Scientific Research Publishing

Journal of Software Engineering and Applications, 2024, 17, 787-802
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Development of Two-Factor Authentication to Mitigate Phishing Attack

Varun Dixit[1*], Davinderjit Kaur[2]

[1]Omnissa LLC, Palo Alto, CA, USA
[2]MedROAD Lab, University of Alberta, Edmonton, Canada
Email: *varundixit@vdixit.com

How to cite this paper: Dixit, V. and Kaur, D. (2024) Development of Two-Factor Authentication to Mitigate Phishing Attack. *Journal of Software Engineering and Applications*, **17**, 787-802.
https://doi.org/10.4236/jsea.2024.1711043

Received: September 22, 2024
Accepted: November 3, 2024
Published: November 6, 2024

Abstract

The two-factor authentication mechanism is gaining popularity as more people are becoming aware of the need to secure their identities. In the current form, existing 2FA systems are defenseless against phishing attacks. They do not provide any visual indicator to the user to check the website's validity before logging in during phishing attacks. This exposes the user's password during the phishing attack. Two-factor authentication needs to be enhanced to provide a mechanism to detect phishing attacks without adding a significant burden on the user. This research paper will propose a novel 2-FA TOTP mechanism to provide a subconscious indicator during a phishing attack. In comparison, the new proposed novel approach provides better security against phishing attack. Lastly, the mathematical analysis is performed to understand the TOTP variance and validate the security considerations against the existing 2FA systems with respect to adversary attack.

Keywords

Two Factor Authentication, 2FA, Phishing Attack, Fixed 2FA, TOTP, HMAC

1. Introduction

A phishing attack is a type of social engineering cyber-attack where an attacker tries to imitate an institution or individual, often to steal user data. The most common form of phishing attack is deceptive phishing or cloned phishing [1]. In this form, a link is sent to the victim's email, pointing to a malicious website. On this website, the hacker waits for the victim to access and enter certain information, which the hacker can exploit. Federal Bureau of Investigation (FBI), in its July 2019 announcement, reported a total loss of more than 12 Billion dollars worldwide due to Business E-mail Compromise (BEC)/Email Account Compromise

(EAC) [2]. Further, BEC can result in Reputational Damage, Intellectual Property Loss, and Direct Costs for the company [3]. A survey conducted found that 83% of Global Infosecurity respondents experienced phishing attacks in 2018, which demonstrates a 9 percent year over year increase [4]. Google experiences 100 million phishing messages per day [5]. **Figure 1** and **Figure 2** represent the prevalence of phishing attacks.

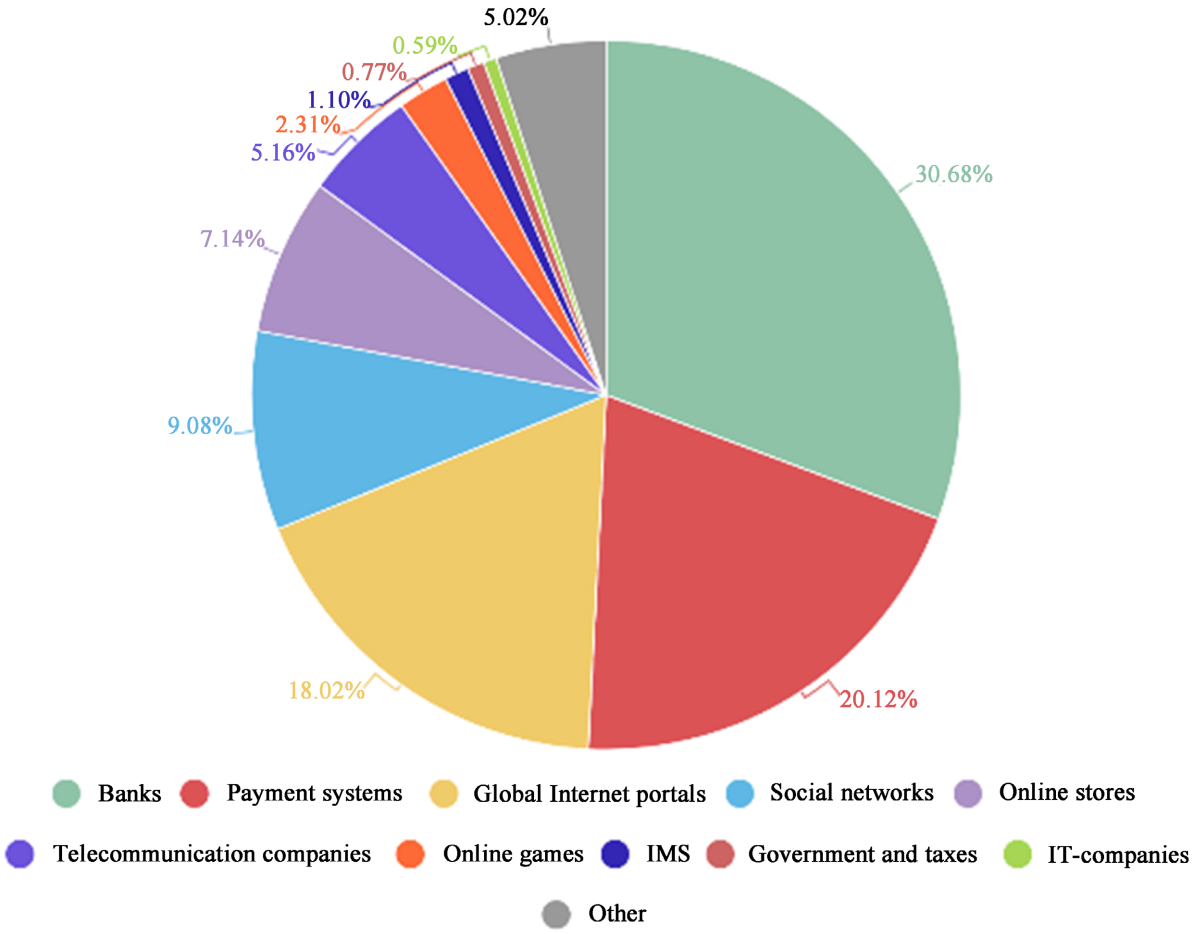

Figure 1. Distribution of organizations subjected to phishing attacks by category, Q2 2019 [7].

The industry has pivoted to two-factor authentication to add an additional layer of security to users' accounts (2FA). In the 2FA system, a user must provide additional information other than just a username and password. A survey found that the awareness of 2FA has increased from 44% in 2017 to 77% in 2019 [6], and its adoption is on the rise.

While two-factor authentication (2FA) adds an additional layer of security by requiring users to present two forms of identification, it is not without vulnerabilities, especially in the context of phishing attacks. A critical limitation of 2FA is that it does not inherently protect users from phishing attempts where attackers trick users into willingly providing both their credentials and 2FA tokens. For instance, in a phishing scenario, attackers can create fake login pages that request

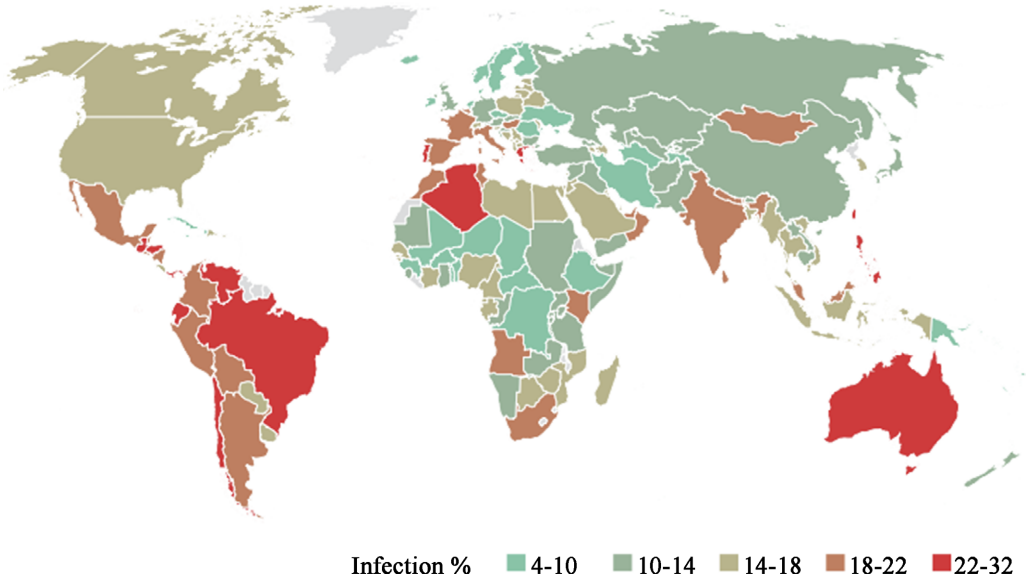

Figure 2. Geography of phishing attacks [7].

not only usernames and passwords but also the one-time passcodes or tokens generated by 2FA systems. Once the attacker captures this information, they can immediately use it to authenticate themselves on the legitimate platform, as 2FA tokens are typically short-lived but valid for enough time for the attack to succeed. Even in the cases where the 2FA token cannot be used, the username and password are already exposed to hackers. As nearly 83% of users use the same password for multiple websites, the victim's other accounts can be compromised [8].

2. Existing 2FA Mechanisms to Avoid Phishing Attack

1) Token based authentication;

2) User Pre-Selected Image.

Token Based Authentication

Token based two-factor authentication requires users to present information from two types from the following three:

a) Something they know.

b) Something they are.

c) Something they have.

Something they know traditionally refers to information such as username/email and password. This combination is generally the first form of authentication mechanism.

Something they are referred to fingerprints or other biometric information that can be used to verify an individual's identity. *Something they have* generally refers to a hardware or software-based tokens which can be used as a second layer of authentication [5].

The most common form of second-factor authentication in use is hardware or software based time-based one-time code generators. These codes are generally generated by the user's device (Mobile Phone, RSA Token) on the fly or sent to

user through a separate channel such as phone call or SMS [9].

Limitations of TOTP authentication:

While 2FA systems provide a strong defense against many hacking methods, it does not prevent exposing the passwords. This exposure can result in a chained attack against a user. For example, a password extracted from gmail.com can be used against facebook.com if the passwords are the same and so on. Figure 3 represents the user's action and information stored by the hacker. It is evident that during phishing attacks that require the user to login, username, password, and 2FA token can get compromised.

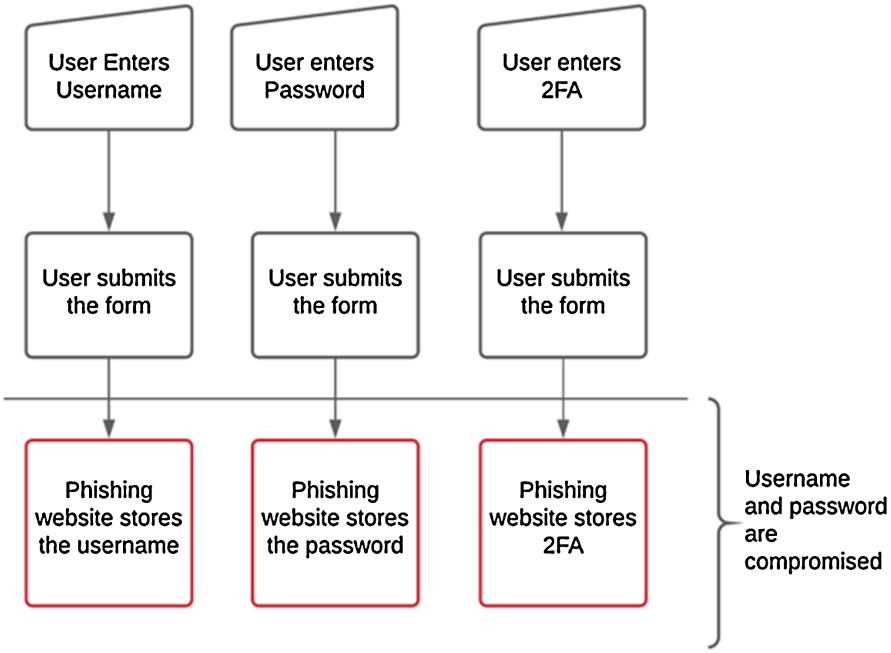

Figure 3. Flow diagram of user action and impact during phishing attack.

Following is the sample flow of a phishing attack where 2FA authentication exposes the password.

Steps:

1) Redirect the user to a phishing website.

2) Make the user enter a username and password.

3) Make the user enter 2FA authentication.

4) Redirect the user to a real website with an error message stating the login failed.

For example:

1) An attacker created a phishing website for google.com under the domain g00gle.com.

2) Attackers send an email to the victim, say John Doe, with a link to https://www.g00gle.com. When a user clicks the link, it gets redirected to before mentioned phishing website.

3) John will enter the username.

4) John enters the Google password.

5) John enters the 2FA verification code.

6) Attackers extract the password entered by John in Step 4 and use it to log in to other third-party accounts.

In the above scenario, even though www.google.com is able to prevent an attacker from its services, it is unable to protect its user's password.

The following visual diagram flow (**Figure 4**) represents the forms displayed to a victim while accessing www.g00gle.com.

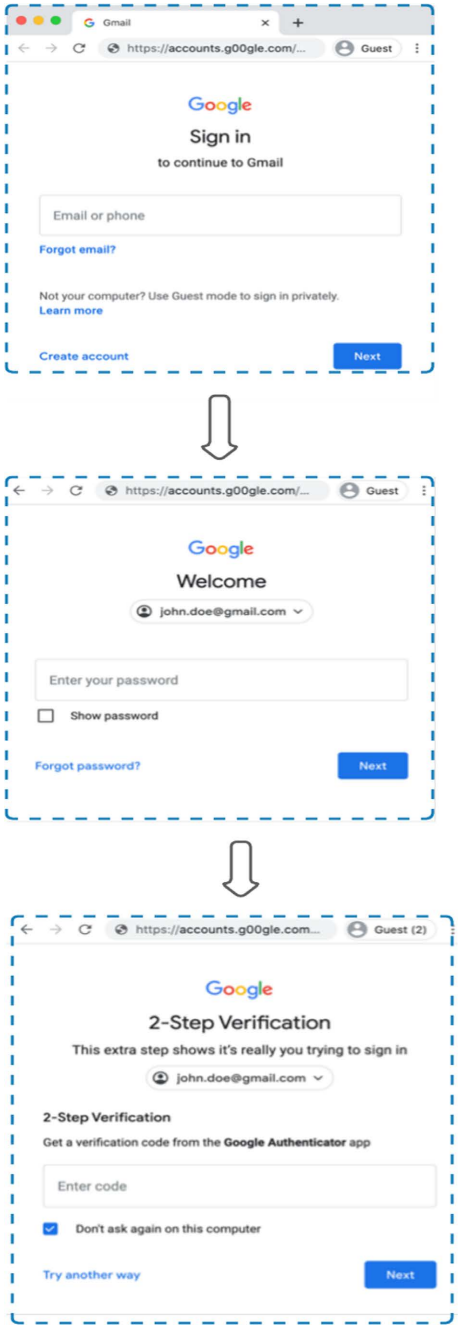

Figure 4. Flow of forms displayed to a victim while accessing www.g00gle.com.

User Pre-Selected Image

In this approach, the user as part of first-time signup, or when enabling the MFA selects a certain text or image that is expected to be displayed the next time the user logs in. After entering the username on the website, a trusted image/text is shown on the screen. If the image/text is not what the user expects, user gets sub-consciously alerted to be more vigilant and hence can try to verify the authenticity of the algorithm. Figure 5 represents a sample login page using this technique.

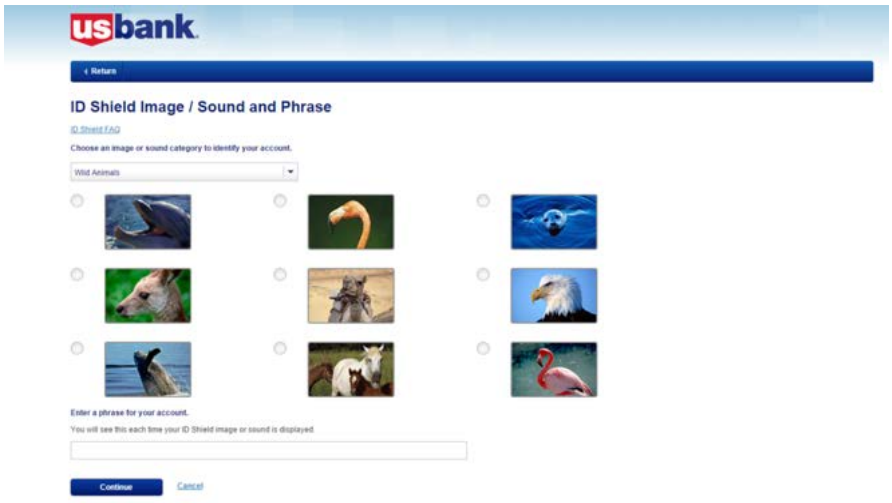

Figure 5. US Bank Pre-Selected login form [10].

Limitations of pre-selected image:

1) There is no standardized set to show pictures/text. Each website follows its standards, making it tedious for the user to remember what their trust identifier for each website was. This makes users ignore those trust identifiers.

2) For personalized phishing attacks, this technique fails as an attacker can place the trust identifier itself on the phishing website. An attacker can easily achieve this by entering the victim's username on the original website, copying the trust identifier, and replicating it on the phishing website. Furthermore, since these identifiers are static, an attacker does not need to perform this copy in real-time.

3. Proposed Solution

The need of the hour is finding a solution that works in the following constraints.

1) Sub-consciously provides feedback to the user to become vigilant in case of a phishing attack.

2) Does not lower the security standards of existing 2FA authentication.

3) Provides a user mechanism to enter 2FA token even on non-homogenous/slow networks without adding burden on the user.

An effective way to stop phishing attacks without adding any significant burden on the end-user is to provide a subset of characters from the 2FA code to the user where the end-user needs to login. The user is forced only to enter the missing

characters. If the trust characters do not match the user's token, the user is subconsciously made to think that something is wrong, which prompts the user to investigate if it is a phishing attack.

This is achieved by displaying a subset of trust characters on the form where the user needs to enter a password and 2FA token. Figure 6 represents the complete flow for the proposed solution.

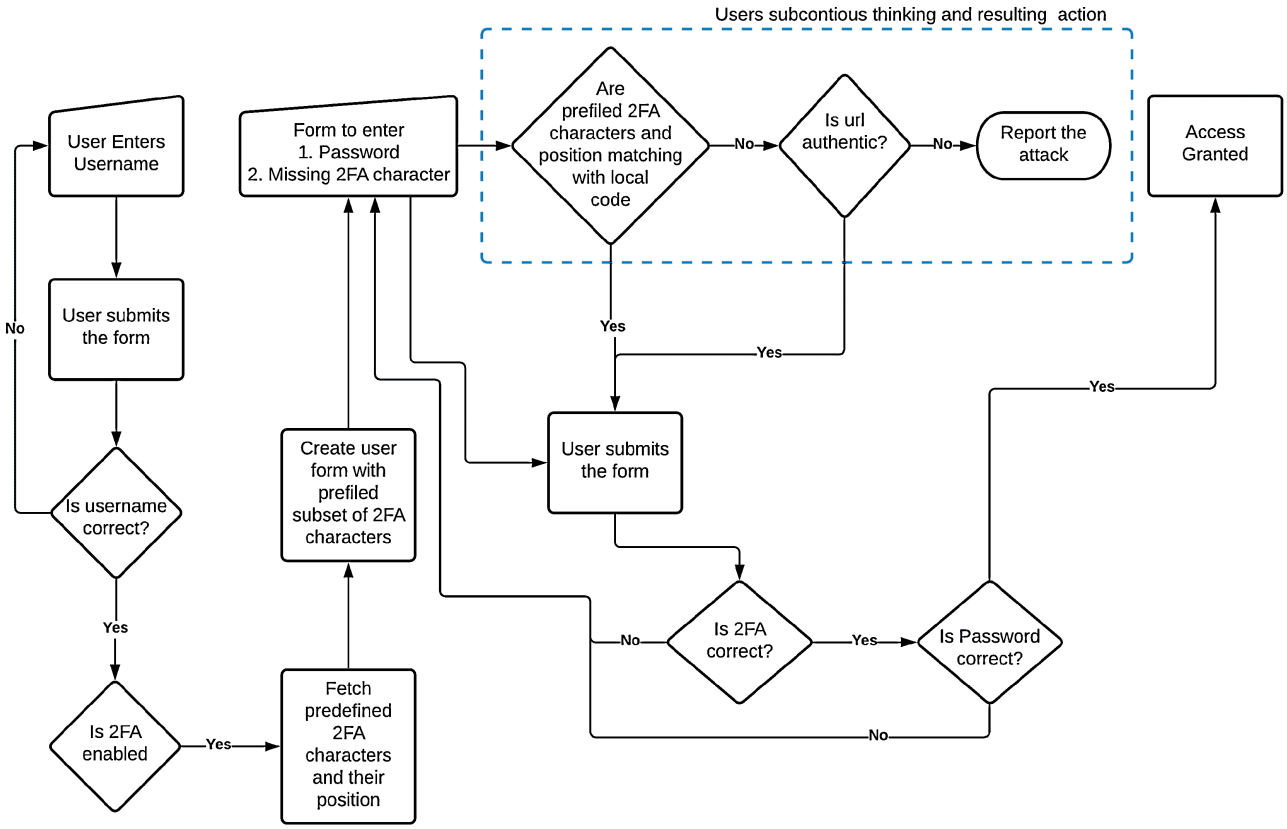

Figure 6. Flow diagram of 2FA authentication with trust characters.

User clicks the phishing link and opens the phishing website in the browser.

1) The server acknowledges the user's request and responds with a login page to enter the username.

2) The user enters the username and submits the form.

3) The server validates the username. A subsequent request is made to 2FA Token service to provide trust characters that will be displayed to the user on successful validation. Once the trust characters are received, the server sends a new form to the user to enter the password and 2FA token, while displaying the trust characters. Figure 7 represents a sample login screen flow where user first enters the password on the first screen and TOTP with trust characters on the second screen.

4) The user starts entering the 2FA code and subconsciously compares the trust characters for correctness alongside entering the missing numbers. The correctness is evaluated by comparing the trust characters displayed on the page with the complete token provided to the user. Figure 8 represents a sample screenshot of

the TOTP Authenticator client app on users mobile. This is the source of the 2FA code which user will enter.

5) The user enters the password.

6) The server verifies that the username and password combination is correct. Later it calls the 2FA service to verify the token entered by the user.

7) Once the verification of the 2FA token is positive, the user gets authenticated.

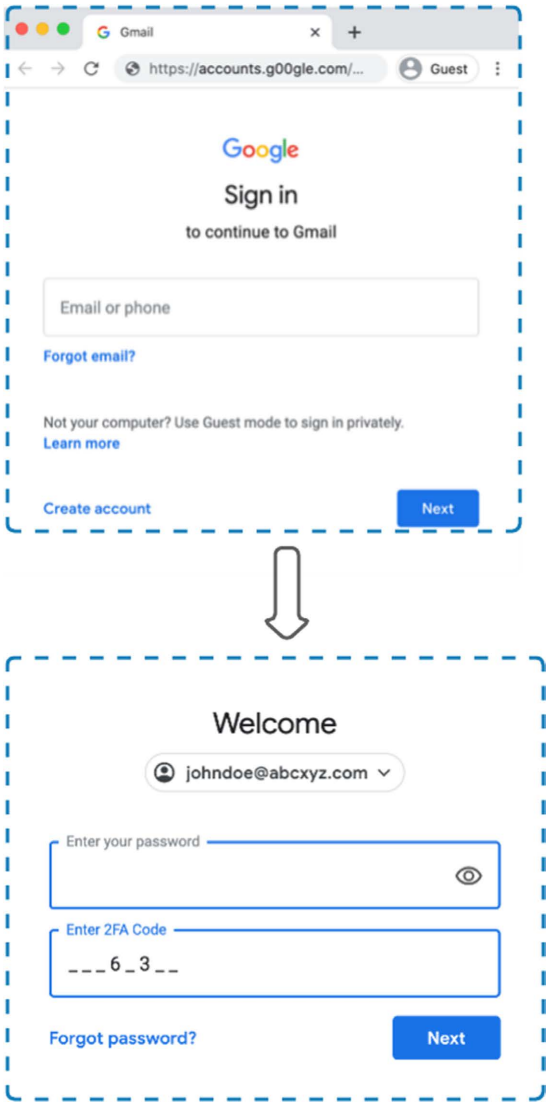

Figure 7. Flow of forms displayed to user while accessing with proposed solution.

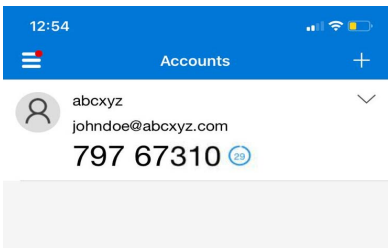

Figure 8. Screenshot of 2FA token app on mobile device of the user.

4. 2FA Setup for Trust Character Based TOTP

The proposed solution leverages the existing TOTP 2FA authentication setup to provide enhanced security. Figure 9 represents the values required to push to 2FA code to set up. The variables defined in black are pre existing variables in the current 2FA system. The variables defined in red are the new variables introduced as part of the new trust character-based TOTP 2FA proposed solution. Figure 10 represents the updated QR code that a user can scan on the mobile authenticator app to get the TOTP token.

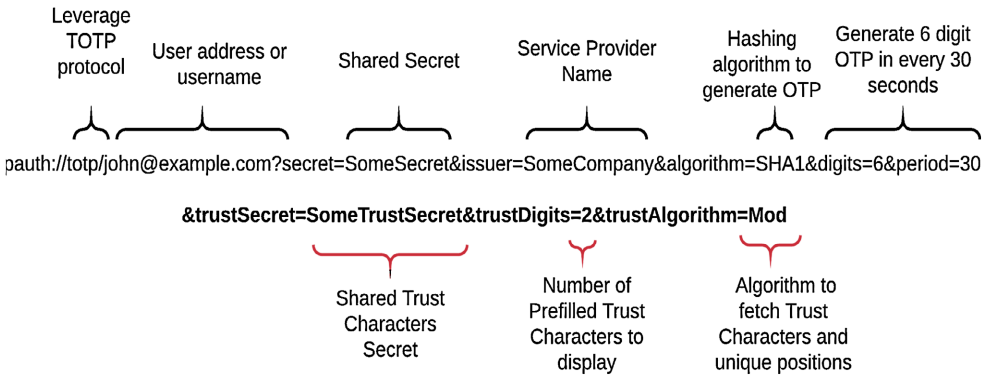

Figure 9. Structure of trust character based QR Code URL.

Figure 10. Sample QR Code with proposed solution.

5. Algorithm to Setup for Trust Character Based TOTP 2FA

The TOTP uses the HOTP algorithm by substituting the counter with a non-decreasing value of unix timer. To generate the position of trust character, and index at which it needs to be inserted, the same TOTP output can be leveraged. As TOTP value always returns the same character set for a given unix time and secret, leveraging its output to derive the position and index will also return a fixed subset for a given unix time and secret.

Presume following function:

```
return generateTOTPToken (SomeSecret, currentTime, timeStepSeconds, TOTPDigitCount);
```

Now with the proposed approach, the current algorithm will change to:

```
getTOTPCode(){
  oTPLength = TOTPDigitCount + trustCharacter-
Count;
  token = generateTOTPToken (SomeSecret, cur-
rentTime, timeStepSeconds, TOTPDigitCount);
  trustCharacters= generateTOTPToken
(SomeTrustSecret, currentTime, timeStepSeconds,
trustCharacterCount);
  trustCharacterPosition = getUniquePosition
(trustCharacterSecret,timeStepSeconds,oTPLength)
  mergedTOTPCode = getMergedCode (token,
trustCharacters, trustCharacterPositions, oT-
PLength);
  return mergedTOTPCode;
}

getUniquePositions (trustCharacterSecret, cur-
rentTime, timeStepSeconds, oTPLength) {
  timeStepValue = currentTime / timeStepSeconds;
  hashed_value = hash(trustCharacterSecret +
timeStepValue);
  Set uniqueCharacterSet;
  int index=0;
  while (uniqueCharacterSet.size < trustCharacter-
Count && index<hashed_value.length){
        int position = uniqueCharacterSet.cha-
rAt(i) % oTPLength
        if(! uniqueCharacterSet.contains(position)){
              uniqueCharacterSet.add(position)
        }
  }
  if(uniqueCharacterSet.size < trustCharacter-
Count){
        // add positions index in serial order starting
zero
  }
            return uniqueCharacterSet;
}


getMergedCode(token, trustCharacters, trustCharac-
terPositions, oTPLength) {
  trustCharacterIndex=0,
            tokenIndex=0;
  traverse (index 0 .. to .. oTPLength) {
        if(trustCharacterPositions.contains(index)){
              mergedCode [index] = trustCharac-
ter[trustCharacterIndex];
              trustCharacterIndex++;
                    }else{
              mergedCode [index] = token [token-
Index];
              tokenIndex++;
                    }
        }
        return mergedCode;
}
```

For the TOTP edge case defined in RFC6238, following are the values after appending trust characters. Table 1 displays the results taken with variables as:

Some Secret = ASCII String value "12345678901234567890";

Trust Secret = ASCII String value "09876543210987654321";

Time Step = 30;

Trust Digits = 2;

Trust Algorithm = Mod;

T_0 *= 0 Unix epoch as the initial value to count steps.*

Table 1. TOTP values.

Time in Seconds	Original TOTP	Trust Character Positions	Trust Characters	Final Code	Algorithm
59	481090	1, 5	0, 7	40810790	SHA-1
59	771515	3, 6	2, 2	77125125	SHA-256
59	949331	1, 3	1, 8	91489331	SHA-512
1111111109	135816	1, 7	6, 6	16358166	SHA-1
1111111109	202986	0, 2	4, 0	42002986	SHA-256
1111111109	827329	3, 5	6, 8	82763829	SHA-512
1111111111	127384	2, 6	5, 5	12573854	SHA-1
1111111111	348986	1, 5	9, 8	39489886	SHA-256
1111111111	807694	0, 4	6, 4	68074694	SHA-512
1234567890	229225	1, 3	3, 5	23259225	SHA-1
1234567890	643556	1, 2	7, 3	67343556	SHA-256
1234567890	447659	1, 6	9, 5	49476559	SHA-512
2000000000	108487	0, 4	2, 3	21083487	SHA-1
2000000000	161267	3, 7	4, 7	16142677	SHA-256
2000000000	290684	3, 4	2, 7	29027684	SHA-512
20000000000	84444	2, 3	4, 3	08434444	SHA-1
20000000000	891795	4, 6	6, 7	89176975	SHA-256
20000000000	120478	1, 3	9, 5	19250478	SHA-512

6. Security Considerations for Trust Character Based TOTP 2FA

As per rfc4226, the security of the underlying HMAC-SHA-1-based HOTP can be defined as

$$Sec = sv/10^{Digit} \quad (1)$$

where:

- *Sec* is the probability of success of the adversary;

- s is the look-ahead synchronization window size;
- v is the number of verification attempts;
- *Digit* is the number of digits in HOTP values

In case where no trust characters are shown to user, the number of permutations for the right combination are

$$X = \frac{n!}{(n-r)!} \tag{2}$$

where

X = Total number of permutations available without trust characters;

r = Total length of the code;

n = Distinct characters available to choose from.

As per the proposed solution, since a subset of digits will be displayed to the user, the above new equation formed is as below:

$$X' = \frac{n!}{(n-(r'-F))!} \tag{3}$$

where

X' = Total number of permutations available with a fixed subset of characters;

F = Number of trust characters;

r' = New total length of the code;

n = Distinct characters available to choose from.

To keep the Sec same, the permutations should remain the same.

$$X = X' \tag{6}$$

$$\frac{n!}{(n-r)!} = \frac{n!}{(n-(r'-F))!} \tag{5}$$

$$r' - F = r \tag{7}$$

$$r' = r + F \tag{8}$$

The code length needs to be increased to maintain the status quo of the existing 2FA TOTP security level with the newly proposed solution with respect to the original implementation.

For example, for a 6-digit numeric 2FA token, the permutations are decreased from 151,200 to 5040 if two trust characters are shown. If the code length is increased to 8 with 2 trust characters, it regains the same permutations, *i.e.* 151,200, and the end-user only has to enter only 6-digits like in the previous case.

Probability that phishing website will be able to provide the trust characters at the right location is

$$Y = \frac{1}{n^F} \tag{9}$$

where

Y = Probability of right placement of characters;

F = Number of trust characters;

n = Superset of characters that make 2FA code.

So, in case of numbers between 0 - 9,

$$Y = \frac{1}{10^F} \tag{10}$$

To summarize, as proved in Equation (10), the overall length of TOTP needs to increase by the number of displayed trust characters in order to keep the TOTP adversary attack probability the same.

TOTP Variance

The following proof establishes the lemma estimate biases as described in RFC4226.

Let the random variable X be uniformly distributed over $Z\_\{N\}$.

Table 2. TOTP variance.

Values	Probability that each appears as output
0, 1, ..., 483647	$2148/2^{31}$ roughly equals to $1.00024045/10^6$
483648, ..., 999999	$2147/2^{31}$ roughly equals to $0.99977478/10^6$

Even though, it is slightly skewed towards lower bound numbers, but at the scale of 10^6 it is negligible.

The probability of HOTP before the introduction of trust characters as established in RFC4226 is

$$P\_\{N,m\}(z) = Pr\left[x \bmod m = z : x \text{ randomly pick in } Z\_\{n\}\right] \tag{11}$$

Then for any z in $Z\_\{m\}$

$$P\_\{N,m\}(z) = \begin{cases} (q+1)/N & \text{if } 0 \le z < r \\ q/N & \text{if } r \le z < m \end{cases} \tag{12}$$

Since the trust characters also follow the very same algorithm, hence the TOTP variance and Probability of trust characters in exactly as defined in Table 2 and Equation (12) respectively.

7. Benefits and Comparison of This Trust Character Based TOTP 2FA

The existing 2FA TOTP token mechanism is extended in the proposed solution, hence reducing users' learning curve.

The trust characters displayed on the screen allow the user to detect if the current user is under a phishing attack on a valid indented website.

In the scenario that due to manual/network delay, the characters displayed on the screen are not the same as that on the authenticator app, the overwrite functionality still provides the ability to user to enter the complete 2FA code and login. Hence, the proposed solution does not hamper the user experience or delay logins.

As the solution leverages a separate secret token for displaying the trust characters, this keeps intact the security aspects of the current 2FA token-based system.

Resynchronization

Because the proposed approach leverages the TOTP output to derive the trust characters' digit and index, HOTP resynchronization as defined in RFC4226 section 7.3 does not need to happen.

As defined in TOTP RFC6238, resynchronization is still recommended as the client and server clock can go out of sync, or due to network delays, the user input gets delayed. Since the solution accepts the overwritten numbers for trust characters, the resynchronization technique in TOTP RFC6238 is still valid. Hence a limit can be set both forward and backward from the calculated time step on receipt of the OTP value.

Further, since users can override the trust characters; this helps the system overcome network delays and out-of-sync clocks. As users will only opt for overriding the trust characters after verifying the website's validity, this does not reduce the proposed solution's security benefits.

Comparison with Existing TOTP 2FA

Existing TOTP 2FA does not provide any visual indicators to user in case of phishing attack. The proposed trust character based TOTP 2FA does provide the user a sub-conscious warning to validate the authenticity of the website in case of phishing attack. By increasing the length of OTP equivalent to the number of trust characters to display, both solutions provide the same level of security in case of brute force adversary attack as proved in eq 8. Hence with trust character based TOTP, the service can protect user from leaking password in case of phishing attack.

Comparison with User Pre-Selected Image

Existing Pre-selected image mechanism is a non-standardized way to provide visual indicators. Further, the onus of remembering the valid images for each website resides on the user. The new proposed trust character based TOTP eradicates the responsibility of user to remember the images per website. Instead, the proposed solution provides a run time mechanism to have visual indicators. The user can leverage the hardware/software/SMS based token numbers to validate the authenticity of the website.

8. Limitations of Proposed Solution

The proposed solution won't be able to guard against a single user under a targeted relayed phishing attack. In case of target-relayed phishing attack, the phishing website can in real time relay the username entered by the user to the real website, get the trust tokens and present the same to the user on the phishing website.

Though this limitation can be overcomed by leveraging other security systems, like IP stamping, and keeping a record of existing login systems etc.

In addition to this, there are some additional challenges related to practical application of the proposed solution. With the new proposed protocol, ensuring compatibility with existing TOTP authenticators could be challenging. Additionally, Attackers continuously evolve their methods to circumvent security mechanisms. As

phishing attacks become more sophisticated, relying solely on TOTP based mechanisms even with the proposed enhanced protocol for additional security may not be sufficient. Once the attacker has access to the TOTP secret and trust character secret, the proposed will deem useless against the attack.

9. Conclusion

The existing 2FA systems do not prevent the user from exposing the passwords. The new novel solution proposed is to display trust characters on the same page where the user needs to enter a 2FA token and password. The existing TOTP mechanism is extended to provide random trust characters and their respective locations, which remain consistent between the server and client. The trust character-based TOTP 2FA subconsciously prompts users to check the website's validity in case of a mismatch of trust characters, which happens during the phishing attack. Since the system provides the user overwrite ability, the look back and forward and the resynchronization process of existing TOTP remains intact. This paper further discusses the security considerations of the novel approach. It has been proved that by increasing the number of OTP length equivalent to the number of characters, the probability of adversary attack remains the same as that of existing TOTP. On the contrary, due to the presence of trust characters, the service is now more secure against phishing attacks. The TOTP variance and TOTP with trust characters are published for the edge cases of epoch time. Further, when compared with the existing pre-selected image 2FA, it is established that the new system is much more user-friendly and reduces the user overhead of remembering images. In terms of limitation, the new approach would not safeguard users' credentials in case of a targeted relay attack, but this limitation can be overcome by leveraging the new solution in conjunction with other techniques like IP stamping, and AI-based login historical score.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Randed (2018) 8 Most Common Types of Phishing and How to Protect against Them All. https://web.archive.org/web/20201001042454/https://randed.com/types-of-phishing/?lang=en

[2] Vade Secure (2020) The Corporate Impact of Phishing. https://www.vadesecure.com/en/blog/the-corporate-impact-of-phishing

[3] Vade (2023) What Is Email Security? https://www.vadesecure.com/en/email-security

[4] Help Net Security (2019) 83% of Global Respondents Experienced Phishing Attacks in 2018. https://www.helpnetsecurity.com/2019/01/25/experienced-phishing-attacks/

[5] Reese, K., Smith, T., Dutson, J., Armknecht, J., Cameron, J. and Seamons, K. (2019) A Usability Study of Five Two-Factor Authentication Methods. *Proceedings of the*

Fifteenth USENIX Conference on Usable Privacy and Security, Santa Clara, 12-13 August 2019, 357-370.

[6] Frazier, S. (2020) The 2019 State of the Auth Report: Has 2FA Hit Mainstream Yet? https://duo.com/blog/the-2019-state-of-the-auth-report-has-2fa-hit-mainstream-yet

[7] Vergelis, M., Shcherbakova, T. and Sidorina, T. (2019) Spam and Phishing in Q2 2019. https://securelist.com/spam-and-phishing-in-q2-2019/92379/

[8] Zane (2018) Password Security Report: 83% of Users Surveyed Use the Same Password for Multiple Sites. https://www.cyclonis.com/report-83-percent-users-surveyed-use-same-password-multiple-sites/

[9] Kan, M. (2019) Google Detects Steady Stream of Phishing Attacks from Cyberspies. https://www.pcmag.com/news/google-detects-steady-stream-of-phishing-attacks-from-cyberspies

[10] Anand, P. (2015) Those Security Images on Your Bank Log-in Pages? They're Useless. https://www.marketwatch.com/story/banks-find-online-security-images-offer-little-protection-2015-11-05