Privacy Protection Against Shoulder Surfing in Mobile Environments

David Darling, Yaling Liu, and Qinghua Li(✉)

University of Arkansas, Fayetteville, AR, USA
{dwdarlin,yl050,qinghual}@uark.edu

Abstract. Smartphones and other mobile devices have seen an unprecedented rise in use among consumers. These devices are widely used in public locations where traditional computers could hardly be accessed. Although such ubiquitous computing is desirable for users, the use of mobile devices in public locations has led to rising privacy concerns. Malicious individuals can easily glean personal data from a mobile device screen by visual eavesdropping without a user's knowledge. In this paper, we propose two schemes to identify and protect private user data displayed on mobile device screens in public environments. The first scheme considers generic mobile applications' complex user interfaces as an image, and uses a deep, convolutional object detection network to automatically identify sensitive content displayed by mobile applications. Such content is then blurred against shoulder surfing attacks. To allow users to identify custom fields in applications that they think should be hidden, we introduce methods for dynamic sample generation and model retraining that only need users to provide a small number of seed samples. The second scheme focuses on web applications due to the popularity of the web platform, and automates the detection and blurring of sensitive web fields through HTML (HyperText Markup Language) parsing and CSS (Cascading Style Sheets) style modification as showcased via a Chromium-based browser extension. Evaluations show the effectiveness of our schemes.

Keywords: privacy · mobile phones · web browsing · obfuscation · shoulder surfing attack

1 Introduction

Mobile devices such as smartphones and tablets have rapidly grown in popularity in recent years. These types of devices offer unparalleled convenience and ease of access for end users who increasingly need to be able to access applications and services on the go. Estimates of smartphone sales trends have shown that the market has rapidly grown since the early 2000s. In 2021 alone, smartphone manufacturers sold an estimated 1.43 billion devices [1]. This was an enormous increase over estimates in 2007 which placed sales at only 122 million units [2].

S. Longfei and P. Bodhi (Eds.): MobiQuitous 2022, LNICST 492, pp. 133–152, 2023.
https://doi.org/10.1007/978-3-031-34776-4_8

With this saturation of mobile devices among consumers, usage has likewise increased rapidly in public spaces.

Due to the convenience of having access to emails, text messages, and other applications, individuals frequently utilize devices in scenarios such as eating, riding a subway, or while walking where others behind or to the side of a user can easily see or visually eavesdrop the content on their device screens [3]. This clearly constitutes a privacy and security risk, as a majority of users are likely to access sensitive apps in public [4] such as text or productivity.

Although there exist privacy screen filters and protector products for mobile devices, they suffer from several drawbacks. They can only address visual eavesdropping attacks that come from the two sides of a user and are out of their view angles (60°C or 90°C for most screen filters/protectors), but attackers behind the user or within the view angle can still see the screen content (see illustration in Fig. 1). They are also known to cause darker screen all the time (since it is uneasy or inconvenient to remove the screen protector) which hurts use experience. Lastly, such filters/protectors cost a user a few dollars to tens of dollars, dependent on the brand and model.

To address the growing problem of visual eavesdropping, we explore software-based solutions for mitigation of visual eavesdropping in public places to allow users to still access their desired apps or websites while providing better protection for sensitive content. Specifically, we propose two schemes. The first scheme considers general mobile applications. To have an easily-deployable solution that does not rely on or make changes to an app's source code, we propose the idea of *user-interface-as-an-image* (UIaaI) to enable image processing and computer vision techniques to be applied to process live app UIs without being hurdled by the complexity of multi-layer UIs. A YOLO [5] style deep convolutional neural network (DCNN) is used to automatically detect sensitive content such as text messages in apps. To improve real-world usability and allow users to customize and dynamically adjust the model, we develop a method for dynamically retraining the sensitive content detection model based on user-specified sensitive content, without requiring too many seeding data samples from users. The second scheme provides an alternative protection method for web browser-based applications which are very popular among users, and provides accurate sensitive content field detection and protection. Specifically, we propose a novel HTML (HyperText Markup Language) parsing and CSS (Cascading Style Sheets) injection method to automatically detect and blur sensitive content on web pages, and implement a prototype system as a Chromium web browser extension that can protect web pages without requiring the pages to be modified by their developers.

The remainder of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 describes the visual eavesdropping attack scenario. Sections 4 and 5 present the two proposed schemes and their evaluation results. Section 6 concludes the paper.

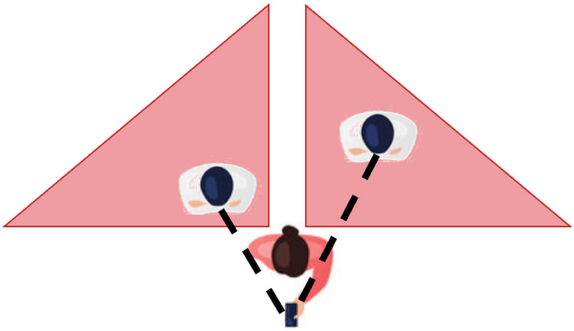


Fig. 1. Visualization of a potential attack scenario. The vulnerable "danger areas" relative to the user are shown in red with attacker gazes shown as black dashed lines. (Color figure online)

2 Related Work

Privacy protection in mobile and ubiquitous environments has been widely studied [6–10]. In the area of mitigating shoulder surfing attacks, much work has been done too. The majority of work relating to shoulder surfing attacks has focused on protecting phone password and authentication credentials. Kumar et al. [11] present a gaze-based password entry method for preventing shoulder surfing attacks by utilizing a user's gaze rather than their hands to enter password digits. Chakraborty and Mondal [12] present a honeypot-based scheme whereby if a shoulder surfer attempts to enter a credential containing a tag digit they would be detected. Yu et al. [13] propose an evolvable password protection scheme using images as keys rather than digit-based authentication. Zhang et al. [14] present an similar evolving password scheme using augmented reality displays to present an input field that is only visible to a specific user. Sun et al. [15] propose a graphical authentication scheme resistant to multiple attacks for password entry. Some works obscure other content types present on the screen. Zezschwitz et al. [16] distort photos on phones against unwanted observations. Eiband et al. [17] convert the font of text messages to the user's handwriting based on the assumption that people's handwriting is harder to read to strangers. Different from them, our work is not limited to any single type of content and attempts to mitigate attacks in more general cases.

Zhou et al. [18] present a scheme that can detect shoulder surfers, and respond by grey-scaling the screen, dimming the screen, limiting the visible area to a small view port that can be dragged, and replacing sensitive data with placeholders. These protective measures do not differentiate private content and non-private content, but our solutions only blur private content without affecting the usability of non-private content. Some works [19,20] adopt eye tracking to display only the portion of the screen that is gazed by the users. Rendering only users' gaze area could potentially expose information that users are reading to attackers, which increases the risk of privacy leakage.

Some works focus specifically on the detection and alerting of shoulder surfing attacks for mobile devices. Saad et al. [21] study several different methods for actually alerting mobile users such as through vibration or visual indication. Lian et al. [22] leverage multiple types of sensors to detect shoulder surfing attacks. Ali et al. [23] present a solution that detects the existence of bystanders, and develop an Android application, iAlert, that notifies the user if content on the screen is readable by bystanders. Our work does not focus on methods for alerting users, but provides an option to automatically identify and protect private content in public environments.

Other works detect or prevent shoulder surfing attacks on non-mobile platforms. Watanabe et al. [24] develop a dummy cursor system for desktop or laptop platforms to hide the true cursor from attackers' views. Li et al. [25] present a shoulder surfing detection scheme for ATMs by tracking human bodies and faces. Brudy et al. [26] employ various sensors to detect shoulder surfing attackers and protect data for large public displays. Our work primarily focuses on mobile platforms due to their ubiquitous and uniquely susceptible nature.

3 Mobile Attack Scenario

In this work, we consider an attack scenario as any occasion where a mobile device user is located in a public space with other individuals. In this environment, the attacking party could be any person near enough to the user to recognize the content displayed on the device screen. Attackers can be located to the side or behind the user where the device screen is visible and not obstructed by the user's body. This work primarily aims to deal with these cases where the user is uniquely vulnerable to being spied on. Figure 1 provides a visual representation of a possible attack scenario. The attackers are featured behind and to the sides of the user in the blind spots highlighted in red.

4 User Interface as an Image (UIaaI) for Private Content Detection and Hiding

4.1 The Basic Idea

With modern mobile applications, there are often many complex layers of UI code to dynamically generate and present content to users in an accessible manner. Because of this, it may be difficult for many mobile app developers to add additional functionality for hiding private content dynamically. This could present an enormous commitment of time for large mobile apps with complex and nested user interface elements where it may be difficult to identify which content is currently visible and needs to be hidden from visual eavesdroppers.

To help mitigate this issue and provide a standardised methodology for dynamically hiding sensitive content on complex applications, we present *UI as an Image (UIaaI)*. The basic idea of this concept is that, computer vision techniques such as object detection and image filtering are highly applicable to

the domain of private content detection, and could replace tedious and time consuming code conversion of existing applications. Under UIaaI, dynamically generated UI views are presented to the user entirely as a pre-rendered, interactive image rather than as direct views of the app UI. This approach offers several benefits, namely the technique for converting applications to utilize UIaaI is standardized for all applications no matter the underlying UI complexity, computer vision techniques can be applied directly to existing views of UI without the need to directly program any content-hiding logic, and complex private content can be automatically identified. Although utilizing images as displayed UI elements eliminates some interactivity, UIaaI can be used temporarily for situations where users are in public spaces. Apps can easily be switched back to their normal behavior once a user no longer has to worry about eavesdropping.

4.2 The Framework

In order to make UIaaI as universally adoptable as possible, we use a simple pipeline for providing conversion in apps. An application on iOS or Android that needs to implement dynamic content protection can follow the following general guidelines:

1. Implement a switchable mode which a user can activate content protection via a button or other UI element.
2. Once a user has switched to UIaaI mode, the application should pre-render its current UI view as a single image frame.
3. The pre-rendered UI view is displayed to the user on-screen.
4. A pass of content detection is performed over the image to identify any sensitive content that should be hidden.
5. A Gaussian blurring filter is utilized to dynamically hide any sensitive content. Note that other alternative blurring methods can be used here too.
6. Once a user has switched back to normal mode, the regular app display logic can be used again.

Utilizing this general guideline, even highly complex app UIs will not require complicated UI logic to enable dynamic hiding of sensitive content. The goal of this is to enable developers to easily make their applications shoulder surfing resistant and encourage adoption of content hiding to benefit end users. Additionally, being able to operate on pre-rendered images enables a variety of interesting computer vision techniques to be applied over the UI such as content swapping to hide content from view with dummy content. Most importantly, we propose that pre-rendered UI can be used to automatically detect private content with an object detection network.

The drawback to this design is a slight loss in user-interactivity. This is due to the fact that image-based views of complex UIs cannot directly emulate all of the visual actions that a traditional UI view is able to generate. For example, although a user's touch events and coordinates of the touch can be recorded and translated into the activation of a button or other element, the normal

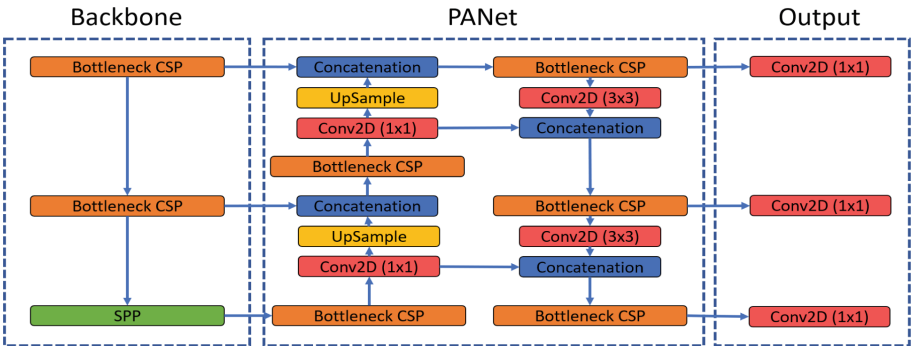


Fig. 2. Architectural overview of the private content detection network.

visualization of a button being depressed would require an additional series of pre-rendered views to be generated. Despite this, we maintain that for suitably large applications, the ease of enabling private content hiding through UIaaI outweighs this temporary lack of interactability.

4.3 Private Content Detection Network

In order to automate the detection of private content, we find that object detection networks such as YOLO [5] or EffecientDet [27], which have been used to great effect in fields of object tracking or generalized object detection, are uniquely useful in visually identifying user interface elements. This is due primarily to the fact that user interface elements generally share very similar visual attributes. For example, in an application which displays emails in an inbox, each email summary view will feature the same general attributes (a subject line, an icon representing whether the item has been read or not, and a brief excerpt from the email text). These similarities in visual appearance which are extremely common among UI elements should be learnable by an object detection network.

We implement an object detection network based on the YOLOv5 [28] architecture. This type of network features high inference speed and smaller model size relative to other state of the art networks. These features make it a prime candidate for inclusion onto mobile platforms. Figure 2 provides an overview of the implemented network architecture. The network relies on a feature extraction backbone built around cross-stage-partial (CSP) network layers [29] along with a spatial pyramid pooling (SPP) layer [30] for getting feature tensors of fixed output size irrespective of input image size. The feature aggregation or neck portion of the network is built around the path aggregation network (PANet) architecture [31] which has been used to great success in competing object detection networks. The final output layers are output from different downsampled feature spaces in the network. This is done to allow the network to identify spatially larger or smaller objects in an input image at different levels of granularity.

Resulting output vectors contain bounding box anchors, box width, box height, class prediction, and confidence interval information.

4.4 An Image Augmentation Method for User-Defined Content Retraining

We recognize that it is an impossible task to fully identify all forms of private content that a user might wish to be protected across many different mobile applications beforehand. Users may have a financial app which displays a summary of account balances which they would not want strangers in public to see for example. To this end, we propose a scheme for dynamic training sample generation. Under this scheme, users can manually identify a UI element they wish to automatically detect and hide within an app. Then many training image samples of the same element will be automatically generated using visual transformations. That way, the user does not have to manually generate many training samples. This is more user-friendly.

Flip Transformation. In order to simulate the different orientations that UI elements might take on a mobile phone, random horizontal and vertical flipping of pre-rendered views provides the model with variation in positioning that might be encountered in the wild as a user accesses their app.

Rotation Transformation. Similar to the flip transformation, the rotation transformation is proposed as a means to introduce more positional variation into a generated dataset. Rotations in 90° intervals are used, as mobile UIs almost never feature non-right angle rotations in the wild. The rotations combined with flip transformations are found to offer a great deal of positional variety even for very small numbers of original training samples.

Random Cropping and Tiling. To simulate the fact that many complex mobile UIs stack or layer UI elements together in the same views, we utilize randomized image cropping and tiling. This method is used to combine together different views of the same elements to force the model to learn to identify elements at any region of a pre-rendered UI view. Complementary slices of training images are used to form a full-sized training sample.

Figure 3 shows sample augmented images generated from a subset of the original text message dataset with only 10 images.

4.5 Private Content Blurring

Once private content is identified on a UI view, a new image is generated with areas containing private content rendered blurry. The obscured image is displayed on the phone screen as a means of defense against eavesdroppers.

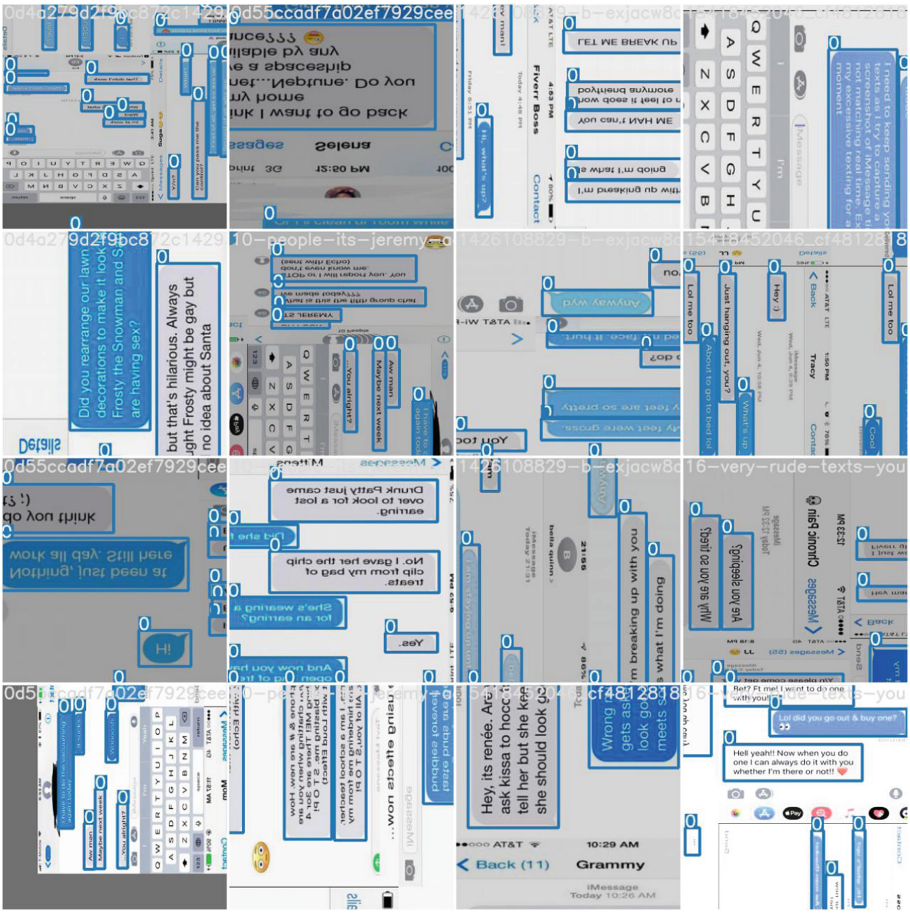


Fig. 3. Visualization of random transformations and augmentations applied over a subset of the full text message dataset.

To reduce the details and noise in the UI view image, we use OpenCV due to its large collection of low-pass filters. The blurring algorithms aim to remove high-frequency content present in an image, such as edges and noise, by convolving the image with low-pass filters. Some of the main blurring techniques provided by OpenCV are Averaging, Gaussian Blurring, Median Blurring and Bilateral Filtering [32]. Since our goal is to filter high frequencies and pass only low frequencies, we find Gaussian Blurring to be the optimal technique. Compared to other filters in OpenCV, Gaussian filters applies a smoother, blurrier effect on the edges.

The Gaussian filter from the OpenCV API takes a kernel size as an input to determine the amount of blurring to apply [33]. It allows a user to customize their needed blurriness level based on the device screen size and use habit.

Although Gaussian blurring is chosen in our implementation, it is worthy to note that our framework is compatible with any blurring algorithms that fit application needs.

4.6 Evaluation Results

Sensitive Content Detection. In order to evaluate the content detection network's ability to detect and classify different types of common UI elements that would contain private content in the real world, a dataset for text messages in the iOS Messages app was created for training the network. Text messages were selected as the UI element of choice in this case because they feature many visual similarities to other common private elements such as email summaries or phone call notifications. Text messages also have the potential to be challenging for an object detection network due to the large variation in size across different messages.

The created dataset contains 188 rendered text message conversations. The network was trained over 1000 epochs to determine how quickly it was able to converge. The resulting model was able to achieve 98.17% mean average precision at 0.5 intersection over union (mAP@0.5). The model was able to achieve this performance after 450 epochs of training. The results for this training are visualized in Fig. 4 along with precision and recall metrics. Sample forward pass predictions are visualized in Fig. 5. These results generally show that the model is highly capable of learning to distinguish UI elements even in a complex layout such as text message stacks.

Training takes 14 min and testing over an image takes about 10ms on an Nvidia Tesla T4.

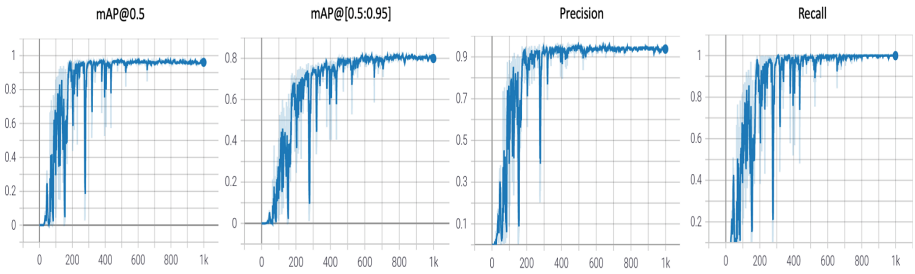


Fig. 4. Bounding box mean average precision (mAP), precision, and recall metrics over progressive training epochs. Note that charts are smoothed with original data shown as a shadow behind. The model is found to converge under all metrics after 450 epochs.

Gaussian Blurring. To achieve the proper amount of blurring effect and to give users more options, we define three blurring levels for the Gaussian filter, {LOW, MEDIUM, HIGH}. Figure 6 displays an example UI view image with Gaussian filters applied. Note that the blurriness levels shown in the figure are

classifications.png

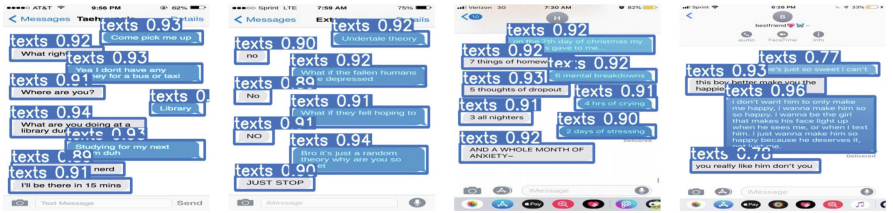


Fig. 5. Sample text detection output with confidence metric. The network is capable of accurately detecting text messages of varying sizes and positions.

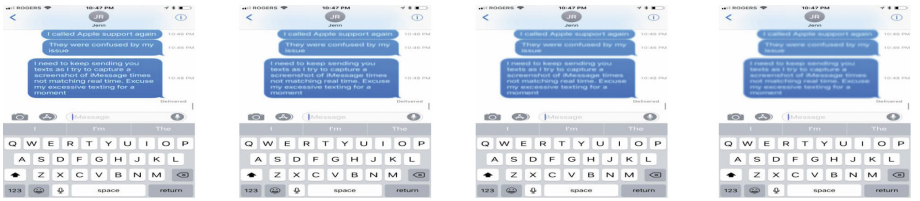


Fig. 6. Gaussian blurring filters applied to private data detected on an UI image of the iOS Message App. The subsequent three images represent three levels of blurring: LOW, MEDIUM, and HIGH.

for illustration purposes; they might be different from what a user sees on a smartphone screen due to difference in image size. In practice, a user can adjust the blurring parameter for their needs.

We conduct an experiment to evaluate the execution time of the Gaussian Blurring program. A total of 90 test cases, allotting 30 for each of the three blurring levels, are run in PyCharm IDE on a Windows 10 computer equipped with a 11th Gen Intel Core i7-1165G7 processor. On average, Gaussian blurring for one image takes 9.9ms of CPU time and 8.3ms of wall time to blur an image. The CPU time exceeding the wall time is due to the fact that the computer uses a multiprocessor.

Evaluation Results for Model Retraining. To validate the training data generation scheme, we utilize a subset of the original text message dataset with only 10 images. These images were used to generate a new dataset of 150 augmented images (see samples in Fig. 3). The same model architecture as presented in Sect. 4.3. The model was able to nearly match the validation average precision of the model trained on the full dataset, reaching a mAP@0.5 of 95.65%. This level of accuracy demonstrates that it is possible to train a robust model for UI recognition with much fewer training samples than are traditionally required in object detection tasks. This is attributed primarily to the similarity in lighting conditions, visual properties, and shape of most UI views which are rarely so similar for more generalized recognition tasks. The results show that a user only

needs to manually label a small number of data samples to train an accurate private UI detection model.

5 Private Content Detection and Hiding for Web Applications

Web users frequently access private services via web applications on mobile devices, which can give strangers the opportunity to acquire sensitive personal information. While the UIaaI approach applies to web browser applications too, in this section, we explore an alternative method dedicated to web applications, considering the unique characteristics of Web UIs.

To protect sensitive user input content in web pages, we develop an HTML parsing approach for detecting user input fields that might contain sensitive information and a CSS style modification approach for blurring sensitive field. In the following, we describe our approach based on a Chromium web browser extension that we implemented as a proof-of-concept. Our extension can be loaded in all Chromium-based browsers, including applications for mobile devices such as Safari on iOS and Kiwi on Android. In 2021, among the top five browsers making up the largest market shares worldwide, all but Firefox were chromium-based [34]. Thus, an extension targeting the Chromium-based browsers can benefit a majority of web users on mobile devices. It is worthy to note that our general approach of HTML parsing and CSS style modification can be adjusted and applied in other browsers too.

Our approach works as follows. Upon page loading, a content script runs on any URL (Uniform Resource Locator) that starts with a permitted scheme (http, https, file, or ftp, etc.). The sensitive data hiding process in the content script begins with the discovery of private information displayed on the web page. If sensitive data is identified, obfuscation techniques based on CSS style are applied to the page sections that contain the data. This logic is also configured to run dynamically to cover a wider range of browsing scenarios.

5.1 Background

Web pages in browsers are based on HTML. HTML uses tags to determine the format of the displayed content. The tags are used to describe headings, paragraphs, tables, link, etc. JavaScript code and CSS can also be embedded in a web page using these tags.

When a user enters a URL, the HTML code is fetched from the server and parsed by the browser. Parsing involves turning the data into tokens and building a DOM (Document Object Model) tree, which mirrors the relationships of the HTML tags. The root of the DOM tree is an $< html >$ element, and a parent node encompasses its child nodes.

Combined with the CSS rules, the DOM tree is then converted to a render tree where a set of computed styles is mapped to each visible node. During the reflow step, the sizes and locations of these nodes are calculated, and in the paint step they are converted to actual pixels on the screen. A page repaint could be caused by additional reflows as the page continues to load and as resources, such as images, get downloaded.

5.2 Identifying Sensitive Content

Private data on web pages, such as passwords, email addresses, phone numbers, and bank accounts, could frequently show up in log-in/sign-up forms and online transactions requiring credit card information. In addition, it is common for businesses to embed a single email address input field in their websites for subscriptions. In order to capture such sensitive content, our solution considers four generalized HTML patterns.

- The HTML $< form >$ elements that use POST as the HTTP method. For any $< form >$ that requires inputs of sensitive data such as passwords, the submitted data should not be exposed in the URL. Thus, these forms must use POST as their HTTP methods.
- An $< input >$ tag with a type attribute that specifies the input type. Among a list of values for the type attribute, 'email', 'password' and 'tel' (short for telephone) allow the $< input >$ element to contain private data. Therefore, these input fields need to be searched for individually if they are not wrapped up in a $< form >$ tag.
- An $< iframe >$ is used to embed another HTML document into the current website. Some web pages involving online payment enclose all the input fields (such as a credit card number field) in an $< iframe >$ for stronger security protection. According to the Same Origin Policy, if the embedded content comes from a different origin than the current window where the extension script runs, the content inserted in the $< iframe >$ tags cannot be accessed by the script. This extension checks whether the *allowpaymentrequest* attribute of a captured $< iframe >$ is set to true. If so, the $< iframe >$ must contain input fields that ask for input of credit-card-related information.
- Login buttons. Clicking these buttons changes the DOM tree. Sometimes, the change displays input fields for users to enter their usernames and passwords.

5.3 Dynamic Detection of Sensitive Content

Sensitive content detection needs to be performed dynamically when the loaded pages contain scripts or JavaScript events that generate private content.

Our web browser extension carries out content detection when a page is first loaded. The content script of the extension sets the *run_at* field to *document_idle* indicating that either *window.onload* event or *DOMContentLoaded* event has been called. However, in either case, there might still be scripts running as the event is triggered. Any additional node added into the DOM tree by these scripts cannot be captured in time because it does not exist in the tree when the content detection script is running.

To tackle this problem, we first explore the effectiveness of *window.requestAnimationFrame* which takes as an argument a callback function. The browser calls the callback whenever a page repaint occurs. This implementation catches all freshly inserted nodes. However, since page repainting occurs often to modify styling when a page is loaded, websites can suffer from severe performance deterioration. A better approach is to invoke the *setTimeout* function. The content detection and CSS style injection logic can be directly passed as a parameter callback to this function. This process is less costly, in terms of performance, because the callback can be delayed a few milliseconds before executing. This generally allows for enough time for all web page elements to load before attempting to detect sensitive content.

Another scenario where a content detection pass occurs is when any JavaScript event triggers a change in the DOM tree. This could occur when a user interacts with an element on a web page which activates a login field. To get around these interactive elements, a list of keywords is used: "login", "log in", "signin", "sign in", "sign up", "signup", "register", "join", "create new account", and "try it free". Any interactive element containing these keywords is attached with a click event listener to the element's event list.

5.4 Blurring Sensitive Content via Applying CSS Styles

Once sensitive content has been detected on a web page, the extension applies a set of CSS styles including color, text-shadow and font-weight for text and blurring for images to adjust their visual appearance, such that attackers will find it significantly more difficult to identify specific content like letters or numbers. The following algorithm applies the blurring CSS styles to sensitive content:

1. Apply styles to any captured HTML $< form >$ elements that use POST methods. These could include forms such as payment information submissions.
2. If email input fields are the only captured elements, CSS styles are applied to these and the associated submit buttons or links. Buttons or links are identified as closest in the DOM tree rooted at the parent node of the email address input field. A recursive search of the parent tree is performed until

one is found. This step is vital for many online shopping websites, as they often contain email address input fields to allow easy subscriptions.
3. If the discovered content contains any HTML elements other than email address input fields, the CSS styles are simply applied to the entire page. In this scenario the DOM tree could contain iframes, password fields, or telephone fields.

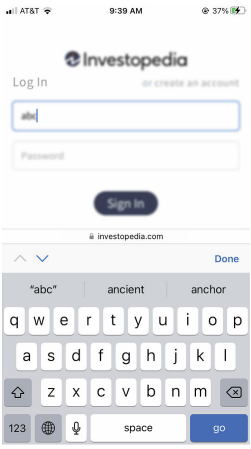
5.5 Evaluation

Evaluation of Private Content Detection Accuracy. Our web browser extension for private content protection was tested on both Android and iOS devices. Kiwi Browser is a chromium-based browser available in the Play Store on Android 12, and it supports most chrome desktop extensions. With its built-in functionalities, we could easily load our extension in the browser. As for testing in Safari on iOS 15.4.1, extra steps were required in converting our source file to a Safari web extension using Xcode's command-line tool.

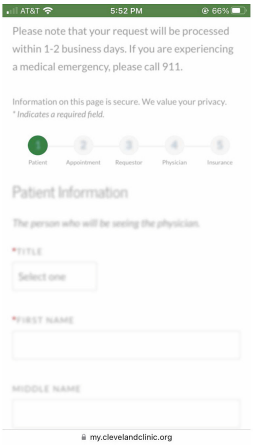
We looked at a series of 100 websites with the highest traffic ranked by RankRanger [35]. Except for four insecure and pornographic websites ytmp3.cc, xnxx.com, xvideos.com and pornhub.com, the other 96 websites were visited for testing, navigating a total of 483 pages which averages 5 pages per website. Throughout the process, two categories of test cases, namely false negative and false positive, are documented. False negative cases represent sensitive information failing to activate the extension app, and false positives indicate that blurring effects are applied to web pages that do not contain private data.

The results of Kiwi browser on Android and Safari on iOS prove to be consistent. Out of a total of 483 browsed pages, 24 (5.0%) pages are false negative and 18 (3.7%) are false positive. The main reasons that the extension fails to capture sensitive content stem from sign-up and log-in forms. For those that are triggered by JavaScript click events, some of the forms cannot be detected by the extension because the list of keywords we incorporated (see Section IV.C) does not cover all possible scenarios of web design. Additionally, some forms containing private data are not configured by a POST method, and some blanks that are not properly marked in the source code, such as an address input field on a McDonald's delivery service page. On the other hand, false positives are primarily due to the fact that some websites wrap up certain HTML elements, including search fields, 'Add to Cart' buttons or product options, with a POST-method $< form >$ element, which triggers the blurring effect application. We leave these issues to future work.

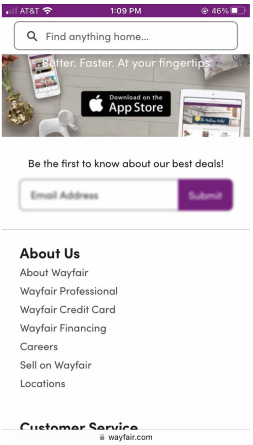
Blurring with CSS Style. Fig. 7 displays example web pages with obfuscating styling applied. In practice, a user can customize the blurriness level based on their needs.



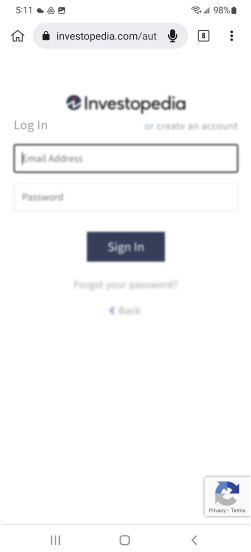
(a) Investopedia sign up page on iOS



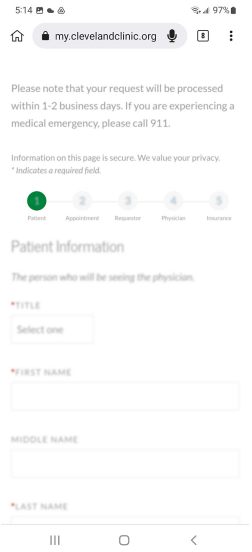
(b) Patient information on iOS



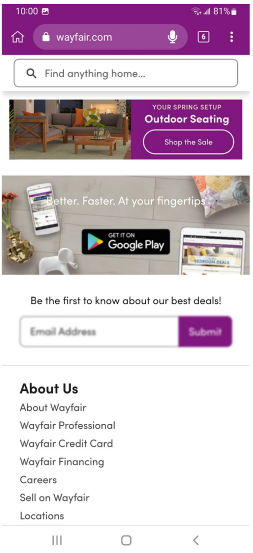
(c) Wayfair email subscription page on iOS



(d) Investopedia sign up page on Android



(e) Patient information on Android



(f) Wayfair email subscription page on Android

Fig. 7. CSS blurring styles applied to the automatically detected user inputs. (a) to (c) are obfuscation results of three web pages on iOS, and (d) to (f) show results of the same web pages on Android. Note that the blurriness levels shown here are for illustration purposes; they might be different from what a user sees on a smartphone screen due to difference is image size. In practice, a user can customize the blurriness level based on their needs.

Fig. 8. Total Blocking Time measurements for web pages that contain private data and those that do not contain private data.

Evaluation of Web Page Loading Time. The extension is applied when a web page is loaded in the browser, which introduces JavaScript code to run at the page load. Due to the critical importance of keeping web pages' loading time low [36], we evaluate whether the JavaScript code causes any significant delay.

The experiments are run on Chrome Lighthouse [37] in Windows 10, and it emulates Moto G4 phone when loading web pages. Chrome Lighthouse is designed to measure web page qualities automatically. For each run, it generates a Lighthouse report that logs metrics indicating different aspects of the web page's performance.

To test web pages with the highest traffic, the top ten websites are selected from the same list used for evaluating content detection accuracy [35]. For each website, we picked two pages to run, one that contains private data and one that does not. To increase the accuracy of the experiments and lower the variance,

five lighthouse tests are run on every page, and the averaged speed is calculated and exported as the final collected data.

Among the list of metrics, we choose Total Blocking Time [38] as the indicator for whether the extension delays users from browsing and interacting with the web page. Total Blocking Time records the duration of time before the page can respond to any user actions, such as scrolling down the page and clicking a button. The recording process begins when any part of the page content is rendered on the screen, and ends when the page becomes fully interactive. The results of the experiments are visualized in Fig. 8 with the orange line representing the Total Blocking Times of web pages when the extension is not applied, and the blue line illustrates the metric of the same web pages when the extension is run upon page load. For pages containing private content and those without private content, the orange line and the blue line generally overlap each other in both cases, which means that the extension does not increase the loading time and does not block users from their normal browsing experience. The reason is that the detection and blurring run fast, and they run as the web pages load content.

6 Conclusion and Future Work

In this work, we presented solutions for protecting private content on mobile device screens from visual eavesdroppers in public spaces. First, an automated private content protection scheme for generic applications was proposed. The scheme considers user interface as an image to simplify the detection of private content and achieved 98.17% mean average precision at 0.5 intersection over union. To allow users to more easily retrain the network to recognize custom identified content, we presented a series of training image transformations which allow for generation of new sample images from a small manually-labeled set. Upon the UI images containing private content, blurring is applied before they are displayed to the users. Then, we proposed an alternative approach for protecting content on web applications and a Chrome browser extension capable of automatically detecting and applying blurring to web page input fields which could contain private user information. We evaluated this extension on 96 high-traffic websites and found that the extension was able to perform well in most scenarios for both log in, sign up, and payment forms. These results demonstrate the potential for privacy protection against shoulder surfing attacks.

In future work, we will perform a user study to assess the effectiveness of privacy protection in the real world, and explore the tradeoff between privacy and usability by adjusting the blurriness level.

References

1. Gartner Says Global Smartphone Sales Grew 6% in 2021. https://www.gartner.com/en/newsroom/press-releases/2022-03-01-4q21-smartphone-market-share. Accessed 31 Aug 2022
2. Market Share: Smartphones, Worldwide, 4Q07 and 2007. https://www.gartner.com/en/documents/619509/market-share-smartphones-worldwide-4q07-and-2007. Accessed 31 Aug 2022
3. Saad, A., Liebers, J., Gruenefeld, U., Alt, F., Schneegass, S.: Understanding Bystanders' tendency to shoulder surf smartphones using 360-degree videos in virtual reality. In: Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction (MobileHCI 2021), Article 35, pp. 1–8. Association for Computing Machinery. New York, NY, USA (2021). https://doi.org/10.1145/3447526.3472058
4. How Americans Use Their Cellphones in Public. https://www.pewresearch.org/internet/2015/08/26/chapter-2-phone-use-in-public-areas/. Accessed 31 Aug 2022
5. Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement (2018)
6. Darling, D., Li, A., Li, Q.: Automated bystander detection and anonymization in mobile photography. In: EAI International Conference on Security and Privacy in Communication Networks (SecureComm) (2020)
7. Darling, D., Li, A., Li, Q.: Feature-based model for automated identification of subjects and bystanders in photos. In: IEEE International Workshop on the Security, Privacy, and Digital Forensics of Mobile Systems and Networks (MobiSec) (2019)
8. Li, A., Darling, D., Li, Q.: PhotoSafer: content-based and context-aware private photo protection for smartphones. In: IEEE Symposium on Privacy-Aware Computing (PAC) (2018)
9. Li, A., Du, W., Li, Q.: PoliteCamera: respecting strangers' privacy in mobile photographing. In: International Conference on Security and Privacy in Communication Networks (SecureComm) (2018)
10. Li, A., Li, Q., Gao, W.: PrivacyCamera: privacy-aware photographing with mobile phones. In: IEEE International Conference on Sensing, Communication and Networking (SECON) (2016)
11. Kumar, M., Garfinkel, T., Boneh, D., Winograd, T.: Reducing shoulder-surfing by using gaze-based password entry. In Proceedings of the 3rd symposium on Usable privacy and security (SOUPS 2007), 13–19. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/1280680.1280683
12. Chakraborty, N., Mondal, S.: Tag digit based honeypot to detect shoulder surfing attack. In: Mauri, J.L., Thampi, S.M., Rawat, D.B., Jin, D. (eds.) SSCC 2014. CCIS, vol. 467, pp. 101–110. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44966-0_10
13. Yu, X., Wang, Z., Li, Y., Li, L., Zhu, W.T., Song, L.: EvoPass: evolvable graphical password against shoulder-surfing attacks. Comput. Secur. 179–198 (2017)
14. Zhang, R., Zhang, N., Du, C., Lou, W., Hou, Y.T., Kawamoto, Y.: AugAuth: shoulder-surfing resistant authentication for augmented reality. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6 (2017). https://doi.org/10.1109/ICC.2017.7997251
15. Sun, H.-M., Chen, S.-T., Yeh, J.-H., Cheng, C.-Y.: A shoulder surfing resistant graphical authentication system. IEEE Trans. Dependabl. Secur. Comput. **15**(2), 180–193 (2018). https://doi.org/10.1109/TDSC.2016.2539942

16. Zezschwitz, E., Ebbinghaus, S., Hussmann, H., Luca, A.: You can't watch this! Privacy-respectful photo browsing on smartphones. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI 2016). Association for Computing Machinery, New York, NY, USA, 4320–4324 (2016). https://doi.org/10.1145/2858036.2858120
17. Eiband, M., Zezschwitz, E., Buschek, D., Hußmann, H.: My scrawl hides it all: protecting text messages against shoulder surfing with handwritten fonts. In Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA 2016). Association for Computing Machinery, New York, NY, USA, 2041–2048 (2016). https://doi.org/10.1145/2851581.2892511
18. Zhou, H., et al.: Enhancing mobile content privacy with proxemics aware notifications and protection. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI 2016). Association for Computing Machinery, New York, NY, USA, 1362–1373 (2016). https://doi.org/10.1145/2858036.2858232
19. Khamis, M., Eiband, M., Zürn, M., Hussmann, H.: EyeSpot: leveraging gaze to protect private text content on mobile devices from shoulder surfing. Multimod. Technol. Interact. **2**(3), 45 (2008). https://doi.org/10.3390/mti2030045
20. Ragozin, K., Pai, Y., Augereau, O., Kise, K., Kerdels, J., Kunze, K.: Private reader: using eye tracking to improve reading privacy in public spaces. In Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI 2019). Association for Computing Machinery, New York, NY, USA, Article 18, pp. 1–6 (2019). https://doi.org/10.1145/3338286.3340129
21. Saad, A., Chukwu, M., Schneegass, S.: Communicating shoulder surfing attacks to users. In: Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia (MUM 2018), pp. 147–152. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3282894.3282919
22. Lian, S., Hu, W., Song, X., Liu, Z.: Smart privacy-preserving screen based on multiple sensor fusion. In IEEE Trans. Consum. Electron. **59**(1), 136–143 (2013). https://doi.org/10.1109/TCE.2013.6490252
23. Ali, M.E., Anwar, A., Ahmed, I., Hashem, T., Kulik, L., Tanin, E.: Protecting mobile users from visual privacy attacks. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication (UbiComp 2014 Adjunct), pp. 1–4. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2638728.2638788
24. Watanabe, K., Higuchi, F., Inami, M., Igarashi, T.: CursorCamouflage: multiple dummy cursors as a defense against shoulder surfing. In: SIGGRAPH Asia 2012 Emerging Technologies (2012)
25. Li, C., Liang, M., Xiao, K., Fong, S., Wang, Q., Song, W.: Human body and face detection based anti-shoulder attack system on ATM. In: Proceedings of the International Conference on Big Data and Internet of Thing (BDIOT2017), pp. 145–148. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3175684.3175706
26. Brudy, F., Ledo, D., Greenberg, S., Butz, A.: Is anyone looking? Mitigating shoulder surfing on public displays through awareness and protection. In: Proceedings of The International Symposium on Pervasive Displays (PerDis 2014), pp. 1–6. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2611009.2611028
27. Tan, M., Pang, R., Le, Q.V. : EfficientDet: Scalable and Efficient Object Detection (2020)

28. Jocher, G., e al.: ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations (2021)
29. Wang, C. -Y., Mark Liao, H. -Y., Wu, Y. -H., Chen, P.-Y., Hsieh, J.-W., Yeh, I.-H.: CSPNet: a new backbone that can enhance learning capability of CNN. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 1571–1580 (2020). https://doi.org/10.1109/CVPRW50498.2020.00203
30. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8691, pp. 346–361. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10578-9_23
31. Liu, S., Qi, L., Qin, H., Shi, J., Jia, J.: Path Aggregation Network for Instance Segmentation (2018)
32. OpenCV: Smoothing Images. https://docs.opencv.org/3.4/d4/d13/tutorial_py_filtering.html. Accessed 31 Aug 2022
33. OpenCV: Image Filtering. https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html. Accessed 31 Aug 2022
34. Browser Market Share Worldwide -December 2021. https://gs.statcounter.come/browser-market-share. Accessed 31 Aug 2022
35. Top 100 Websites Ranking on the Web. https://rankranger.com/top-websites. Accessed 31 Aug 2022
36. Kivilohkare, G.: Optimizing the Critical Rendering Path for Decreased Website Loading Time. Åbo Akademi (2020)
37. Lighthouse - Chrome Developers. https://developer.chrome.com/docs/lighthouse/. Accessed 31 Aug 2022
38. Total Blocking Time - Chrome Developer. https://developer.chrome.com/docs/lighthouse/performance/lighthouse-total-blocking-time/. Accessed 31 Aug 2022