A User-Friendly Two-Factor Authentication Method against Real-Time Phishing Attacks

Yuanyi Sun, Sencun Zhu
School of EECS, The Pennsylvania State University
University Park, PA 16802
cognsoft@gmail.com, sxz16@psu.edu

Yan Zhao, Pengfei Sun
Shape Security
Santa Clara, CA
{yzhao,psun}@shapesecurity.com

***Abstract*—Today, two-factor authentication (2FA) is a widely implemented mechanism to counter phishing attacks. Although much effort has been investigated in 2FA, most 2FA systems are still vulnerable to carefully designed phishing attacks, and some even request special hardware, which limits their wide deployment. Recently, real-time phishing (RTP) has made the situation even worse because an adversary can effortlessly establish a phishing website without any background of the web page design technique. Traditional 2FA can be easily bypassed by such RTP attacks. In this work, we propose a novel 2FA system to counter RTP attacks. The main idea is to request a user to take a photo of the web browser with the domain name in the address bar as the 2nd authentication factor. The web server side extracts the domain name information based on Optical Character Recognition (OCR), and then determines if the user is visiting this website or a fake one, thus defeating the RTP attacks where an adversary must set up a fake website with a different domain. We prototyped our system and evaluated its performance in various environments. The results showed that PhotoAuth is an effective technique with good scalability. We also showed that compared to other 2FA systems, PhotoAuth has several advantages, especially no special hardware or software support is needed on the client side except a phone, making it readily deployable.**

I. INTRODUCTION

Two-factor authentication (2FA) is a user authentication technique which requires end users hold at least two types of information that can confirm his claimed identities, based on something they know, something they have, or something they are. While 2FA can significantly improve account security, the arms race with phishing has never stopped. Traditional 2FA is still vulnerable to phishing because a deceived user may input their second factor information (e.g., PINs received through emails, SMS) into the fake website, which defeats the second layer of protection. Moreover, in the past an adversary had to manually design the web page layout to mimick the real one, which is time-consuming for the adversary. Recently, the new real-time phishing (RTP) tools, like "Evilginx" [4], have made the situation even worse. Now, an adversary only has to download the tool and run it with proper configuration to automatically replicate from the real website. In other words, RTP tools have significantly lowered the technical barriers for adversaries to launch more powerful phishing attacks.

Several methods have been proposed to detect the traditional phishing by measuring the similarity of web page elements (e.g, image size, position) [18] or tree structures of two websites [22]. Unfortunately, such methods would not work in RTP, because the fake website keeps replicating its content from the real website through reverse proxy. Recently, new 2FA systems have been introduced and deployed, such as Duo Push [3], U2F [11]. The research community has also proposed novel proof-of-concept 2FA systems [14], [17], [23]. However, most of them require special devices (or hardware configurations) [11], [17], or some are still vulnerable to RTP attacks [3], [17].

In this work, focusing on defeating the advanced RTP attacks, we propose a new 2FA system called *PhotoAuth*. Here, after a user passes the first factor (e.g., password) authentication, the web server will require the user to take a photo of the web browser with domain name in the address bar as the 2nd authentication factor. The phone automatically uploads the photo to the web server through a web app invoked in the browser of the phone. The web server side extracts the domain name based on Optical Character Recognition (OCR), and then determines if the user is visiting this website or a fake one, thus defeating the RTP attacks where an adversary sets up a fake website with a different domain.

Compared with many other 2FA methods, PhotoAuth has several advantages. First, PhotoAuth can counter RTP attacks that traditional 2FA cannot handle. Second, PhotoAuth does not need any special device other than smartphones that are commonly used. No additional extension/plugin or Bluetooth is required for the browser, so users can even log into the web server securely on a public computer through PhotoAuth.

This work presents the design and implementation of PhotoAuth on both the phone side and web server sides. To build an efficient and attack-resilient PhotoAuth, we train a deep learning model for address bar detection based on transfer learning, and combine it with OCR output to extract the domain names correctly. We tested the accuracy and efficiency of PhotoAuth under different environments. The results showed that PhotoAuth is an effective technique with good scalability, and it is readily deployable.

In summary, we make the following contributions:

- We propose PhotoAuth, a novel 2FA mechanism based on browser photos to counter real-time phishing attacks and homographic phishing attacks.
- Neither the computer nor the phone needs to pre-install any software or apps to finish 2FA, which differentiates it

from many other 2FA systems. Especially, no Bluetooth or special devices are needed in PhotoAuth and it is compatible with most legacy devices.
- An address bar content recognition mechanism is proposed to counter adversaries from making fake domains at the titles or web content. We labeled and trained the first address bar detection model with 16,454 items.

Note that PhotoAuth is designed and implemented for the case when a user uses a PC browser to log into a website with his phone as his 2nd factor device. We understand that nowadays many people also use the same phones for website login. We will provide a variation of our method to defeat the RTP attacks, as elaborated in Appendix. *The main body of our presentation will focus on the first case which involves both PC and phone.*

II. PRELIMINARIES

A. Real-time Phishing Workflow

Figure 1 shows how a real-time phishing (RTP) works, where the adversary is in the middle of the benign user and the real website. The detailed steps are as follows.

- Step 1: The adversary sets up a fake website (microsoft1.com), which replicates a target website (e.g, microsoft.com), with a mature RTP tool (e.g, Evilginx [4]). With proper settings, the RTP tool can establish the fake website automatically and make it a man-in-the-middle web proxy for microsoft.com. Then the adversary distributes the url of the fake website to users through phishing channels.
- Step 2: A user (referred to as *Bob*) does not pay close attention to the domain name and treats the fake website as the real one because of the same web-page layout. In this example, Bob inputs his Microsoft user name and password to microsoft1.com, the phishing site.
- Step 3: The adversary gets Bob's credentials after Bob submits them to the fake website. Then the RTP tool opens a new session to access the real Microsoft website, and enters Bob's credentials to login. Now the adversary impersonates Bob on the Microsoft website. During the process, the RTP tool automatically modifies appropriate message fields when relaying them between the benign user and the real website so that neither the user or the web server notices the difference from normal use cases.
- Step 4: To verify the login, the Microsoft website sends a one time password (OTP) to Bob's phone.
- Step 5: Bob gets the OTP from the Microsoft website and inputs it into the fake website for authentication.
- Step 6: The adversary gets Bob's OTP and inputs it into his own login session to the Microsoft website as Bob. Finally, he successively logs into Bob's account. To the Microsoft website, it gets a valid login request from the user Bob without knowing the existence of an adversary.

B. Security Model

No 2FA system is attack-proof if we assume the second factor can also be compromised. In our system, we assume neither the user's PC browser nor his smartphone is compromised by the adversary. The link (referred to as *phone link*) between user's smartphone and the website is also secure from interception (e.g., man-in-the-middle attack [8]). We only consider the RTP attack alone. That is, we do not consider phishing mixed with other attacks like DNS spoofing, Domain hijacking, browser hijacking, or system compromises.

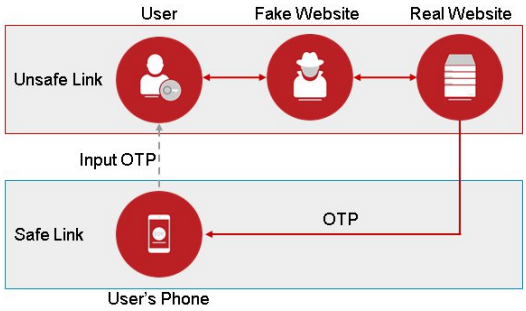


Fig. 1: Real-time Phishing (RTP) Workflow with OTP

C. Design Goals

High Compatibility The system should be compatible with the traditional 2FA system to support most legacy devices. The traditional 2FA system can be upgraded to the new 2FA system easily without changing much on the 2FA workflow.
High Usability For pervasive deployment of our 2FA system, no special hardware other than smartphones will be required. To support 2FA, we will not require the installation of browser plugin/extension; otherwise, users of a public computer (e.g, a public library computer) will not be able to use it. Moreover, some non-tech-savvy users may not know how to install browser plugins/extensions.
High Accuracy The system should provide high accuracy for authentication. In other words, it should incur very low false positive rate and low false negative rate at the same time. False positives happen when benign users failed to pass the 2FA even though they followed the procedure; false negatives happen when the adversary was able to pass the 2FA while impersonating a benign user. As our system assumes the first (i.e., password-based) authentication factor is unreliable, here the high accuracy requirement is only upon our second authentication factor, which should be very difficult or not possible for the adversary to forge.

III. SYSTEM ARCHITECTURE AND DESIGN

A real website and the fake one have different domain names. While the adversary cannot register the same domain name that is owned by the real website, he can register a very similar one (microsoft1.com) or seemingly valid one (e.g., microsoft.com.jp) to confuse the users. As the RTP attack is getting so advanced, in our design we do not rely on end users to detect the attack by themselves. That is, we do not assume users are able to detect a phishing website based on domain names or browser green lock icons. Instead, it is the

job of the web server's side to distinguish benign users from RTP adversaries. In a nutshell, our system, like FIDO2 [5], leverages the third aspect, i.e., domain names, to distinguish benign users and the adversaries, but on the server's side, with a software 2FA mechanism involving smartphones only. No hardware device or pre-install app is needed. Technically speaking, the main task is to deliver (the domain name part of) the website URL in the browser address bar to the authenticated web server in a convenient and secure way. If usability is not a concern, there could be many ways to achieve this goal. For example, with built-in browser support customized for U2F, the browser passes the URL to the local mounted USB U2F device for signing and then transfers it to the web server for verification.

A. System Overview

System Workflow: When a benign user visits a RTP-based phishing website through his PC browser and inputs his credentials, the phishing tool opens a new session to authenticate to the real server as the benign user. The real server determines that this new session needs to pass 2FA, so it requests the user to take a photo of his PC browser and send it back through a PhotoAuth web app (a JavaScript-based server-side webpage loaded by the phone's browser). Indeed, what truly matters is the domain name; in other words, our scheme does not require a full URL and the browser size does not matter either except it is too small to show the domain name part. Based on the domain name the server tells whether the user is accessing the real server itself or a fake website.

Comparing PhotoAuth to one-time password (OTP) under RTP attacks (shown in Figure 1), there are two major differences. First, in OTP, the phone link (i.e., the one between user's phone and server, which is considered safe) is uni-directional, whereas in PhotoAuth, it is bidirectional. Second, in OTP, a user provides the OTP information to his PC browser, which then passes it to the website through an *unsafe* link, whereas in PhotoAuth, the PhotoAuth web app gets the real information from user's PC browser, and then passes it to the web server through the *safe* phone link. Essentially, in PhotoAuth, the trust model is extended from the server itself to user's phone that runs the PhotoAuth web app (i.e., a web page loaded from the server), and the phone takes the role of the authenticator for the website. All sensitive information flows through the safe link rather than the unsafe link. Figure 2 shows the system architecture and detailed workflow of PhotoAuth. On the user side, there is a *PhotoAuth web app* running on the phone browser and triggered by clicking a link pushed from the server. This web app is not a standalone app to be installed on the phone, but a server side web page implemented with HTML5+CSS+Javascript. On the server side, there is a *PhotoAuth module*. In the workflow, steps 1-6 are similar to those steps in RTP workflow (Section 2.1), so we will not re-introduce them except Step 3. In Step 3, the web server's response includes a web cookie, according to the HTTP protocol and today's web security requirement.

In Step 7, the real server consults the PhotoAuth module to decide whether a login request requires the PhotoAuth 2FA. If it is required, the server sends a notification message containing a unique link to the phone (e.g., through push, SMS, email, etc, as long as it is considered a safe link) (Step 8), which, after being clicked on the phone, invokes the PhotoAuth web app in the user's phone. For example, when a user (say Bob) logs into the web server (say microsoft.com) with his PC, the server keeps its PC session id. The server then sends a notification message to his phone, which contains a short link like "microsoft.com/c/6895272031". Each link sent by the server is unique, where the 10-digit number is randomly generated and stored together with Bob's PC session id for future lookup. In this way, this random number is linked for this specific login. Now, when Bob clicks the short link on his phone, his phone browser will load the web page "microsoft.com/c/6895272031", which hosts a simple web app with JavaScript code to start the following task. This is to prevent others from submitting responses while impersonating the user. Under a RTP attack, the attacker will not know the short link for submitting, so this method can easily defeat the attack. Moreover, we can increase the length of the random number to defeat brute force attacks.

In Steps 9 and 10, the PhotoAuth web app guides the user to take a photo and automatically upload it to the server for processing. In Step 11, the PhotoAuth module extracts from the photo the domain name in the address bar of the PC browser based on deep learning and OCR techniques. If the domain name does not match its own one, the user must be visiting a fake website and the authentication request must be from an adversary, so it notifies the the server to deny the authentication request. In the rare case of a false positive, caused by the poor quality of the photo, the user may retry the login process and provide a different photo.

B. User Side Design

On the user's phone, there are generally three ways to receive notifications from the server, e.g.,, through push message, SMS, or email link. When the user clicks the notification on the phone, the default browser will be launched (if not yet), which in turn loads the PhotoAuth web app (from the server) to handle the server challenge. After that, the user only needs to click one button to take a photo and then upload it in the background. In particular, image pre-processing resizes the raw photo image into a smaller and lower resolution image, and further converts the RGB image into a gray-scale one. By leveraging the computational resource of edge devices, we can reduce the bandwidth overhead and hence increase the scalability and throughput of the entire system without sacrificing the accuracy. Detail of compression ratio and bandwidth saving can be found in the evaluation section.

C. Server Side Design

After the server receives the preprocessed photo from a user's phone, it should accurately extract the domain name in the browser address bar. It extracts all the text contents from the entire photo based on Optical Character Recognition (OCR),

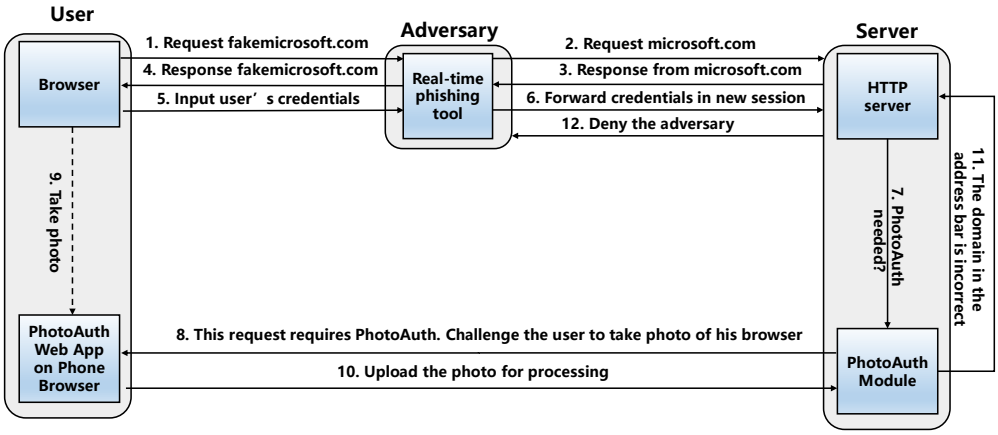
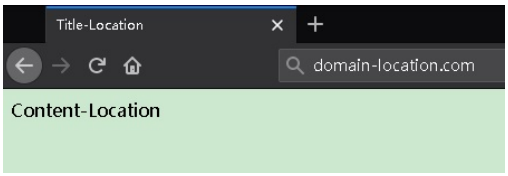


Fig. 2: System overview

and then predict which texts are from the correct domain name (that is, located inside the browser address bar).

Here, the key is to correctly locate the address bar inside a photo, not only for accuracy reason, but more importantly for security reason. In particular, we deal with two types of domain name injection attacks. Figure 3a shows the photograph of a simple webpage with three fields where a domain name may appear. Figure 3b is the corresponding OCR result. The first three lines include all the texts extracted from the photo and the fourth line represents the coordinates of the bounding box for all these texts. The following three lines show the texts and area locations of the title area, address bar area, and web page content area, respectively. Here OCR successfully extracts the texts and outputs their locations. However, it does not tell us which is the text from the address bar. An adversary may want to get around PhotoAuth by injecting the valid domain name either in the title of the webpage or in its content.



(a) Photograph of a simple webpage (upper-left corner)

```
"Title-Location
domain-location.com
Content-Location
"bounds: (8,12),(421,12),(421,99),(8,99)

"Title-Location"bounds: (48,12),(124,12),(124,21),(48,21)

"domain-location.com"bounds: (279,48),(421,48),(421,58),(279,58)

"Content-Location"bounds: (8,87),(139,87),(139,99),(8,99)
```

(b) OCR result for the above example

Fig. 3: An Example for Browser OCR

To prevent the above injection attack, we need to make sure the correct domain is extracted from the OCR texts inside the predicted address bar area. Let A_o and A_b donate the area of a bounding box from OCR and that of a predicted addressbar, respectively. Then we can define the metric *cover rate* in the following formula, which basically indicates how much a text bounding box resulted from OCR overlaps with the predicted address bar. Only if CR is above a threshold value (to be determined in the evaluation section) and the extracted domain name matches with the server's, the user is accepted. In all the other cases, the user needs to retake a photo.

$$Cover\ Rate(CR) = \frac{A_o \cap A_b}{A_o}$$

The other injection attack happens when an attacker embeds a fake browser address bar as an image inside the actual browser, the so-called "picture-in-picture" phishing attack [16] (or a similar case when the user has multiple browsers open). As far as we are aware of, no real websites embed a second address bar in their login page, so it is certainly a very suspicious case. Therefore, in our system, whenever the server detects more than one address bar, it will reject the result and warn the user about phishing attack possibility. Meanwhile, it will request the user to retake a photo with only one address bar, which includes the website the user is actually visiting.

Currently, there is no tool for address bar identification, especially for address bars in photographed browser screens. Address bar identification is not a trivial task because users may take photos of *different types* of browsers at *different angles*, *different distances*, and *different illuminations*. Simply applying some heuristics based algorithms will not work well. Thus, we choose to train a deep learning based object detection model to predict the address bars .

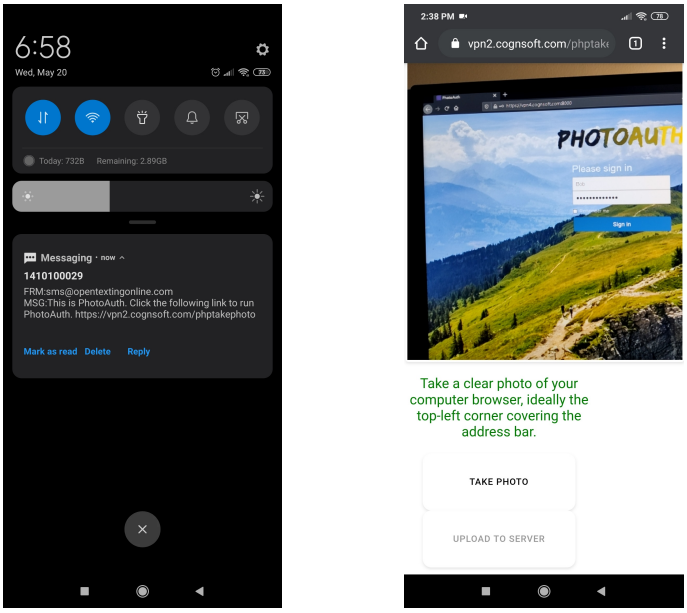
IV. Prototype Implementation

A. Environment

We choose Apache 2.4.39 as the web server, MySQL 5.7.26 as the database, and PHP 7.2.18 as the script language. The hardware configuration of the server is: CPU (I7-8700K Up to

4.7GHz), GPU (GTX 1080Ti 11GB GDDR5X), RAM (32GB 3200MHz) and it runs Windows 10. We use a Samsung S10 phone with 6GB RAM and Android 10.0.

B. Website Front-end Implementation



(a) Phone receive PhotoAuth SMS message

(b) PhotoAuth web app GUI with a taken browser photo

Fig. 4: Screenshots from our prototype implementation

As a proof-of-concept prototype, we place two input boxes in the server login page to obtain the username and password. If both the username and the password are correct, the server passes the logic to the PhotoAuth module for 2FA. Then, the user's phone will receive a notification, which, in our prototype, is an SMS notification through a free online SMS provider [9], as shown in Figure 4a. When the user taps the link inside of the SMS notification message, the default browser in the phone will load the PhotoAuth web app. We develop the PhotoAuth web app with HTML5+CSS+Javascript. The web app requires users to take a photo and then upload it. It will first request users to grant the browser camera-related permission for taking photos. The permission is requested *only once* per website, and the browser makes sure the permission for the web server will not be misused by other websites. Given the permission, the web app will use "navigator.mediaDevices" to invoke the system camera app on the phone (Figure 4b). Then the user takes a photo of the address bar in his PC browser by pressing the "capture" button in the camera app. The captured photo (in jpeg format) is displayed in the middle region of the GUI. If the user is happy with the photo, he can click the "upload" button to upload it to the server. We use the "POST" method of an asynchronous "XMLHttpRequest", a built-in browser object that allows to make HTTP requests in JavaScript, so that the image can be transferred to the server in the background. After the server receives the photo, it checks the domain name located in the address bar of the photo against its own one. If the domain name is correct, it will authorize the user to login; otherwise, it will deny the user.

C. Back-end Implementation

We use python to develop the server side back-end logic. Specifically, we use concurrent programming and multi-threading to achieve multi-tasking for better performance. For OCR, we directly use google vision API, mainly as a proof-of-concept. In practice, the server should deploy its local OCR software (e.g., the open-source PaddleOCR [10] for better privacy and efficiency). We use "from google.cloud import vision" to upload a photo to the google vision server for OCR and get the result. For address bar detection, our main job here is to train a deep learning model. The question is: what are the important features of an address bar to differentiate it from any text-filled rectangle object? Our intuition is to leverage the commonly displayed icons, including the backward/forward arrows, reload and other icons on the left of an address bar. Therefore, for training our model, we need to manually label the address bars with such surrounding areas as the ground truth. Manual labelling of address bars, however, is a time-consuming process. Therefore, for the proof-of-concept purpose, instead of labelling a huge dataset to train an entirely new deep learning model, we adopt transfer learning with a relatively small training set of 15,062 photos.

Specifically, we choose Yolo v3 [20], [21] as the object detection algorithm and adopt one of its pre-trained model [12]. All its parameters and layers are kept as is. We use 50 epochs for transfer learning with learning rate 1e-3, batch size 32, and the Adam optimizer. In this stage, we only update the weights in the last three layers by freezing the weights in all the other layers. Then, in the next 50 epochs, we fine-tune the model with learning rate 1e-4, batch size 8 and Adam optimizer, while unfreezing all the layers to update all weights.

As address bar detection is one type of object detection, we also use the Intersection over Union (IoU) metric to determine whether an address bar is detected correctly. IoU reflects how much the ground truth area and the predicted area overlap. During the training process (and also the testing process), IoU is an input parameter. If one sets the IoU score too big (for example above 0.8), only when the two areas fit very well with each other will it be considered a correct prediction; thus, the precision of the model will be very low. On the other hand, if the IoU score is too small (e.g. 0.2), then the predicted area might be too large (even cover some title areas). Therefore, in our model training, we use the default score 0.5, as used by other object detection models.

V. EVALUATION

In PhotoAuth, authentication accuracy and scalability are very important performance metrics. Next, we evaluate several related performance metrics, including bandwidth use, user non-action waiting time, PhotoAuth's OCR accuracy, address bar detection accuracy, and whole system accuracy.

A. Dataset Composition

To train our transfer-learning based address bar detection algorithm, we took totally 16,454 photos. Our data set is very diverse. First, there are two classes of devices initiating login requests, "desktop", and "laptop". "Desktop" includes monitors of different aspect ratio (e.g., 16:9, 21:9). "Laptop" includes different screen sizes (e.g., 14 inch, 15 inch). Second, because different browsers have different fonts and different styles of address bars, our data set covers many famous browsers, including Chrome, Firefox, Edge, IE, Opera. Third, there are numerous browser skins available (e.g., Chrome web store provides many themes), so it is too complicated to enumerate all browser skins. We chose two different windows color modes, "light" and "dark" themes. These themes change browser skins accordingly. Note that the PhotoAuth web app converts the colorful photos into gray-scale ones before uploading them to the server for OCR. In gray scale, skin personalization makes little difference from that of the light theme or dark theme.

Fourth, the tilting angles and turning angles from the perfect shot angles are within the $[-15°, 15°]$ range and the shot distances are between 30-50 cm. Note that if we request users to take photos at a very close distance (e.g., 10 cm) and only cover the address bar region, the detection accuracy would be close to perfect. In our experiment, however, to make it more challenging, we took the photos from much farther away and the photos covered a very large portion of the PC screens, if not entirely. Furthermore, we also took photos in different environments and illumination settings. Finally, all photos in our training and testing are resized into the resolutions of 1920x1080. 1920x1080 resolution is supported by most cameras today. Therefore, if a user sets his camera at a higher resolution, the detection accuracy will be about the same because of the above resizing.

Among 16,454 photos, we randomly picked 15,062 photos as the training set, 792 photos as the validation set, and 600 as one part of the test set. To test the transferability of our addressbar detection model, we then took 248 photos from 360jisu and brave browsers as the second part of the test set. Our trained model has not seen these two browsers before. In the end, the test dataset contains 848 browser photos, which covers not only the the login page of our own test website but also 57 other popular websites (e.g., Chase, Bank of America). Figure 5 shows the composition of our test dataset.

B. Bandwidth Use

To save bandwidth for photo uploading, in our prototype the PhotoAuth web app compresses the photos into small size, gray-scale ones. As OCR and objection detection algorithms do the same for their input, as long as the resolution of the compressed photos is good enough, it will not introduce errors into our system while saving bandwidth. Based on our empirical study, we find the resolution of 1920x1440 (or 1920x1080 depending on the camera aspect ratio) is sufficient when taking photos a meter away (800x600 when 50cm away). The size of a jpeg file at this resolution is around 160KB.

C. User's Non-action Waiting Time

To end users, the latency of our 2FA system is an important performance metric to care about, because it reflects the usability of our system. As users may spend different time to take photos, we will not consider the part of latency due to user's action. Instead, we test user's *non-action waiting time*, which is mainly composed of photo uploading time and server side processing time. On the server side, OCR and address bar predication are carried out in parallel. Here we only count the OCR time, because our tests showed that it was always more than that for address bar prediction. We ignore the other types of waiting time in the system (e.g., the time for the server to locate the domain name based on the outcome of OCR and address bar predication, transmission time of the server's response), because they are negligible compared to the two listed above.

Figure 6 shows the uploading transmission time and overall non-action waiting time in both WiFi and LTE settings. The bandwidth of LTE is typically smaller than WiFi, so the transmission time of LTE (in our experiment the median is 1393 ms) is higher than that of WiFi (median 80 ms). Moreover, because in LTE many users share the same base station, so its transmission time fluctuates more than in WiFi case. The server side photo processing time does not depend on the uploading channel, either via LTE or via WiFi. In the WiFi setting, such processing time domains the overall waiting time, whereas in the LTE setting, the processing time accounts for roughly half of the waiting time.

D. Domain Name OCR Precision and Recall

PhotoAuth relies on accurate text output from OCR (specifically the texts from the address bar) to decide whether to authorize an authentication request. In this section, we test the precision and recall of OCR under various real-world settings to see how robust our system is to extract the domain names. Note that in the case when precision and recall are not perfect, it does not mean an adversary can bypass our system. We will elaborate on this point in Section V-F.

In traditional object detection, an area is called a predicted area if the confidence score of detection is above a threshold. If the IOU (Intersection over Union) of the predicted area and the ground-truth area is above a specific threshold, it is a true positive; otherwise it is a false positive. If no area is predicted, it is a false negative.

Domain name OCR is different from traditional object detection in that it has two stages: *detection* and *recognition*. A true positive occurs only when *both* the right area (in our case, the address bar area) in an image is detected and the domain name (not the entire URL) inside the area is correctly recognized. Otherwise, we may have false negatives (when the area is not detected) or false positives (when the domain names are wrongly recognized).

Figure 7a shows the results with the 848 images (including 56 unique domains) introduced in our test set. The recall was 100%. This means, as long as there was an address bar in the photo, OCR could detect the text area accurately. The OCR

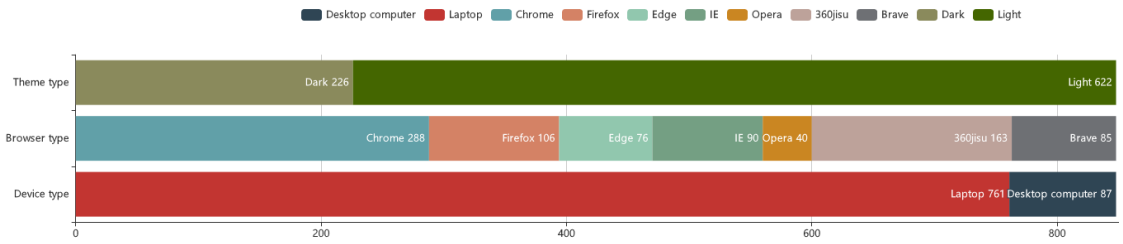


Fig. 5: Test set composition

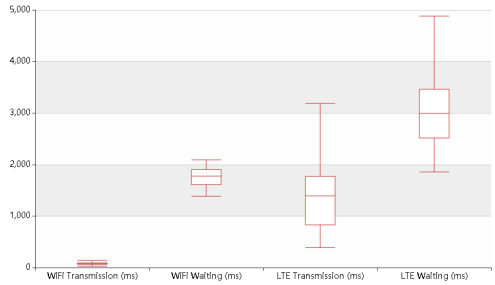
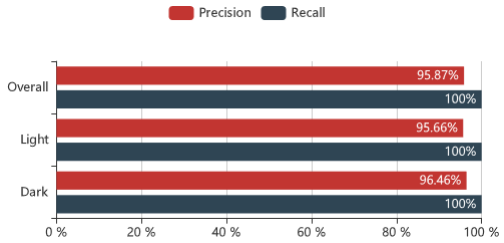
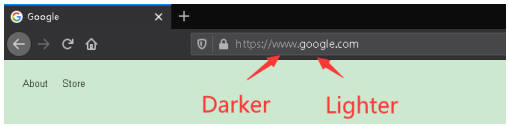


Fig. 6: User's non-action waiting time



(a) OCR precision and recall for both light and dark color modes and the overall



(b) OCR false positive example

Fig. 7: OCR Experiment

generated 35 false positives. There were two types of false positives. First, the dot '.' in the domain names was too small to be recognized. Second, certain letters were mis-recognized, e.g., in one case 'o' was recognized as 'a' and the other case 'o' as 'e'. The overall precision was 95.87%.

Figure 7b shows a failing example of OCR with the dark color mode of a browser. This is because the browser automatically made the "www." part of a hostname (e.g., "www.google.com") darker, causing the OCR to miss '.'. Note that even in this challenging setting, this type of error only happened occasionally.

Note that the precision of domain name OCR can be greatly improved with better quality photos. We believe that in practice, once users understand that the 2FA is based on the address bar content, they can naturally take photos at smaller distances while focusing on the address bars instead of the entire PC screen. We did an additional test with the domain names of Alexa top 50 websites [1]. We randomly changed 11 (out of 37) 'o's into '0's, 5 (out of 17) 'l's into '1's in the names. Differently, this time we took photos at the distance of about 20 cm and focused on the top-left corner. In the end, among the 527 characters in all these domains, there was only a single recognition error – one '1' was recognized as 'l'. The accuracy can improve further with a smaller distance.

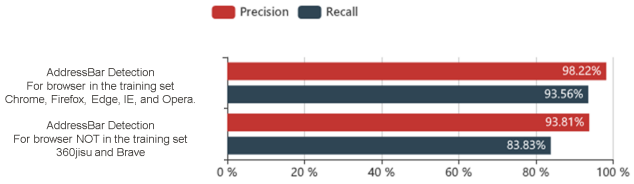
E. Address Bar Detection Precision and Recall

With the dataset introduced in Section V-A, we train our addressbar detection model and measure its performance. Figure 8a shows that the precision and recall of addressbar detection for known browsers (i.e., covered in the training set) are 98.22% and 93.56%, respectively. The precision and recall of addressbar detection for unknown browsers are 93.81% and 83.83%, respectively, which look reasonably good. This relatively lower accuracy is not surprising because the address bar features of Brave and 360jisu are different from that of the other five browsers. For wider deployment of our system, we believe a better approach is to train a model with additional types of browsers, for example, top 10 browsers.

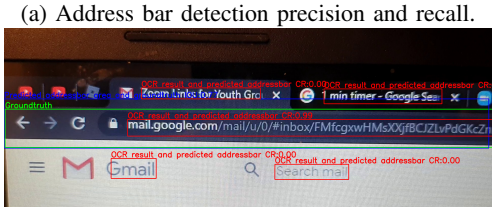
Figure 8b shows an example output. The green bounding box is the labelled ground truth address bar area, and the blue one is the address bar area predicted by our deep learning model. One may notice that the ground truth area covers not only the address bar, but also commonly displayed icons on the left of the address bar as they provide the important features for address bar detection.

As an object detection task, **not** an object classification task, our model either outputs a predicted address bar, or outputs nothing. It *does not* know the ground truth area, although in our evaluation we manually labelled the ground truth areas to measure the detection accuracy. When no address bar is predicted in our testing, it is clearly a false negative. Now, when a address bar is predicted, it can be either a real one (IOU score above 0.5) or a false one (IOU score below 0.5). In the example in Figure 8b, the actual IOU is 0.77, which is above the threshold, so it is a true positive case. If the actual IOU is under 0.5, which means the predicted area is much different from the ground truth area, it will be counted as a false positive case with respect to object detection.

In practice, both false positive and false negative errors could cause the failure for the server to extract domain names correctly, and hence photo re-takes would be necessary. Also, in practice, our system may request users to take a photo that only focuses on the top-left corner of the browser at a closer distance. This will make address bar detection much easier.



(a) Address bar detection precision and recall.



(b) A true positive case

Fig. 8: Address bar detection

Based on the test dataset, we found that the median address bar detection time for one photo is 71 ms (the maximum 88 ms). It does not lag the whole system when parallelized with OCR because OCR takes at least 1 second to return the result.

F. Whole System Evaluation

In the whole system evaluation, we combine the detection results of OCR and address bar detection and report the final results. Figure 8b shows an example with five bounding boxes for OCR texts, two for the texts in top titles, one for the URL in the address bar, and two others for texts in page content, they are all in red rectangles. Here we do not show the areas for texts extracted from the web page. The blue rectangle is the predicted address bar. For each red rectangle above, we calculated the cover rate (CR) (defined in Section III-C), and got 0.32 (for the top-right title) and 0.99 (for the URL in the addressbar) and 0 for the rest. After analyzing all the data, we found that the CR threshold of 0.8 can distinguish texts in webpage content/title from texts in the address bar very well.

Finally, we measure the error of our entire system with the metric named *retake rate*. Despite the cause of errors, as long as the server failed to recognize the domain name correctly, in our measurement it was counted as a retake case, where the user is requested to take a photo again. We used the 600 photos in our test set to measure the retake rate while setting CR=0.8. The retake rate was 6.83%. It is relatively high mainly because the low quality of the photos, which introduced errors into OCR and address bar detection. In practice, a user can easily fix the problem by taking a photo at a closer distance, at the right angle and focusing on the top-left corner. In our testing, with the CR threshold of 0.8, there was not a single case where a title or any texts inside a webpage was mis-identified as a domain name. Only the address bar areas have been detected, which shows that the system worked as expected.

Finally, we also conducted a preliminary user study over a demo version of PhotoAuth with 33 participants (IRB approved). The 33 participants showed a positive attitude on the usability f PhotoAuth (e.g., over 50% considered it as convenient as the 2FA they have used before). Due to page limit, we omit the details here.

VI. DISCUSSIONS

Evasion Attacks: An attacker may attempt to evade our system in different ways. First, as PhotoAuth relies on OCR to extract correct domain names, an attacker may register visually similar domain names (e.g., by registering g0og1e.com replacing 'o' with '0', 'l' with '1', also called typosquatting domain names). This is one type of *homograph attack* [7]. In Section V-D, we have already shown that, with better-taken photos, the chance for this attack to succeed could be very low (1 error out of 527 characters in Alexa Top 50 domain names). As the OCR technique is advancing, such errors will be further reduced. Moreover, even if the attacker has successfully tricked a user into trusting his website (e.g., through phishing emails), he does not have the control over how the user takes photos; therefore, an OCR error may rarely result in a valid domain name, not to mention the case of perfectly turning into the target domain name.

Another type of homograph attacks [7], [15] explore unicode for better success. Not only a human user cannot recognize such Unicode letters easily, but the state-of-the-art OCR tools cannot recognize them correctly either. Fortunately, all major browsers only allow ASCII letters in the address bar, and they automatically convert the Unicode letters into the ASCII letters (Punycode). For this reason, such homograph attacks will not succeed.

The second type of evasion attacks is a potential redirection attack against the workflow of our system. Specifically, after the user input his login credentials (Step 5 in Figure 2), an attacker may try to redirect the user to microsoft.com, so that the user with PhotoAuth will take the picture on the genuine domain. However, this attack will not work. Recall that in Step 3, the real server (here "microsoft.com") sends a web cookie to the attacker. If the attacker replays (forwards) this cookie to user's browser (Step 4), the user browser will store it for the fake webserver because it was received from the fake server instead of the real web server. Based on the same origin policy (SOP), the user's browser will *not* send this web cookie to the real web server if the user is instead redirected to the real web page that displays the same real login user interface. As a result, the real server will not receive a (valid) cookie from the user's browser, hence denying the login process.

VII. RELATED WORK

A. Industrial Solutions

Duo Push [3] is a phone application that receives push information when the user sends a request in the browser login

page and the user taps a button to respond. Unfortunately, it is vulnerable to the RTP attacks. In the 2nd factor authentication phase, the adversary can deceive the user press the "Approve" button. The user may think the 2FA is for himself to authenticate to the website, but the truth is that the 2FA is for the adversary to get authentication. Recently Google released a new software-based 2FA tool leveraging phone's built-in security key [6]. It requires pre-installed phone app to generate the key, special built-in browser support, and Bluetooth (or NFC) to establish a secure channel between the computer and the phone, such requirements could restrict its usability. In 2017 the FIDO Alliance proposed a Universal 2nd Factor (U2F) protocol [11], where end users carry a single U2F device which works with any relying party supporting the protocol. Later, the FIDO Alliance proposed FIDO 2 [5]. U2F/FIDO2, based on public-key cryptography, can counter RTP attacks very well.

While the industrial solutions look promising to solve the RTP attacks, it may take a long time to be widely deployed because of several possible factors such as cost and usability issues [2]. Other use options may require pairing between phone and PC, or BlueTooth or NFC. The concepts and procedure for deploying U2F/FIDO2 could still look complicated to some non-tech-savvy users because of the needed registration, installation or configuration. To this end, alternative solutions are still very needed.

B. Academic 2FA Solutions

Shirvanian et al., [23] proposed a 2FA system based on mix-bandwidth devices. Czeskis et al., [14] proposed a 2FA system named PhoneAuth. Its overall protocol shares the same same spirit with U2F protocol except it involves a smartphone instead of a USB dongle. Parno et al., [19] proposed a system to establish a secure bookmark on the phone side to control the authentication. Azimpourkivi et al. [13] introduced a camera-based 2FA system called Pixie, which establishes trust between a user and his web server based on both the knowledge and possession of an an arbitrary physical object. However, the lack of a binding between the trinket and the website the user is visiting leaves the system vulnerable to RTP attacks.

Karapanos et al., [17] proposed a 2FA system based on the ambient sound. Basically, both user's browser and user's phone record the ambient sound at the same time. If the sound signals are much different, it is likely an attack case. The system can handles RTP attacks with the support of Bluetooth and microphone recording.Ulqinaku et al. [24] proposed 2FA-PP, which leverages a Web Bluetooth API, to create a secure Bluetooth connection between a website and the user's smartphone that runs a special mobile app. To defeat phishing attacks, it leverages network latency measurements to tell if the user is connected to the legitimate server or to the attacker's site. The system has high accuracy when the attackers are not located within the same region as the victim.

VIII. CONCLUSION

In this paper, we proposed PhotoAuth, a 2FA system to defend against real-time phishing (RTP) attacks. In PhotoAuth, a user takes a photo of the PC browser with the address bar area, and uploads the photo to the server. The server automatically extracts the domain name information from the address bar and detects fake domain names. PhotoAuth is easy to use and also compatible with the traditional 2FA system to support most legacy devices. It does not require special hardware (except user's phone), We prototyped the system and tested it in various environment settings and with multiple types of browsers. The results showed that PhotoAuth is able to effectively prevent and detect attacks.

REFERENCES

[1] "Alexa top 50 sites (freely accessible list)," https://www.alexa.com/topsites.

[2] "The authentication solution government has been slow to adopt." https://www.fifthdomain.com/civilian/2018/02/01/the-authentication-solution-government-has-been-slow-to-adopt/.

[3] "Duo push," https://duo.com/product/trusted-users/two-factor-authentication.

[4] "Evilginx," https://github.com/kgretzky/evilginx2/.

[5] "Fido 2," https://fidoalliance.org/fido2/.

[6] "Google phone's built-in security key," https://support.google.com/accounts/answer/9289445.

[7] "Idn homograph attack," https://en.wikipedia.org/wiki/IDN_homograph_attack.

[8] "Man-in-the-middle attack," https://en.wikipedia.org/wiki/Man-in-the-middle_attack.

[9] "Open texting online," https://www.opentextingonline.com/.

[10] "Paddle ocr," https://github.com/PaddlePaddle/PaddleOCR, accessed June 22, 2021.

[11] "U2f," https://www.yubico.com/services-with-yubikey/fido-u2f/.

[12] "Yolo v3 weights," https://pjreddie.com/media/files/yolov3.weights.

[13] M. Azimpourkivi, U. Topkara, and B. Carbunar, "Camera based two factor authentication through mobile and wearable devices," *Proceedings of ACM Ubicomp*, 2017.

[14] A. Czeskis, M. Dietz, T. Kohno, D. Wallach, and D. Balfanz, "Strengthening user authentication through opportunistic cryptographic identity assertions," in *Proceedings of the ACM CCS*. ACM, 2012.

[15] E. Gabrilovich and A. Gontmakher, "The homograph attack," *Commun. ACM*, vol. 45, no. 2, 2002.

[16] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth, "An evaluation of extended validation and picture-in-picture phishing attacks," in *1st International Workshop on Usable Security, USEC, 2007*, S. Dietrich and R. Dhamija, Eds.

[17] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun, "Sound-proof: usable two-factor authentication based on ambient sound," in *USENIX Security*, 2015.

[18] E. Medvet, E. Kirda, and C. Kruegel, "Visual-similarity-based phishing detection," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*. ACM, 2008.

[19] B. Parno, C. Kuo, and A. Perrig, "Phoolproof phishing prevention," in *International Conference on Financial Cryptography and Data Security*. Springer, 2006.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[21] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[22] A. P. Rosiello, E. Kirda, F. Ferrandi *et al.*, "A layout-similarity-based approach for detecting phishing pages," in *SecureComm*. IEEE, 2007.

[23] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan, "Two-factor authentication resilient to server compromise using mix-bandwidth devices." in *NDSS*, 2014.

[24] E. Ulqinaku, D. Lain, and S. Capkun, "2fa-pp: 2nd factor phishing prevention," in *Proceedings of the ACM WiSec*, 2019.