Practical Mitigation against Phishing Attacks

Panjapol Kongkhieo, Chanathip Namprempre* and Chaipat Suwannaphum
Thammasat School of Engineering, Thammasat University, Klong Luang, Pathumthani 12120, Thailand

Abstract

Phishing attacks are effective and prevalent. The main purpose of these attacks is to lure users to expose their secret credentials (e.g. username-password pairs). In this paper, we first provide an analysis of the approach taken by a local bank to prevent phishing attacks and show that it is completely insecure. Then, we describe our proof-of-concept implementation of a practical alternative to authentication that can mitigate phishing attacks. The system implements two-factor authentication through the use of Google Authenticator, a mobile application that in turn implements the well-known one-time password algorithms specified in RFC 6238 and RFC 4226. Finally, we discuss other practical means to mitigate phishing attacks. This work is a part of an undergraduate thesis project.

Keywords: two-factor authentication, phishing, server authentication, client authentication

1. Introduction

Phishing refers to the type of attacks whose goal is to obtain users' credentials, for example, usernames and passwords, that have been registered for certain target websites. In a perfect world, phishing would not be a problem since many cryptographic schemes and protocols that can prevent this type of attacks already exist and have been implemented [8]. Specifically, authenticated key exchange protocols secure against active attacks can be used

- to authenticate a server to a user's client device based on a valid server's certificate and
- to authenticate a user's client device to a server based on a valid client's certificate.

However, these protocols are rarely used to their full potential. In reality, two sources of problems, among others, persist. [1] Most users and client devices do not have certificates, and [2] most users ignore notifications indicating that server certificate validation has failed. The former gives rise to the prevailing practice of using username-password pairs to authenticate users to websites. The latter is mitigated by educating users to pay attention to sometimes obscure error messages displayed by the browsers when server certificate validation fails. The prevalence of successful phishing attacks proves that these approaches to addressing the problem are lacking.

The banking industry is one of the most lucrative targets for phishing attacks. The mitigation techniques of individual banks vary. Some banks employ two factor authentication, a practice which requires users to use their username-password pairs and an additional piece of information from another source to authenticate themselves to the bank servers. Often times, the additional information is obtained from a hardware token or the user's phone in the form of a "One-Time Password" (OTP) code that has been sent to the phone. Some banks, however, opt for less costly solutions in order to avoid the need to distribute an additional piece of hardware or the need to ask users to perform a complicated sequence of actions simply to perform a simple task such as logging in to view their account balance.

In a 2016 news article, a system deployed by a local Thai bank in order to mitigate phishing attacks without requiring a hardware token or an OTP is described [4]. In order to login to the online banking system, a user would first send his/her username to the server. A "secure word" that has been registered with the bank beforehand will then be sent back to the browser. The user is supposed to ensure that this secure word is correct before entering his/her password. The reasoning seems to be that, since a phishing adversary does not know a correct secure word for a particular user, it should not be able to fool the user into submitting his/her password to it.

In this paper, we argue why this approach to addressing phishing attacks is not only inadequate but could exacerbate the problem. We then suggest alternatives that the local bank can use in place of the current design to mitigate the damages of successful phishing attacks.

2. An Insecure Approach: Server authentication with secure words

We begin by describing how a local Thai bank web application for online banking works. We discuss only the parts that are relevant to the phishing mitigation attempt based on the use of "secure words" here. We note here that all communication between the bank web application server and the web browser is over SSL/TLS (henceforth referred to as simply TLS) with unilateral authentication of the server by the client device turned on and that the server uses a valid certificate recognized by common browsers. In short, a user *can* look for the lock icon on the left most edge of the URL address bar and can watch out for any invalid-certificate warning to determine whether he is conversing with an authentic server. He may or may not do so, however, as will be discussed later in this section.

INITIAL REGISTRATION. The registration process is performed online and is based on an "ePIN" that has been registered with the bank. A customer can set an ePIN using an ATM card and the corresponding ATM PIN at an ATM machine or by going to a bank branch to tie a phone number to his account so that an ePIN can be sent to it via an SMS message. Once an ePIN is established for an account, the customer can use it along with other basic information (such as a citizen identification number) to set his username, password and secure word.

LOGIN PROCESS. In order to login, a user first submits his username to the server. The server then responds with the user's "secure word" albeit as an image clearly showing the text of the secure word.

Figure 1 An example of the image of a secure word returned by the server of the local bank.

(See Figure 1 for an example of the image of a secure word.) The user checks that the secure word is the same as the one he has chosen at registration time and submits his password if so. The system allows him to login if and only if his password is correct.

2.1 An offline attack

Clearly, since the server indiscriminately replies to any request for a secure word, an attacker can request for the secure word for any username of his choice. The question is whether this weakness can be exploited in an automated manner to obtain a large number of username-password pairs via phishing attacks. First, we describe the structure of the attack.

HARVESTING SECURE WORDS. An attacker can harvest secure words without soliciting any help from a user. We referred to this type of harvest as an *offline harvesting* attack. An attacker can proceed as follows.

1) Create a database of legitimate usernames.
2) Submit usernames in the list to the server one by one and obtain images correspondding to the secure words for the usernames.
3) Extract secure words from the images thus obtained.
4) Add the username-secure word pairs to its database.

We note that step 3 may be omitted if the attacker simply stores the images of the words rather than the actual text in which case it would simply add the username-image pairs to the database instead. In order to simplify our discussion, we henceforth assume that the attacker aims to extract the actual text of the secure words.

GETTING USERS TO VISIT A MALICIOUS SERVER. An attacker can use common phishing methods against the bank customers. The goal is to get each user to click on a link that would lead to the attacker's server, rather than that of the bank, in order to obtain the username-password pair of the user. Different phishing techniques use different channels to send the link. Common ones are visually deceiving URLs (also known as homograph attacks) such as bank0ne.com attempting to pose as bankone.com, email messages purported to be from the bank, short messages transmitted via mobile messaging applications such as Line or Short Message Service (SMS, thus giving rise to the term "smishing"), and links to website that manage to come up first as a result of Google search perhaps through the use of aggressive

search engine optimization (SEO) techniques. It is well known that these techniques have proven to be effective [15].

OBTAINING USERNAME AND PASSWORD. Once a user clicks on the link to a malicious server, the transaction will likely proceed as follows.

1) The user submits his username to the server expecting his secure word in response.
2) The attacker retrieves the secure word for the given username from his database, creates an image containing the secure word, and sends the result as part of the reply.
3) Upon seeing his secure word being displayed correctly, the user submits his password to the malicious server. The attacker now has all the information he needs to login to the bank system as the user.

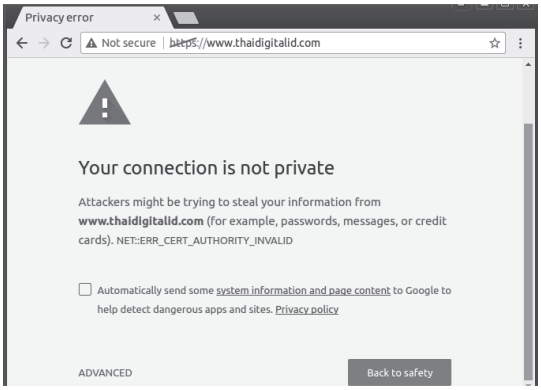

Figure 2 An example of a warning message indicating that the server certificate validation has failed.

We point out that, since this communication is performed over TLS, the user should have noticed, before submitting his username in step 1, that the server certificate check has failed. The manifestation of this failure vary depending on the type of the browser being used. Figure 2 shows an example on Chrome version 54.0.2840.59m. It is our understanding that the local bank's motivation for using secure words is that users are too naive to notice this type of warning.

AUTOMATING THE ATTACK. An attempt to automate the harvesting phase of the attack may be met with the following obstacles.

1) The attacker does not necessarily know the usernames of the targets.
2) The attacker may be rate-limited by the server if the latter implements a mechanism that blocks repeated requests from particular clients based on, say, an IP address.
3) The attacker needs to process the returned images of the secure words in order to obtain the actual text.

We argue that these obstacles are not particularly difficult to overcome. In regards to the first obstacle, there are many well known ways for attackers to harvest usernames [17]. Besides, usernames are ostensibly not viewed as secrets. Users most likely will not, and arguably should not be expected to, treat them as secrets. As for the second obstacle, attackers can avoid sending requests from a single source by exploiting botnets. Such attacks have been mounted in the wild and have proven to be effective and lucrative [18]. As for the third obstacle, OCR accuracy is currently close to 100%. In fact, researchers have moved on to more challenging problems such as visual CAPTCHA recognition for images embedded with noises [14]. The accuracy rate for CAPTCHAs has reached 90% using techniques based on active deep learning [16].

2.2 An online Attack

The database containing triples of username, secure word, and password can also be built using an *online harvesting* attack. We describe the attack here, highlighting only the parts that are different from the offline attack outlined in the previous section.

Rather than gathering secure words based on a list of common usernames, an attacker can enlist the user's help directly at phishing time. Figure 3 illustrates the structure of the attack. Here,

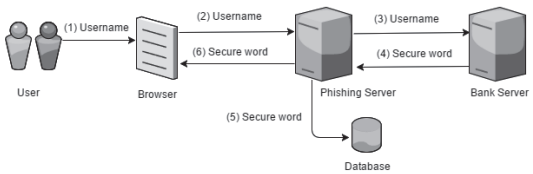

Figure 3 Online harvesting of secure words via a man-in-the-middle attack. The numbers indicate the sequence of message flows

the attacker gets a user to visit his malicious server using various phishing techniques, mounts a man-in-the-middle type attack asking the user to send his username, then passes the username along to the real bank server. Upon obtaining a response containing the (image of the) secure word for the user, the attacker simply sends it to the user. Seeing that the secure word is correct, the user then sends the malicious server his username and password.

Of course, this whole sequence needs be done quickly enough so as to avoid triggering suspicion. To this end, the attacker only needs to be fast enough to beat the usual network delay.

2.3 Further comments

We argue here that the incorporation of secure words worsens the overall security of the system. First, the term "secure word" is an unfortunate choice. It is ambiguous and misleading. Specifically, users may be inclined to view it as a word that will be kept "secure" by the bank where "secure" here most likely would be taken to mean "secret" Imagine an inexperienced user trying to set his secure word, username and password at registration time. He may naively choose to set the first and the last to the very same word. The result is a system that ends up being much less secure than before the concept of secure words is incorporated.

Second, the news article implies that other banks should follow suit and use secure words to prevent phishing attacks against their systems. Unfortunately, if they do, they will also be vulnerable to attacks described here. Furthermore, it is well established that many users reuse their passwords. Thus, it is likely that they would also reuse their secure words. Consequently, a database of username, secure word, password triples for one bank becomes a valuable piece of data that can be lucratively traded to enable attacks against other banks.

Third, usability is worse. Users are already overburdened with confusing instructions on what to watch out for while banking online. Passwords need to be hard to guess yet memorable, kept secret, and not written down. The URL of the bank server must be exactly correct as publicized. Now, they are told to come up with an additional piece of information whose security role is most likely unclear to them. This additional burden takes attention away from what is more important: the server certificate must be valid. Worse yet, telling the users that secure words help protect them against phishing may lead to a misunderstanding that the server certificate no longer needs to be valid. Users who observe that a server certificate validation has failed may incorrectly assume that they may still proceed since there is a separate mechanism to prevent phishing that keep them safe.

Disclaimer. We stress here that we did not attempt to implement a proof of concept attack against the local bank system nor did we attempt to verify whether the bank has put in place mechanisms to rate-limit the number of attempts an attacker can submit secure words. Our analysis is a solely conceptual exercise as we wish to avoid overstepping the ambiguous boundaries set in the Computer Crime Act [3].

3. Device-Assisted Two-Factor Authentication

A different approach to mitigate phishing can be implemented by requiring that, in addition to a password, user authentication relies on another piece of information that is ever changing. The hope is that, since this piece of information changes over time, if an adversary manages to obtain it (e.g. by fooling the user into believing that its malicious server is the real one), the information would be useless for future authentication. This piece of information is often referred to as a *one-time password.*

3.1 Components required

HARDWARE TOKEN. An inexpensive hardware token, often referred to as a key fob, is typically a chip-on-board design with a button that, once pressed, causes a sequence of digits to be displayed on a small screen. This sequence plays the role of a one-time password. Figure 4 shows an example of such a device.

Figure 4 RSA SecurID key fob.

Another variation on this idea is to exploit the fact that virtually everyone now carries a smartphone. A mobile application can be installed on the phone and effectively turns the phone into a hardware token. Google Authenticator is one example of such an application [5]. We implement a two-factor authentication system based on Google Authenticator and describe it in this

paper. First, we describe the building blocks required for this type of solution.

MESSAGE AUTHENTICATION CODES. A message authentication codes (MAC) scheme is a symmetric-key based cryptographic primitive consisting of three algorithms (KG, MAC, VF). The key generation algorithm KG outputs a fixed-length bitstring K chosen uniform randomly from all bit-strings of that length. The MAC generation algorithm MAC takes a secret key K and a message M and outputs a MAC, also known as a tag, T. The verification algorithm VF takes a secret key K and a message-tag pair (M, T) and outputs 1 if the tag is valid for M under K. It returns 0 otherwise.

A secure MAC scheme ensures that it is difficult for an adversary to forge a message-tag pair that would pass verification under a secret key that the adversary does not know, even if it is allowed to see sample tags under the same secret key for messages of its choice. Finally, the winning forgery is not allowed to be one of the sample message-tag pair for which it has previously obtained, or else no MAC schemes could ever be considered secure. This security notion is commonly known as *unforgeability against chosen message attacks.*

One popular secure MAC scheme is HMAC-SHA1. This construction is based on the hash function SHA1. Note that, although SHA1 is no longer considered to be a collision-resistant hash function, the HMAC-SHA1 construction is still a secure MAC scheme [6].

TIME-BASED ONE-TIME PASSWORD ALGORITHM (TOTP). As the name suggests, a one-time password is a password that is used once then is disregarded. One common way to generate a one-time password is to simply compute a tag T using the current time ts as an input message. Now, if the receiver is given both ts and T, then it can easily recompute the tag for ts then compare it with T and accept the tag as valid if and only if the values match. However, for ease of use in many systems, the receiver is often assumed to have a clock drift that is small enough that simple rounding can ensure that the receiver and the sender end up using the same timestamp as an input message. This assumption allows the receiver to use its clock with rounding to obtain the timestamp ts, then recompute the tag and check whether the recomputed value and the tag T that it has received are the same.

The Time-based One-Time Password algorithm (TOTP) described in [13] is based on this idea. Let K be a secret MAC key, and let Truncate be a function that returns "user-friendly values" by truncating and converting its input. The tag computation is defined as

$$\text{TOTP}(K) = \text{Truncate}(\text{HMAC-SHA1}(K, ts))$$

where $ts = \lfloor (t - t_0)/X \rfloor$ t is the current time, t_0 is the Unix epoch (00:00:00 UTC on January 1st, 1970), and X is the number of seconds per time step. The default value for X is set to 30.

3.2 Our implementation exploiting Google Authenticator

The proof-of-concept system that we have implemented operates as follows.

SETUP. The user downloads and installs the Google Authenticator mobile application on a smartphone. This can be done, for example, through Google Play or Apple App Store, if the phone runs Android or iOS, respectively.

REGISTRATION. Each user carries out this process only once to enable two-factor authentication thereafter. Using a web browser on a computer, the user loads the registration page to send a registration request to the web server S for which the user wishes to enable two-factor authentication, e.g. an online banking web application. Upon receiving the request, the server S generates a cryptographic secret key K (typically a string of pseudorandomly chosen bits) then sends it to a server G maintained by Google for this purpose. The Google server G then base 32-encodes K along with additional information such as the username and the issuer name (e.g. the service with which the account is to be associated) to create a QR code and sends the result back to the web server S, which in turn forwards it to the browser to display. The user can then use the Google Authenticator application on the smartphone to scan the QR code. The phone then base 32-decodes the QR code and stores K for future use. This completes the registration process.

LOGIN PROCESS. A user who wants to login to the web application on the server S uses a web browser to connect to S meanwhile ensuring that the server certificate has been properly validated. The user then types in the username and password and waits for the browser to display a form requesting the one-time password code. (Here, the server S terminates the connection if the username-password pair is incorrect, and consequently the browser will not display the form requesting a one-time password

in this case.) Using the Google Authenticator application on the smartphone, the user reads off the code and types it into the form on the web browser. The server *S* then verifies the correctness of the code and accepts the user authentication request if and only if the code affirmatively passes the verification process. Figure 5 shows the details of the message flows.

SECURITY ANALYSIS. Since HMAC-SHA1 is a secure MAC (specifically, it is secure under the notion of unforgeability against chosen message attacks) [6] and since the input to the MAC generation algorithm is essentially a nonce ("number used only once"), it is hard for an attacker to forge a correct TOTP for the timestamp that the server will use during verification. Also, a phishing attacker who has successfully fooled the user into submitting his username, password, and TOTP will be able to use the information to login successfully as the user *only* during the same time period. This means that any successful attack must be performed in real time, thus limiting the value of the triple that the attacker has collected.

We emphasize that the main idea behind the mitigation of phishing attacks described in this section is the enforcement that the credentials now include not only the username-password pairs, but also the time-dependent OTPs. This does not relieve the users of the responsibility of ensuring that they are connecting to legitimate servers. In particular, the users are still required to be vigilant in checking that there are no certificate validation failures (by observing whether their web browsers issue

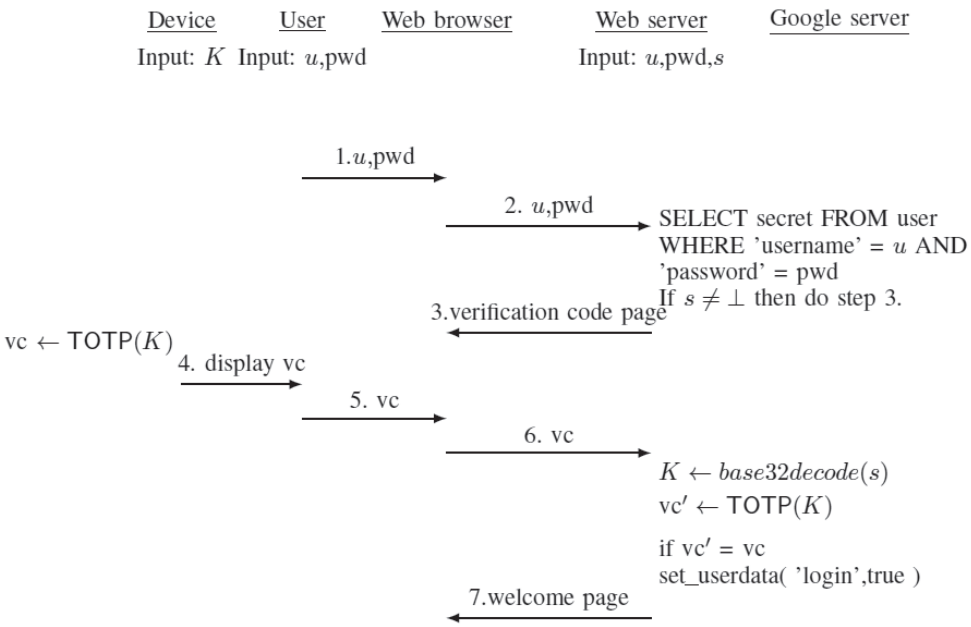

Figure 5 Login protocol for our implementation exploiting Google Authenticator. The first line shows the names of the participants in the protocol. The second line shows the initial inputs of the participants. The variables *K*, *u*, pwd, and *s* refer to the cryptographic secret key, the username, the password, and the base32 encoding of *K*, respectively. TOTP (*K*) refers to the TOTP algorithm, and *base32decode* refers to the base32 decoding algorithm.

warnings) during both the registration and the login processes. The contribution of this mitigation approach is in ensuring that the information obtained by an adversary, in the event that the users do not notice certificate validation failures, is only useful for the short time period during which the OTPs thus obtained are valid.

4. Other approaches for phishing mitigation

Given that phishing has been a long-standing problem in web application security, many approaches have been devised to address the problem. Two of the most notable ones are browser extensions and passwordless login mechanisms.

ANTI-PHISHING BROWSER EXTENSIONS. Many browser extensions and browser toolbars exist to address phishing attacks. The main idea behind many of these tools is the following. Given a URL that a user tries to access, the tools attempts to determine how likely it is a phishing website and raises an alarm if the likelihood is sufficiently high. The methods used to evaluate the likelihood vary as do the effectiveness. Tools in this category include Dynamic Security Skin [7], eBay's Account Guard Toolbar [1], GoldPhish ([9], and TrustBar [10].

PASSWORDLESS LOGIN MECHANISMS. The most sought-after type of credentials for phishing attacks are username-password pairs. Not only can they be stolen through phishing, passwords are often easy to guess as well. Given many weaknesses associated with the use of passwords, many key industry players have joined forces to push for a move away from passwords. The most notable such collaboration is that of the FIDO Alliance. They have pushed for two main protocol suites, namely the Universal 2nd Factor Authentication (U2F) and the Universal Authentication Framework (UAF) [2]. The former essentially takes the two-factor authentication approach similar to our recommendation in this paper. The latter, however, advocates that passwords be eliminated altogether and replaced by the use of local, biometric-based authentication that is to be performed by the user's own personal devices such as a smartphone. We note that we have successfully implemented a proof-of-concept system that allows users to be authenticated via their android smartphone in order to gain access to a web application [12]. The system is simple to use but does require users to have access to FIDO-enabled devices and thus may not be as accommodating to the entire segment of users for a local Thai bank.

5. Open problems

While the approach used by the local bank to addressing phishing attacks is inadequate and exacerbates the problem, we point out a subtle point that the problem being addressed here is one that in fact deserves further investigation. First, server authentication is *not* a solved problem. Although TLS unilateral authentication of servers based on server certificates is widely deployed and is in prevalent use, usability problems persist, and millions of users still end up conversing with malicious servers as evidenced by statistics pertaining to phishing scams [15]. Users are inclined to click ok without reading warnings and error messages. Effective visual tricks such as picture-in-picture attacks have been successfully employed [11]. Second, for web applications that use unilateral authentication of servers, the entity making the decision whether to accept the claimed identities of the servers based on the validity of their certificates is not a human user but a client program (e.g. a web browser). Yet, the entity that needs to respond appropriately (e.g. by proceeding with the rest of the transaction or not) is the human user. The local bank's attempt to include the users into the decision-making process is a valiant one, and whether a mechanism that allows human users to help determine the authenticity of the servers would end up helping or hurting the overall security of the system is a question that requires further usability research to explore.

6. Acknowledgments

This work is supported by the Royal Society and the Thailand Research Fund. We would like to thank Phisan Kaewprapha, Nawin Somyat, and Matthew Dailey for their insightful comments.

7. References

[1] eBay Toolbar and Account Guard. http://pages.ebay.com/help/confidence/account-guard.html.

[2] Fido Alliance Specifications. https://fidoalliance.org/download/.

[3] Computer Crime Act, 2007. https://advox.globalvoices.org/wp-content/downloads/Act_on_Computer_Crime_2550 (2007).pdf.

[4)] Bank web scammers slip through net, 2016. http://www.bangkokpost.com/news/general/1100913/bank-web-scammers-slip-through-net.

[5] Google Authenticator Android app. Opensource version, 2018. https://github.com/google/google-authenticator-android.

[6] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 602-619. Springer, Aug. 2006.

[7] R. Dhamija and J. Tygar. The battle against phishing: Dynamic security skins. *In 2005 IEEE Symposium on Security and Privacy*, pages 77-88. IEEE Computer Society Press, May 2005.

[8] T. Dierks and C. Allen. *RFC 2246 - The TLS Protocol Version 1.0.* Internet Activities Board, Jan. 1999.

[9] S. Groat, D. Shelly, and M. Dunlop. GoldPhish: Using images for content-based phishing analysis. In J. L. Mauri and M. Popescu, editors, *2010 Fifth International Conference on Internet Monitoring and Protection,* pages 123-128, Barcelona, Spain, May 2010. IEEE Computer Society.

[10] A. Herzberg and A. Jbara. Security and identification indicators for browsers against spoofing and phishing attacks. *ACM Transactions on Internet Technology*, 8(4): 16: 1-16: 36, Oct. 2008.

[11] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In S. Dietrich and R. Dhamija, editors, *FC 2007,* volume 4886 of *LNCS,* pages 281-293. Springer, Feb. 2007.

[12] P. Kongkhieo and C. Suwannaphum. Phishing attacks and mitigation. Bachelor's thesis, Thammasat University, Pathum thani, Thailand, 2017.

[13] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. *RFC 6238: TOTP: Time-Based One-Time Password Algorithm,* May 2011.

[14] Y. Nakaguro, M. N. Dailey, S. Marukatat, and S. S. Makhanov. Defeating line-noise CAPTCHAs with multiple quadratic snakes. *Computers & Security,* 37: 91-110, 2013.

[15] Z. Ramzan. *Phishing Attacks and Countermeasures,* chapter 23. Springer, 2010.

[16] F. Stark, C. Hazirbas, R. Triebel, and D. Cremers. CAPTCHA recognition with active deep learning. In B. Hammer, T. Martinetz, and T. Villmann, editors, *Workshop New Challenges in Neural Computation,* Mittweida, Germany, Mar. 2015.

[17] D. Stuttard and M. Pinto. *The Web Application Hacker's Handbook.* John Wiley & Sons, Inc., Indiana, United State, second edition, 2011.

[18] I. T. Union. ITU botnet mitigation toolkit, 2008. http://www.itu.int/ITU-D/cyb/cybersecurity/projects/botnet.html.