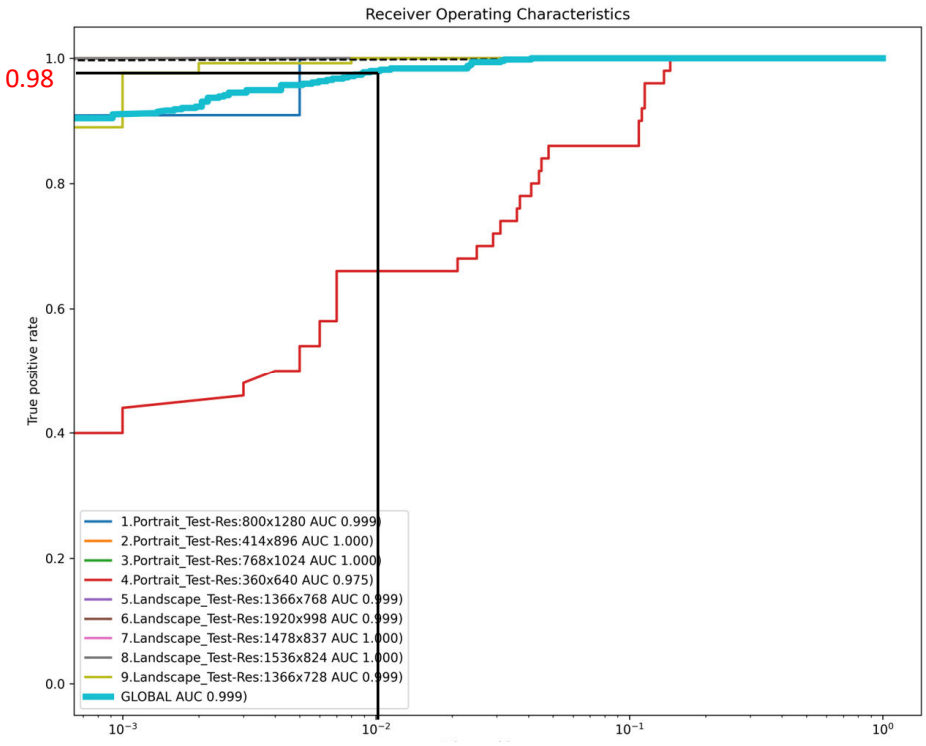


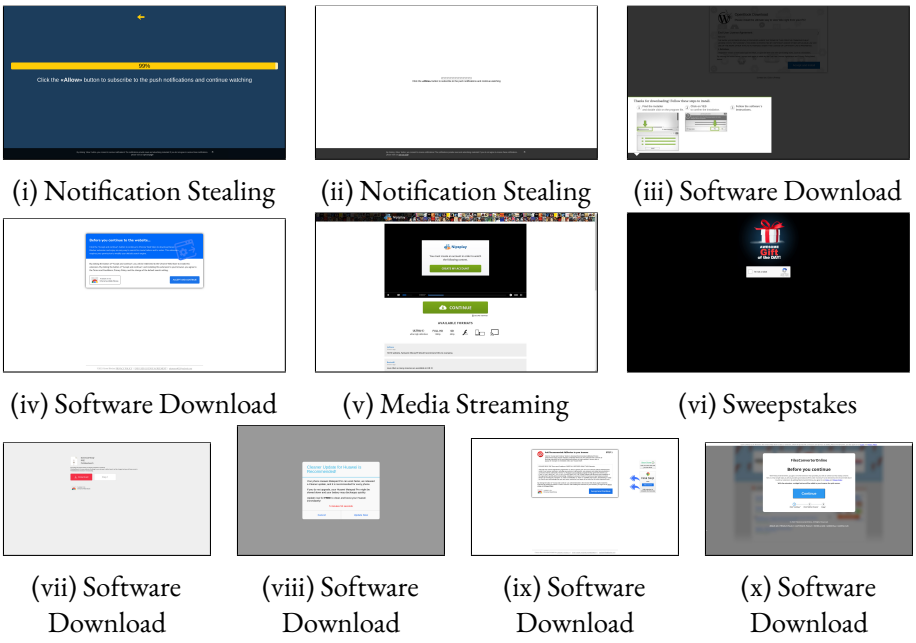
Figure 5.3: ROC Curve for Landscape and Portrait Mode Experiments

The landscape resolution results are similar to those obtained in the mixed-mode experiments, with only a slight improvement in detection performance. For instance, the resolution 1478x837 achieves perfect scores of 100% in all metrics, which is a slight improvement compared to the F1 score of 94.1% in the mixed-mode experiment.

In general, the Global model exhibits 11 instances of false negatives among the 491 SEA examples, which aligns with the results obtained from the mixed-mode experiments. The detection rate global score at a 1 percent false positive rate is determined to be 98%, which is slightly better than that on the mixed-mode experiment where the same score is at 97.8%. As we can see, the overall results when separating the models in landscape mode and portrait mode are relatively comparable.

5.3 Generalization to Never-Before-Seen SE Campaigns

For the second research question, we wanted to see if our model could successfully generalize to the new SEA campaigns and new benign screenshots than the one used in training. We performed the following experiments to answer this question.



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

(iv) Software Download (v) Media Streaming (vi) Sweepstakes

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

Figure 5.4: Examples from excluded SEA campaigns

5.3.1 Single Dataset Experiments

For this experiment, we excluded 10 SEA campaigns and 5000 benign examples irrespective of the URLs they come from and from randomly chosen screen sizes. We trained ten models, and for each model, we excluded one SEA campaign and 500 randomly excluded benign images for testing. The amount of SEA testing images depended on how common the attack campaigns were in our seed URLs. After getting at most 10 images per SEA campaign, the amount ranges from 15 to 110 SEA images coming from 3 to 19 different resolutions. The selection of SEA campaigns for testing was completely random, , and we can see an example of 10 excluded campaigns in Figure 5.4. We can see the results for those models in Table 5.4 and the Receiver Operating Characteristic curve (ROC) in Figure 5.5.

Table 5.4: Results for Generalization to Never-Before-Seen SE Campaigns on Single Dataset

Model name	F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
1.SEA_Single-Model	0.897	0.827	0.979	0.966	TN: 498 FN: 19 FP: 2 TP: 91	0.997	0.973
2.SEA_Single-Model	0.297	0.183	0.786	0.907	TN: 497 FN: 49 FP: 3 TP: 11	0.992	0.87
3.SEA_Single-Model	0.959	1.0	0.921	0.991	TN: 495 FN: 0 FP: 5 TP: 58	0.998	1.0
4.SEA_Single-Model	0.971	1.0	0.944	0.993	TN: 496 FN: 0 FP: 4 TP: 67	1.0	1.0
5.SEA_Single-Model	0.952	1.0	0.909	0.994	TN: 497 FN: 0 FP: 3 TP: 30	0.998	1.0
6.SEA_Single-Model	0.833	1.0	0.714	0.988	TN: 494 FN: 0 FP: 6 TP: 15	0.996	1.0
7.SEA_Single-Model	0.966	1.0	0.935	0.995	TN: 497 FN: 0 FP: 3 TP: 43	0.999	1.0
8.SEA_Single-Model	0.909	1.0	0.833	0.989	TN: 494 FN: 0 FP: 6 TP: 30	0.997	1.0
9.SEA_Single-Model	0.99	1.0	0.98	0.997	TN: 498 FN: 0 FP: 2 TP: 96	0.999	1.0
10.SEA_Single-Model	1.0	1.0	1.0	1.0	TN: 500 FN: 0 FP: 0 TP: 88	1.0	1.0
Global	0.912	0.886	0.94	0.982	TN: 4966 FN: 68 FP: 34 TP: 529	0.997	0.925

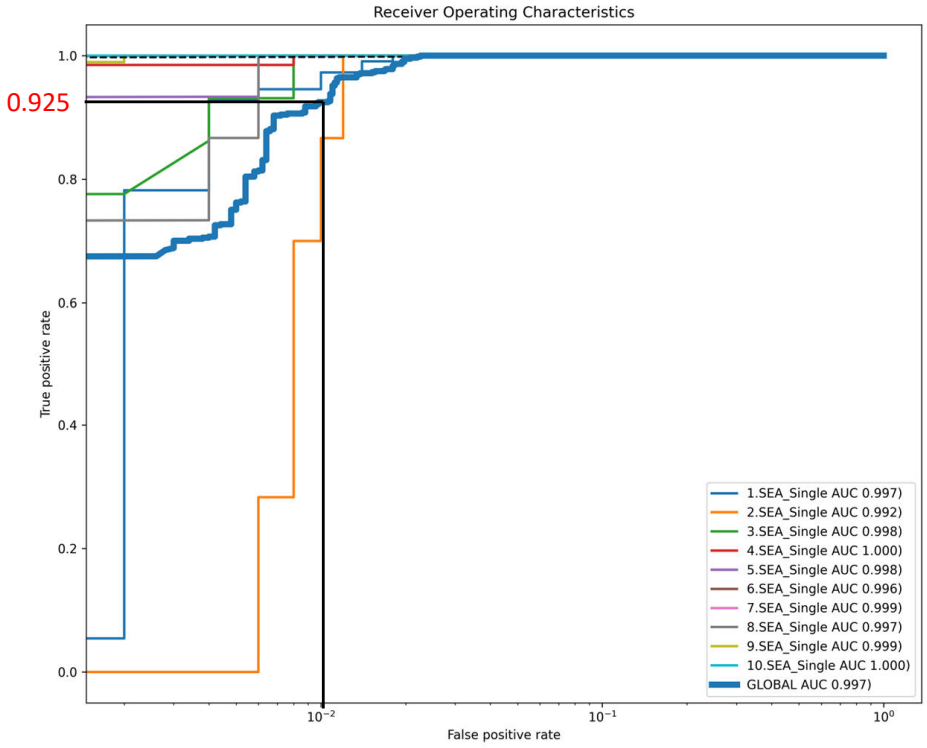


Figure 5.5: ROC Curve for Generalization to Never-Before-Seen SE Campaigns on Single Dataset

The model shows lower detection performance for attack campaign 2, with F1 scores being 29.7%. However, when we consider the detection threshold to obtain 1 percent false positive rate, we reach the detection score of 87%. One possible factor leading to those results is that that attack campaign has a unique design compared to other attack campaigns, making it difficult for the model to differentiate. However, when we see the recall scores, our model can classify

attack campaigns from 3 to 10 with 100 percent recall scores and attack campaign 1 with 82.7

Apart from the attack campaign 2, the F1 scores range between 89.7% and 99%, and the accuracy is between 96.6% and 100%

Trying to classify SEA campaigns that were never seen during training for all the testing examples is a challenging objective. When we check the global scores to understand the model's overall performance better, we can see that the AUC and detection score at 1% false positive rate are 99.7% and 92.5%, respectively, and we have less than 1 percent false positives. To increase the detection performance further, We decided to create a new dataset called "combined" and performed experiments we will mention next.

5.3.2 Combined Dataset Experiments

To increase the detection performance, we concatenated the SEA campaigns with their originating landing pages and benign images from different second-level domains and created the combined dataset, as we mentioned earlier. Here we performed the 3 separate experiments, and at each experiment, we used the same parameters and close image counts with the single dataset experiments. We excluded the same SEA campaigns and trained ten models in each experiment. All of the experiments for the combined datasets contain the same fixed benign dataset. The only difference is that we excluded images from 30 randomly chosen different benign domains, concatenated them randomly among themselves, and ensured they had the same resolution and came from different domains. Our model never saw images coming from those 30 benign URLs during training.

Concatenate new landing and new SEA campaigns

In the first combined experiment, we ensured the model did not see the concatenated images of landing and SEA campaigns during the testing. We wanted to see if the model could detect novel SEA campaigns reached by never-before-seen originating landing pages during the training. We repeated the experiments 10 times for the same SEA campaigns. We can see the results for those models in Table 5.5.

When comparing the models, we observed variations in their performance metrics. Models 4 and 9 achieved the highest F1 scores of 88.4% and 80.5%, respectively, indicating a good balance between precision and recall. Models 5, 6, and 8 exhibited the highest recall of 1.0, suggesting they successfully identified all SEA image instances in the testing set. Model 8 achieved the highest accu-

Table 5.5: Results for Generalization to Combined New Landing and New SEA Campaigns

Model name	F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
1.SEA_Combined_NewLandingNewSEA	0.66	0.591	0.747	0.954	TN: 1337 FN: 45 FP:22 TP: 65	0.977	0.419
2.SEA_Combined_NewLandingNewSEA	0.603	0.583	0.625	0.967	TN: 1338 FN: 25 FP: 21 TP: 35	0.979	0.5
3.SEA_Combined_NewLandingNewSEA	0.777	0.966	0.651	0.977	TN: 1329 FN: 2 FP: 30 TP: 56	0.996	0.85
4.SEA_Combined_NewLandingNewSEA	0.884	0.97	0.813	0.988	TN: 1344 FN: 2 FP: 15 TP: 65	0.998	0.955
5.SEA_Combined_NewLandingNewSEA	0.734	1.0	0.577	0.984	TN: 1337 FN: 0 FP: 22 TP: 30	0.999	1.0
6.SEA_Combined_NewLandingNewSEA	0.536	1.0	0.366	0.981	TN: 1333 FN: 0 FP: 26 TP: 15	0.999	1.0
7.SEA_Combined_NewLandingNewSEA	0.531	0.605	0.473	0.967	TN: 1330 FN: 17 FP: 29 TP: 26	0.968	0.372
8.SEA_Combined_NewLandingNewSEA	0.8	1.0	0.667	0.989	TN: 1344 FN: 0 FP: 15 TP: 30	0.997	1.0
9.SEA_Combined_NewLandingNewSEA	0.805	0.948	0.7	0.97	TN: 1320 FN: 5 FP: 39 TP: 91	0.991	0.729
10.SEA_Combined_NewLandingNewSEA	0.802	0.802	0.821	0.977	1344 FN: 19 FP: 15 TP: 69	0.99	0.784
Global	0.734	0.807	0.673	0.975	TN: 13356 FN: 115 FP: 234 TP: 482	0.986	0.732

racy score of 98.9%, closely followed by Models 4 and 6, with accuracy scores of 98.8% and 98.1%. These models demonstrated a high degree of overall correctness in their predictions. Additionally, the detection rate at 1% FP metric, which measures the model's ability to control true positives at low false positive rates, showed that Models 5,6, and 8 achieved the highest value of 100%, indicating a superior performance in maximizing true positives at low thresholds. Considering the confusion matrices, the global model exhibited many true negatives (TN: 13356) and true positives (TP: 482), indicating its ability to accurately classify both positive and negative instances. However, it had a moderate number of false negatives (FN: 115) and false positives (FP: 234), indicating room for improvement in correctly identifying SEA campaign instances while minimizing false positives.

Concatenate already-seen landing and new SEA campaigns

In the second combined experiment, we paired the previously observed originating landing pages with the 10 SEA campaigns that the model did not encounter during training. The objective of this experiment was to assess the model's ability to detect and when presented with novel examples from familiar landing pages. Those 10 SEA campaigns are the same campaigns we used in other experiments.We can see the results for those models in Table 5.8 and the Receiver Operating Characteristic curve (ROC) in Figure 5.6. Almost all the models has a recall score 100%, except for the model 2, which has 2 FN resulting in 96.7% recall. Models 9 and 10 have high accuracy scores of 99.1% and 98.9%, respectively, and the other model's accuracy is not lower than 97.4%

Models from 2 to 9 have 100% detection rates at 1% false positives. The same score is at the 96.4% at the lowest, which belongs to the model tested on the 1. Attack campaign

Table 5.6: Results for Generalization to Combined Already-seen Landing and New SEA Campaigns

Model name	F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
1.SEA_Combined_SeenLandingNewSEA	0.866	1.0	0.764	0.977	1325 FN: 0 FP: 34 TP: 110	0.998	0.964
2.SEA_Combined_SeenLandingNewSEA	0.853	0.967	0.763	0.986	TN: 1341 FN: 2 FP: 18 TP: 58	0.998	1.0
3.SEA_Combined_SeenLandingNewSEA	0.789	1.0	0.651	0.978	TN: 1328 FN: 0 FP: 31 TP: 58	0.999	1.0
4.SEA_Combined_SeenLandingNewSEA	0.822	1.0	0.698	0.98	TN: 1330 FN: 0 FP: 29 TP: 67	1.0	1.0
5.SEA_Combined_SeenLandingNewSEA	0.625	1.0	0.454	0.974	TN: 1323 FN: 0 FP: 36 TP: 30	0.998	1.0
6.SEA_Combined_SeenLandingNewSEA	0.556	1.0	0.385	0.982	TN: 1335 FN: 0 FP: 24 TP: 15	0.999	1.0
7.SEA_Combined_SeenLandingNewSEA	0.761	1.0	0.614	0.981	TN: 1332 FN: 0 FP: 27 TP: 43	0.998	1.0
8.SEA_Combined_SeenLandingNewSEA	0.667	1.0	0.5	0.978	TN: 1329 FN: 0 FP: 30 TP: 30	0.999	1.0
9.SEA_Combined_SeenLandingNewSEA	0.937	1.0	0.881	0.991	TN: 1346 FN: 0 FP: 13 TP: 96	1.0	1.0
10.SEA_Combined_SeenLandingNewSEA	0.917	1.0	0.846	0.989	TN: 1343 FN: 0 FP: 16 TP: 88	1.0	0.977
Global	0.821	0.997	0.698	0.982	TN: 13332 FN:2 FP: 258 TP: 595	0.998	0.987

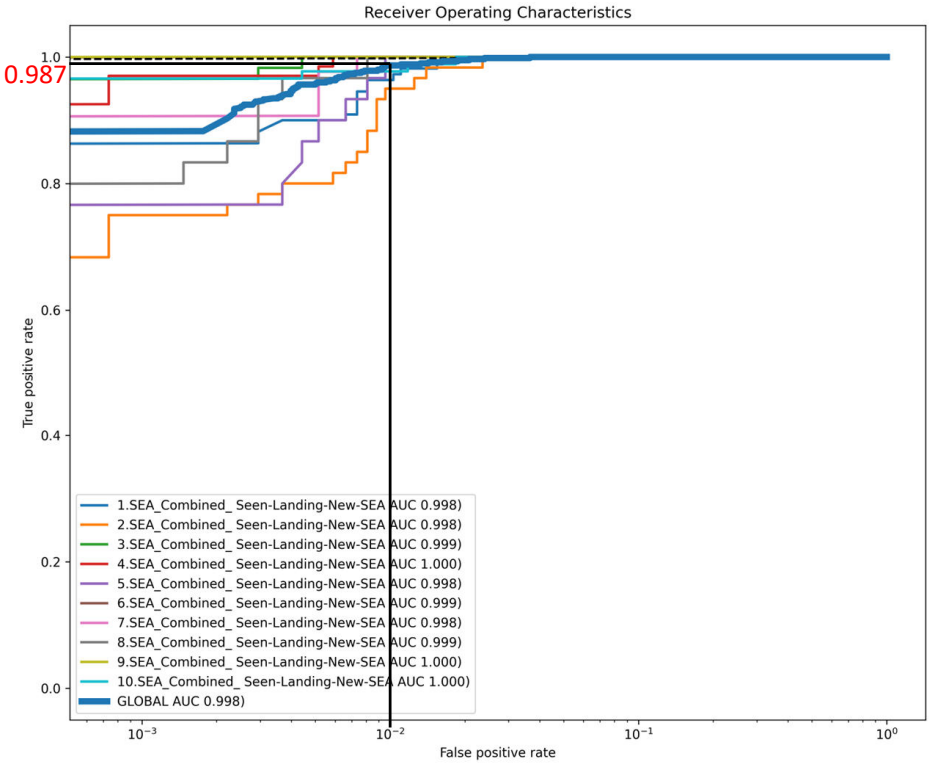


Figure 5.6: ROC Curve for Combined Already-seen Landing and new SEA campaigns

When comparing the combined model's performance on new landing and new SEA campaigns, there is a significant improvement in the detection score at a 1% false positive rate. The detection score increases by 25.5%, reaching 98.7% compared to the previous score of 73.2%. When compared with the single experiment, we get almost 7 percent increase in detection.

This indicates that the model, even when confronted with unfamiliar combined benign pages, demonstrates a strong ability to detect new campaigns it has never encountered before, achieving an impressive AUC score of 99.8%. These results highlight the model's effectiveness in identifying new SEA campaigns, especially when it possesses prior knowledge that a website is the originator of such SEA campaigns. The model's enhanced detection capabilities are evident in its higher detection rate for new SEA campaigns than the other experiments.

Concatenate new landing and already-seen SEA campaigns

In the final combined experiment, we merged previously unseen originating landing pages with the same 10 SEA campaigns. However, in this case, the model was exposed to these campaigns during training. This experiment aimed to observe the model's behavior when encountering a familiar SEA campaign accessed through a new originating landing page.We can see the results for those models in Table 5.7 and the Receiver Operating Characteristic curve (ROC) in Figure 5.7. All the models show pretty good recall scores ranging between 90.7%

Table 5.7: Results for Generalization to Combined New Landing and Already-seen SEA Campaigns

Model name	F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
1.SEA_Combined_NewLandingSeenSEA	0.88	1.0	0.786	0.978	TN: 1329 FN: 0 FP: 30 TP: 110	0.999	0.982
2.SEA_Combined_NewLandingSeenSEA	0.747	0.933	0.622	0.9732	TN: 1325 FN: 4 FP: 34 TP: 56	0.993	0.767
3.SEA_Combined_NewLandingSeenSEA	0.829	1.0	0.707	0.9831	TN: 1335 FN: 0 FP: 24 TP: 58	0.999	0.983
4.SEA_Combined_NewLandingSeenSEA	0.836	0.985	0.725	0.982	TN: 1334 FN: 1 FP: 25 TP: 66	0.998	0.985
5.SEA_Combined_NewLandingSeenSEA	0.698	1.0	0.536	0.981	TN: 1333 FN: 0 FP: 26 TP: 30	0.998	1.0
6.SEA_Combined_NewLandingSeenSEA	0.536	1.0	0.366	0.981	TN: 1333 FN: 0 FP: 26 TP: 15	0.998	1.0
7.SEA_Combined_NewLandingSeenSEA	0.684	0.907	0.549	0.974	TN: 1327 FN: 4 FP: 32 TP: 39	0.991	0.698
8.SEA_Combined_NewLandingSeenSEA	0.69	1.0	0.5263	0.981	TN: 1332 FN: 0 FP: 27 TP: 30	0.998	0.933
9.SEA_Combined_NewLandingSeenSEA	0.857	1.0	0.75	0.978	TN: 1327 FN: 0 FP: 32 TP: 96	0.998	1.0
10.SEA_Combined_NewLandingSeenSEA	0.883	0.989	0.798	0.9841	TN: 1337 FN: 1 FP: 22 TP: 87	0.997	0.955
Global	0.803	0.983	0.679	0.98	TN: 13312 FN: 10 FP: 278 TP: 587	0.997	0.938

and 100%. And the accuracy for all models changes between 97.3% and 98.4%, which is comparable to the combined model trained on already-seen landing and new SEA campaigns. Overall, when the detection rate at 1% false positive is considered on combined global models, the detection performance of this model is higher than the model trained on new landing and new SEA images by around 20.6% and reached 93.8%, and exhibits slightly worse performance than the model trained on the already-seen landing and new SEA campaigns by 4.9%. Notably, the combined models demonstrate promising results when either the originating landing page or the SEA campaign is seen during training.

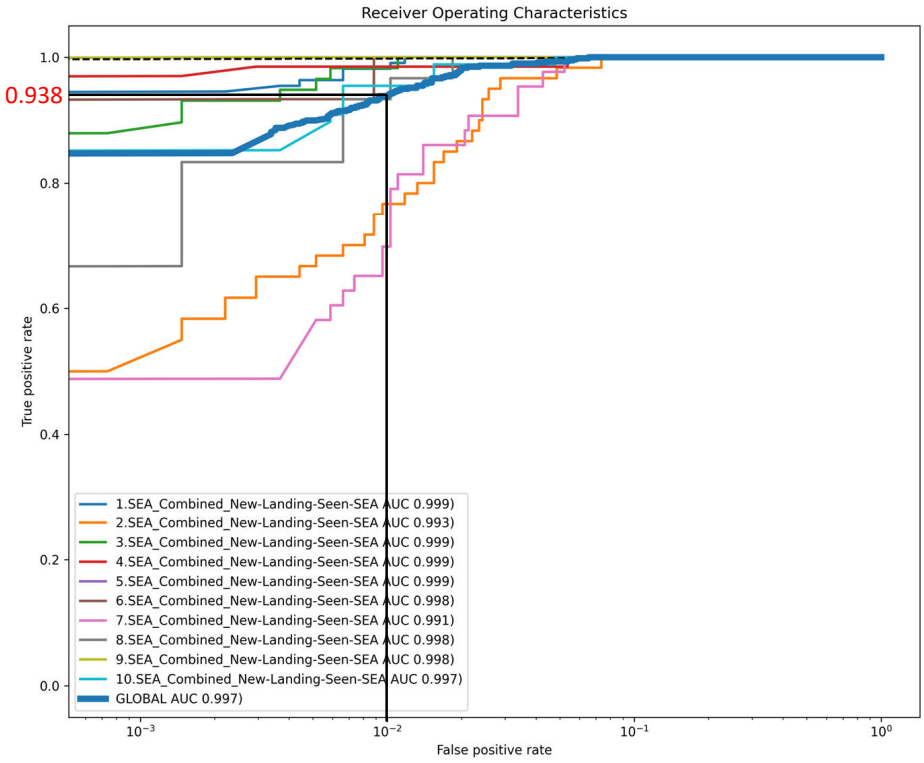


Figure 5.7: ROC Curve for Combined New Landing and Already-seen SEA Campaigns

5.4 The Comparison of Global Models

The results for the comparison of all global models can be seen at table 5.7. When we randomly exclude images irrespective of their SE attack campaigns and screen sizes for testing at SENet Detection, the results yield 99.4% F1 and 99.6% detection rate at 1% false positive rate.

The results when our model sees a new resolution look very promising. At the best model, we got 91.5% F1 score,99% accuracy and 98% detection rate at a 1% false positive rate. We can say that our model does a good job of generalizing to new resolutions.

When our models classify new SEA attack campaigns, The F1 score and the detection score at 1 percent FP rate, and the accuracy are, at best, 91.2%,98.7%, and 98.2%, respectively. In those experiments, the models have never seen those new attack campaigns, and we should note that those are one of the most challenging experiments for the model. To further increase the detection perfor-

mance of the model, we actively and continuously discover new instances of SE attacks and update the model accordingly.

Table 5.8: Results for The Comparison of Global Models

Global Model name	F1	Recall	Precision	Accuracy	Conf. Matrix	AUC	Detection Rate at 1% FP
SENet Detection	0.994	0.992	0.996	0.994	TN: 498 FN: 4 FP: 2 TP: 496	0.999	0.996
Mixed_Resolution_Excluded	0.911	0.978	0.853	0.99	TN: 8674 FN:11 FP: 83 TP:480	0.997	0.978
Landscape_Portrait_Resolution_Excluded	0.915	0.978	0.86	0.99	TN:8679 FN:11 FP:78 TP:480	0.999	0.98
Single_SEA_Excluded	0.912	0.886	0.94	0.982	TN: 4966 FN: 68 FP: 34 TP: 529	0.997	0.925
Combined_NewLandingNewSEA	0.734	0.807	0.673	0.975	TN: 13356 FN: 115 FP: 234 TP: 482	0.986	0.732
Combined_SeenLandingNewSEA	0.821	0.997	0.698	0.982	TN: 13332 FN:2 FP: 258 TP: 595	0.998	0.987
Combined_NewLandingSeenSEA	0.803	0.983	0.679	0.98	TN: 13312 FN: 10 FP: 278 TP: 587	0.997	0.938

5.5 Testing SEGuard

As mentioned in the experimental setup section, when we transitioned to TensorFlow.js ("TensorFlow.js | Machine Learning for JavaScript Developers — tensorflow.org", n.d.) for implementing the browser extension, we encountered performance issues with VGG19 due to its large size and high parameter count of approximately 144 million. It took around 6 seconds to capture a screenshot and classify the webpage as either a SE attack or benign.

Then we decided to run the experiment in which we tested the VGG19 on mixed never-before-seen screen sizes by excluding 9 resolutions and then comparing their global models to see if the MobileNetV2 can perform comparably to VGG19 on our dataset. The ROC curve can be seen in Figure 5.8.

We can see that the detection rate at 1 percent false positive rate for both models is comparable; VGG19 outperformed by 7.8%, and MobileNetV2 successfully detected 90% of SEA campaigns at a 1% false positive rate. The MobileSENet, when applied, shows a detection performance similar to that of VGG19. Subsequently, we conducted a web browser latency test and discovered that the MobileSENet with the MobileNetV2 model could capture and categorize images and alert users at approximately 200ms. This significant reduction in latency, compared to VGG19's 6-second latency, greatly enhances the efficiency of our SEGuard system.

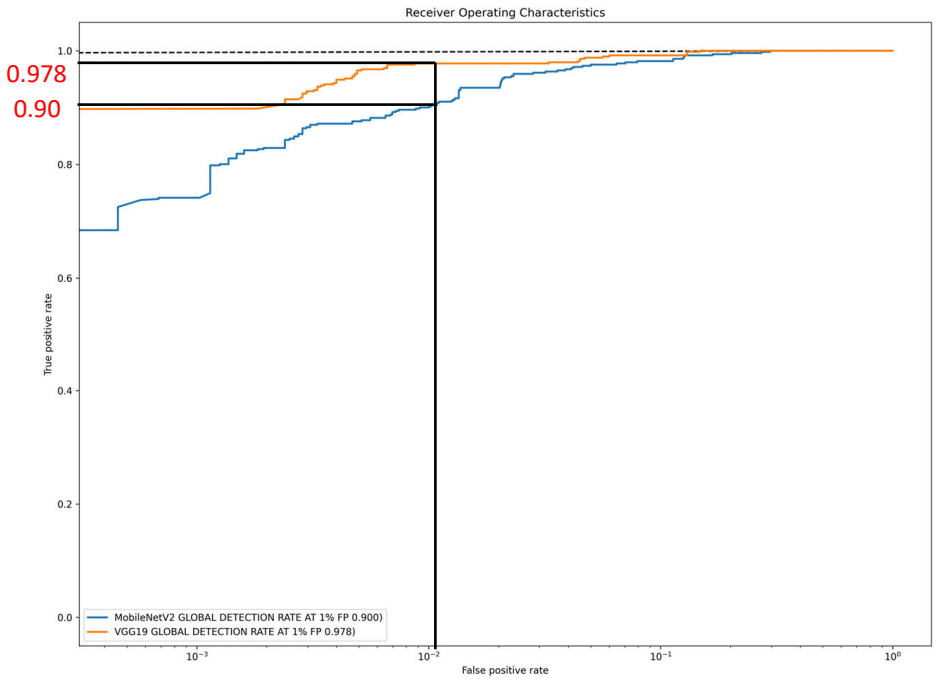


Figure 5.8: ROC Curve for Comparison of MobileNetV2 and VGG19 on Mixed Mode Experiments

Chapter 6

Related Works

Some research focuses on categorical social engineering attacks. Miramirkhani et al. (Miramirkhani et al., 2017) performed an analysis of Technical Support Scams and reported websites and phone numbers used by scammers and the role of malicious advertisements in reaching their purposes. Stone-Gross et al. (Stone-Gross et al., 2013) showed how the victim could be deceived to install fake anti-virus programs by employing scareware SE attacks. Kharraz et al. (Kharraz et al., 2018) employed a machine learning model to detect online survey scams that led victims to reveal personal information for fake prizes or contents. In our research, we do not focus on detecting some specific SE category, instead, we detect generic SE pages, excluding phishing.

Some studies focused on social engineering attacks, primarily identifying, and quantifying the occurrence of social engineering attack campaigns. Subramani et al. (Subramani et al., 2020) created a system called PushAdMiner to collect and discover web push notification messages that can deliver social engineering content, ads, and malicious ad campaigns. Vadrevu and Perdisci (Vadrevu & Roberto Perdisci, 2019) introduced a system that automatically identifies social engineering attack campaigns and detects previously unknown ad networks that promote these campaigns.

Recent work also focused on detecting and blocking web-based SE attacks by employing an in-browser solution (Yang et al., 2023). TRIDENT primarily targets low-reputation ad networks that utilize social engineering techniques, such as transparent overlays, to redirect user clicks to potentially malicious pages. It indirectly detects social engineering webpages, although not all redirects from low-tier ad networks result in such attacks. Our solution finds SE attacks irrespective of low-tier ad networks they have been issued.

Other research focuses on phishing attacks. Phishing is considered as the subclass of Social Engineering attacks (Syafitri et al., 2022). A lot of research

has been done on detecting Phishing websites using visual cues. Abdelnabi et al. (Abdelnabi et al., 2020) used triplet convolutional networks to detect phishing pages by visual similarity. Lin et al. (Y. Lin et al., 2021) created a hybrid system to detect phishing pages using visuality, specifically by the logos of the websites. This system eliminated the need for real phishing examples for training. Liu et al (Liu et al., 2022) presented a technique that is a combination of machine learning and browser instrumentation to detect a phishing page by not only using visual cues but also discovering the intention of a phishing webpage by using prominent webpage elements such as forms. Unlike those solutions, the Social engineering attacks we aim to detect can be effective without directly replicating a particular benign website or immediately requesting users to disclose personal information. However, they might exploit recognizable brands or logos.

There is also other research that focuses on ad blocking based on visual content. Din et al. created "Percival," (Din et al., 2020) a powerful in-browser ad-blocker that employs deep learning to identify and block ads by analyzing perceptual features. They trained a MobilenetV2 model for this purpose.

Chapter 7

Discussion of Adversarial Examples

Our SENet model operates within the web browser, which exposes the underlying model and its parameters. This accessibility creates a potential vulnerability where an attacker can manipulate the model by creating adversarial examples. Adversarial examples are specifically crafted inputs intended to deceive deep learning models. They involve making imperceptible alterations or slight modifications to legitimate inputs to induce incorrect predictions and compromise the overall detection performance of deep learning systems (Chakraborty et al., 2018). It poses a security problem to our SENet model as they can be used to fool it in real-world web browsing.

Potential attackers may attempt to deceive our model by creating modified versions of social engineering examples. In order to achieve this, they could employ techniques such as darkening, brightening, blurring, or adding Gaussian noise to the images (Yu et al., 2019). During the model's training, we employed data augmentation methods and introduced slight variations between the images within a campaign. These strategies aimed to train the model to recognize and adapt to a wide range of similar campaigns, potentially enhancing its resilience against visual perturbations. As part of our future research, we plan to conduct thorough testing to assess SENet's ability to detect and handle such perturbations.

Another method attackers might utilize is the Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014), which computes the gradient of the model's loss function concerning the input image and introduces a small perturbation aligned with the gradient's direction. This perturbed input resembles the original but results in misclassification by the model. The perturbation is controlled with the parameter "Epsilon." The higher the epsilon value, the more the vi-

sual perturbation. Employing larger epsilon values for perturbations increases the likelihood of deceiving a deep learning model, but it diminishes the overall effectiveness of the attack due to the noticeable nature of the perturbations, rendering the victim less susceptible. In future work, we will add those small perturbations calculated by FSGM for various epsilon threshold values to see if our model is robust against them. Depending on the reduction in the detection performance, we will perform adversarial training to render the browser extension more robust against these evasion attacks.

Chapter 8

Conclusion

In this paper, we proposed the SEShield framework, which enables the detection of generic web-based social engineering (SE) attacks beyond phishing without focusing on specific subclasses. The main contributions of this research include the proposal and implementation of SEShield, consisting of three components: SECrawler for discovering and collecting SE attack campaigns, SENet as a deep-learning-based visual classifier, and SEGuard, a browser extension for real-time classification and user alerts.

The primary focus of the research efforts and contributions lies in designing, implementing, and evaluating the SENet classifier. Additionally, SECrawler is built and extended to collect fresh examples of SE attack campaigns, which are then labeled by human labelers to train and evaluate SENet. SENet itself is designed as a fully convolutional network (FCN) based visual detection model using federated learning capable of classifying images of arbitrary sizes, making it adaptable to different browser screen sizes.

The extensive evaluation demonstrated SENet's ability to accurately detect new instances of SE attacks with a good detection rate and low false positives. It showcases SENet's generalization capabilities to previously unseen screen sizes and previously unseen SE campaigns. In order to enhance the detection performance of the model, we proactively and consistently identify new instances of SE attacks and retrain the model. This ongoing process aims to continually improve the model's ability to identify and classify new SE attacks.

Furthermore, a prototype of SEGuard, implemented as a browser extension using TensorFlowJS, shows the practical real-time utility of the entire SEShield framework as a defense against SE attack webpages. Overall, this research provides a comprehensive framework, tools, and evaluation results for effectively detecting and defending against web-based social engineering attacks, thus con-

tributing to the broader field of cybersecurity and assisting in the development of more robust defenses against generic SE attacks.

Bibliography

A research-oriented top sites ranking hardened against manipulation - Tranco — tranco-list.eu [[Accessed 13-Jun-2023]]. (n.d.).

Abdelnabi, S., Krombholz, K., & Fritz, M. (2020). Visualphishnet: Zero-day phishing website detection by visual similarity. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 1681–1698.

Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., & Mukhopadhyay, D. (2018). Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1251–1258.

Chromium blog: Faster and more efficient phishing detection in m92. (2023).

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, 248–255.

Din, Z. A., Tigas, P., King, S. T., & Livshits, B. (2020). Percival: Making in-browser perceptual ad blocking practical with deep learning. *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, 387–400.

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Google safe browsing [https://safebrowsing.google.com/]. (n.d.).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 770–778. https://doi.org/10.1109/CVPR.2016.90

High-performance simulations with tensorflow [[Accessed 13-Jun-2023]]. (n.d.).

Hong, J. (2012). The state of phishing attacks. *Commun. ACM*, *55*(1), 74–81. https://doi.org/10.1145/2063176.2063197

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, *abs/1704.04861*. http://arxiv.org/abs/1704.04861

Kharraz, A., Robertson, W. K., & Kirda, E. (2018). Surveylance: Automatically detecting online survey scams. *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 70–86. https://doi.org/10.1109/SP.2018.00044

Khonji, M., Iraqi, Y., & Jones, A. (2013). Phishing detection: A literature survey. *IEEE Communications Surveys Tutorials*, *15*(4), 2091–2121. https://doi.org/10.1109/SURV.2013.032213.00009

Krombholz, K., Hobel, H., Huber, M., & Weippl, E. (2015). Advanced social engineering attacks [Special Issue on Security of Information and Networks]. *Journal of Information Security and Applications*, *22*, 113–122. https://doi.org/https://doi.org/10.1016/j.jisa.2014.09.005

Lin, M., Chen, Q., & Yan, S. (2014). Network in network. In Y. Bengio & Y. LeCun (Eds.), *2nd international conference on learning representations, ICLR 2014, banff, ab, canada, april 14-16, 2014, conference track proceedings*. http://arxiv.org/abs/1312.4400

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

Liu, R., Lin, Y., Yang, X., Ng, S. H., Divakaran, D. M., & Dong, J. S. (2022). Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. *31st USENIX Security Symposium (USENIX Security 22)*, 1633–1650.

Mann, I. (2008). *Hacking the human: Social engineering techniques and security countermeasures*. Ashgate Publishing, Limited. https://books.google.com/books?id=1veE2f9rPOgC

McMahan, B., & Ramage, D. (2017). Federated learning: Collaborative machine learning without centralized training data. *Google Research Blog*, *3*.

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

New data shows ftc received 2.2 million fraud reports from consumers in 2020 [https://www.ftc.gov/news-events/press-releases/2021/02/new-data-shows-ftc-received-2-2-million-fraud-reports-consumers]. (n.d.).

Openphish: Phishing intelligence. (n.d.).
Phishtank: Phishing intelligence. (n.d.).
PublicWWW - source code search engine [https://publicwww.com/]. (n.d.).
Puppeteer | Puppeteer — pptr.dev [[Accessed 13-Jun-2023]]. (n.d.).
Puppeteer-extra-plugin-stealth — npmjs.com [[Accessed 13-Jun-2023]]. (n.d.).
Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*, 211–252.
Shelhamer, E., Long, J., & Darrell, T. (2017). Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, *39*(4), 640–651. https://doi.org/10.1109/TPAMI.2016.2572683
Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.
StatCounter. (n.d.). Desktop Screen Resolution Stats Worldwide, December 2020 [[Accessed 21-Jun-2023]].
Stone-Gross, B., Abman, R., Kemmerer, R. A., Kruegel, C., Steigerwald, D. G., & Vigna, G. (2013). The underground economy of fake antivirus software. In B. Schneier (Ed.), *Economics of information security and privacy iii* (pp. 55–78). Springer New York.
Subramani, K., Yuan, X., Setayeshfar, O., Vadrevu, P., Lee, K. H., & Perdisci, R. (2020). When push comes to ads: Measuring the rise of (malicious) push advertising, 724–737.
Syafitri, W., Shukur, Z., Asma'Mokhtar, U., Sulaiman, R., & Ibrahim, M. A. (2022). Social engineering attacks prevention: A systematic literature review. *IEEE Access*, *10*, 39325–39343.
Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. *Proceedings of the AAAI conference on artificial intelligence*, *31*(1).
Team, K. (n.d.). Keras documentation: Keras Applications — keras.io [[Accessed 21-Jun-2023]].
TensorFlow determinism [[Accessed 13-Jun-2023]]. (n.d.).
Tensorflow.js | machine learning for javascript developers [https://www.tensorflow.org/js]. (n.d.).
TensorFlow.js | Machine Learning for JavaScript Developers — tensorflow.org [[Accessed 21-Jun-2023]]. (n.d.).
Vadrevu, P., & Roberto Perdisci. (2019). What you see is NOT what you get: Discovering and tracking social engineering ad campaigns. *Proceedings of the ACM Internet Measurement Conference*.

Yang, Z., Allen, J., Landen, M., Perdisci, R., & Lee, W. (2023). TRIDENT: Towards detecting and mitigating web-based social engineering attacks. *USENIX Security Symposium*.

Yu, N., Davis, L. S., & Fritz, M. (2019). Attributing fake images to gans: Learning and analyzing gan fingerprints. *Proceedings of the IEEE/CVF international conference on computer vision*, 7556–7566.