Final Report: 2D-2FA Software Implementation

Zane Globus-O'Harra, Doug Ure

9 June 2023

Abstract

TODO TODO TODO TODO TODO TODO TODO TODO TODO

Contents

1 Introduction

Two-factor authentication (2FA) is the contemporary approach to authorizing a user, where the first authentication factor is the user's password, with the second factor being an additional piece of information that only the user could know. This additional piece of information is generally categorized to three types:

1. something the users knows, e.g., a password;

2. something the user has, e.g., a physical key;

3. something the user is, e.g., a fingerprint [1].

Most commonly, the second authentication factor is a PIN that the user enters on a separate device, or a push notification sent to the user over a secure channel to a trusted device that the user approves [1]. However, both PIN-2FA and push-based 2FA have some issues, such as shoulder surfing [2, 3] SIM swap attacks [4], and neglectful user approvals [5], among others.

A recent (2021) paper [6] by Shirvanian and Agrawal presents a new approach to 2FA, which they have coined "2D-2FA." In this new approach, the two factors are the user's login information (their secret password), and a randomly generated identifier, which is generated by the server and displayed to the user on their client, which they must enter into their personal device for their session to be approved. The two "dimensions" come from the fact that the identifier is used to automatically generate a PIN on the device, and both the PIN and the identifier are sent to the server, which verifies the generated PIN. Hence the name "2D-2FA," where the two factors are the password and the identifier, and the two dimensions are the identifier and the PIN.

In our project, implemented a bare-bones registration phase, and a fleshed out authentication phase of the 2D-2FA system, following the implementation laid out in [6]. In doing this, our hopes were to learn more about multifactor authentication and network security, as well as verifying that the 2D-2FA system functions correctly and meets the design goals as described by Shirvanian and Agrawal.

1.1 Keywords

To avoid further confusion, we will address several important keywords that are important to the design and implementation of the 2D-2FA system.

HMAC Keyed-hash message authentication code, or HMAC, is an algorithm used for message authentication, creating a message authentication code (MAC) using a hash function. This is used to authenticate the source of a message as well as its integrity. The HMAC has two parameters, a message input and a secret key known only to the message sender and the intended receiver [7].

Identifier The identifier is a random value that is generated by the server and presented to the user via the client interface.

Time Slice The time slice is 30-second chunk of time—as defined in section 6.2 of [6]—since the epoch. This is calculated by dividing the time in seconds since the epoch by 30 and taking the integer part.

PIN The PIN is a HMAC, and is generated using the time slice and the identifier as the message, and the secret key as the hashing key.

Client The client is the device that the user uses to log in to the server, most likely a web browser through their personal computer or laptop.

Server The server handles connections from the user via the client, produces identifiers, and verifies PINs, thus authenticating users and allowing them to log in.

Device The device generates PINs using the identifer and the time slice as input. It then sends these PINs to the server for verification.

1.2 Motivations

We are motivated to work on this project for several key reasons. Firstly, the subject of this project has real-world relevance. 2FA is becoming increasingly more common to authenticate users, and as it becomes more prevalent, attackers will focus more of their efforts on finding ways to break through its layers of security. Our project will help us learn about ways to further increase the security of 2FA by using additional information along with the user's credentials and the server-provided "identifier."

This is also a learning opportunity for us. Neither of us are very familiar with security, and it is something that we are very interested in learning about. By completing this project, we will develop valuable technical skills and increase our knowledge base, as well as preparing us for future projects and industry roles.

Lastly, this project could have a real impact on end-users. 2FA enhances users' trust and confidence in online systems by making their personal information and online interactions more secure. This project has the potential to contribute to a larger goal of make a more secure digital environment.

1.3 Objectives

Our objectives for this project are to create a working implementation of the 2D-2FA system in software, focusing mainly on the authentication phase, as described in section 3.2 of [6]. As previously mentioned in our progress report, we have written software to implement the functionality of the server and the device in this authentication scheme. We also added a simple web interface for the client and the device, so that the user can use their browser to use our implementation.

In terms of specific deliverables, these were outlined in our midterm report, but are repeated here for posterity.

- A working implementation of 2D-2FA.
    - Programs for the server and the device, as described in the 2D-2FA paper.
    - This implementation will work across multiple devices.
    - This implementation will work for multiple users.
- Test cases for our code.
- Documentation.
    - Installation instructions.
    - Usage instructions.
    - A design diagram.
    - Well-commented code.

2 Related Work

In this section, we will look at some traditional 2FA implementations, chiefly hardware token-based authentication and single sign-on software token-based authentication.

2.1 Hardware Token Authentication

2.2 Single Sign-On Software Token Authentication

3 2D-2FA

This section will describe the design goals of the 2D-2FA system, as well as going more in-depth into the processes that the system uses to securely authenticate users. We will also cover our implementation of the 2D-2FA system, and describe any differences and changes between our implementation and the implementation outlined in section 6 of [6].

3.1 Design

2D-2FA was primarily designed to meet three usability and deployability goals as described by section 3.3.2 in [6]. These include ease of use, ease of integration, and universal compatibility. These goals are met by using a simple identifier that is easy to enter into the device, not using any third party services, and using standard, well-established technologies.

The 2D-2FA system consists of two phases, the registration phase and the authentication phase. In our project, we assumed that the registration phase had already been completed between the involved parties, but we will go over that phase here because it would be required for the implementation of a full 2D-2FA system, rather than a toy example or proof of concept implementation.

Registration Phase During the registration phase, the parties involved in the 2D-2FA protocol need to establish communication channels between them, and share secret keys between the server and the device.

First, a user would register their username and password with the server. The server would pick a secret key for that user (generating it from their password using HMAC). The secret then needs to be transferred to the user's device (e.g., cell phone), which is typically done by manually entering it into the device, or by scanning a QR code generated by the server. This is so that the device will be able to generate PINs later on in the 2D-2FA process.

The server stores a hash of the user's password and secret key, and the device stores only the secret key related to the user's account stored on the server. The client (e.g., a web browser on a laptop that the user uses to log in to the server) does not store any information during the registration phase.

Authentication Phase The authentication phase involves the interaction of the parties to authenticate the user to the server securely.

First, the user attempts to log in to the server from the client. The user's login information (username and password) are used as the first authentication factor. Next, the server displays a unique identifier to the user through the client (the identifier can be implemented in a variety of ways, either via QR code, a fourgram pattern as described in section 6.1 of [6], and so on).

On the user's device, the user selects the server that they are logging in to, and enters the identifier that was displayed to them on the client. The device generates a PIN using the identifier, the current time, and the user's secret key. The PIN and the identifier are sent to the server.

When the server receives the PIN and the identifier, it authenticates the user's session associated with the identifier, generating several PINs to find one that matches with the time the PIN was generated on the user's device. During this phase, the server keeps a temporary record of all active sessions and identifiers.

3.2 Implementation

Because of limited time and resources, we went with a software-only implementation that can be accessed in the web browser. We used Python because it is a language that we are familiar with, and it also offers a diverse standard library, with built packages for networks and security.

The system that we wrote includes implementations of the functions that would run on the server and the device as described in [6]. Specifically, we will discuss our implementations for both the server and device, as well as go in-depth into the minor differences between our implementation and the implementation described in section 6 in [6].

Device The device contains a list of servers, usernames used on those servers, and keys for that server/username combination. In a full production version, this would be generated in the registration phase; for our proof-of-concept implementation, we simply included these in a text file. The keys are loaded by a `load_keylist` function that is called once, and can be easily swapped out for a different function, depending on how this information is stored in the final implementation.

The core of the device is the PIN generation. First, a time slice is created by taking the current epoch time (in seconds) and dividing by a fixed number, to get the current "time slice". This time slice is then combined with an identifier using a bitwise exclusive or operation. Finally, this combined value is hashed using HMAC, with the user's secret key as the key and SHA256, to produce a PIN, which is sent along with the username to the server.

Currently, the identifier is a six-digit number, as that is convenient for implementing the device on a PC or laptop. This could easily be changed out to a different identifier without any significant changes to the existing code; the only addition would be code for inputting the identifier.

Sending the PIN to the server is accomplished by a standard TCP message. For the purposes of this proof-of-concept implementation, the device uses a simple HTML interface for selecting the server/username and inputting an identifier.

Server The server contains a list of `user:key` pairs, stored and retrieved similarly to how the device handles its list of servers.

In a full deployment, the server-side code would be called by the login portion of a server, likely either by importing the code or through socket communication. For demonstration purposes, it currently uses a simple HTML interface, where a user can input a name and receive both an identifier and whether that name is authorized or not.

To handle identifiers, the server maintains a list mapping usernames to identifiers. This list is initially empty. When asked for an identifier, the server first checks this list, and if the username is there and the identifier is not expired, it returns the found identifier. If not, it generates a new identifier, saves the `user:identifier` pair to the identifier list, and returns the identifier.

Upon receiving a username and PIN from a device, the server uses the key and identifier it has saved for that user, and generates a series of comparison PINs, using the same method the device used to generate its own PIN. This series of comparison PINs is created using a range of time slices (currently ± 2 time slices), in order to account for differences in system clocks, transmission delays, etc. If any of these comparison PINs match the PIN sent by the device, then the user is added to the list of authorized users.

When the server is asked if a user is authenticated, it checks the authenticated user list, and if the user is present and not expired, then they are approved.

Both the identifier list and authenticated-user list include a configurable expiration time for each item. The server ticks periodically to check for expiration, and if the item is expired, it is simply removed from the relevant list. Additionally, a "minimum time" can be specified on the server. When an identifier is requested, the server checks how long it will be until the identifier will expire. If this length of time is less than the specified minimum time, then the old identifier is thrown out and a new identifier is generated, stored in the identifiers list, and passed on to the user. This is done to ensure the user has enough time to input the identifier before it expires.

Implementation Differences TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO

4 Security Analysis

TODO TODO TODO TODO TODO TODO TODO TODO TODO TODO

There are several possible avenues of attack to the 2D-2FA system that are proposed in [6]. In this section, we will go over each attack vector and analyze how the system protects against these attacks, as well as explaining how our implementation prevents these attacks.

Client Compromise In this attack, an attacker would have compromised the client that the user uses to log into the server, and thus gain access to the user's password. Assuming the device is not also compromised, the user's account is still secure.

This attack is prevented because once an attacker logs in using the user's password, they will receive an identifier generated by the server. But, without access to the user's secret key, the attacker has no means of computing a PIN that corresponds to the identifier, and thus no means of providing a valid PIN to the server. Additionally, without access to the user's device, the attacker has no way to generate a valid PIN without guessing the user's secret key, which would not be feasible.

In our implementation, TODO

Device Compromise In this attack, an attacker would have compromised the user's device, and as such they have full control over the device. In this attack vector, we assume that the attacker has access to the secret information stored on the device. This allows the attacker to generate any PIN, for any identifier.

This attack is prevented because for the attacker to gain access to the user's account, the attacker still needs to know the user's password to log in on the client. If the user has chosen a secure password that can not easily be guessed, the attacker must guess the user's password to gain access to their account, which would not be feasible.

In our implementation, TODO

Channel Compromise In this attack, an attacker has control over the channels connecting the parties (the client, device, and server) in the system. Specifically, there are three main channels in the 2D-2FA system. The channel between the client and the server is secure, the channel between the client and the device is through the user, and the channel between the device and the server is a regular channel. In this attack vector, the attacker could either listen to the channels (eavesdrop), or they could modify or block the network traffic.

This attack is prevented because, like with the client compromise, the attacker does not know the user's password.

TODO TODO TODO TODO

In our implementation, TODO

User Negligence In this attack, a user is simply negligent, and inadvertently grants access to an attacker.

This attack is prevented because, a user would only accidentally approve an attacker's session when the user enters the identifier that is displayed to the attacker, rather than the identifier that is displayed to the user. Depending on the type of identifier used, the possibility of this occurring is very low.

In our implementation, we used a 6-digit identifier, which means that if an attacker has logged in and is presented with an identifier, the user has a one-in-a-million chance (0.0001%) of approving that attacker's session.

Attacks on Third Parties In other MFA systems, third party entities are introduced, such as a MFA service provider. The system owner would need to trust these service providers, as well as study the security of the services that they provide to be comfortable with the security of their services. Attackers can target these third parties, as well as the channels connecting the third party services to the main system, which increases the complexity of the security analysis of the system.

Because 2D-2FA does not use any third party systems, this decreases the complexity of the security analysis, and thus reduces the surface of attack into the system. Additionally, 2D-2FA only uses well-established and well-studied technologies (HMAC, SSL/TLS, random number generation, etc.), further reducing attack vectors.

In our implementation, we use the well-established technologies mentioned in and used by [6]. The only variation is that we use TCP connections and encrypt the messages that we send over those connections instead of using TLS.

5 Conclusion

We created a proof of concept implementation of the 2D-2FA system, finding that it was easy to understand, build, and use. While we did not attempt to attack our system, we found the security analysis in [6] to be clear, thorough, and reasonable. Moreover, we did find that the design goals laid out in section 3.3.2 of [6] were accomplished during our implementation. It was relatively easy to implement our system, easy to use, and only made use of the Python standard library (except Flask, which was used for the simple web interface).

5.1 Complications

There were relatively few complications that arose during our project, all of which we were able to overcome. The first arose when we were trying to figure out the socket communications in the Python environment. However, after reading some documentation and getting a better understanding of the Python standard library socket API, we were able to successfully communicate between the device and the server.

Finally, because of the limitations of what information can be passed between the server and the device, we had to use two threads in the server to get everything working correctly. These threads did the following:

1. listen for connections from the device, verify the PIN that was sent from the device, add a user to the "authorized" list, and check for expired entries on the identifier list and authorized list;

2. run the Flask app, listen for users logging in, generate the identifier, display the identifier to the user, and check if a user is authorized (on the authorized list).

However, once we figured out that the server needed multiple threads to function as desired in our implementation, it was relatively light work to add the multithreaded functionality.

5.2 Lessons

The important lessons that we have learned include project planning, collaboration skills, and iterative development. While we had prior experience in all of these areas from previous projects, this project helped ingrain these principles into how we worked, altogether adding to a better workflow and increased productivity.

For the project planning, we had thoroughly read through the implementation section in [6]. From this, we broke down each element of the implementation into smaller chunks that were easier to tackle and implement. This allowed us to develop one module at a time, and ensure that module was functioning in the desired way before continuing to the next module.

In terms of collaboration, we have weekly meetings where we discuss what we have accomplished in the past week, and what we plan on completing for the next week. During the week, we update each other with our progress, as well as asking questions or seeing if we have suggestions for each other.

With regard to iterative development, this goes hand in hand with our project planning. Because we have broken down the problem into bite-sized chunks, we can iteratively implement these small portions, easily adding features and functionality to them as we progress, and iteratively changing them or modifying them when we encounter the need to do so.

5.3 Recommendations

While the study completed in [6] found that 75% of users preferred 2D-2FA over the more commonly know PIN-2FA, we believe that 2D-2FA could be better used in tandem with another 2FA method. There are multiple 2FA methods we found, most notably Sound-Proof [8] and Typing-Proof [9]. For the purposes of brevity, we will focus on how 2D-2FA can be used with Typing-Proof.

Typing-Proof, introduced by Liu, Li, and Deng, uses a registered phone to listen to the user's keystrokes when they type a random code presented to them during login. The time between the keystrokes is also recorded by the server, and is compared with the recording taken by the user's device. When the timing between the keystrokes recorded by the device and by the server match, the user is successfully authenticated.

The advantages that Typing-Proof brings to the table is decreased login time, as well as less interaction between the user and their personal device, which can be cumbersome and annoying, thus increasing usability [9]. In fact, the system usability scale (SUS) score of Typing-Proof is 81.7, whereas the SUS of 2D-2FA is 75. This is likely because 2D-2FA still requires the user to interact with their device by entering the identifier. Note, however, that no study comparing the two 2FA methods has been conducted.

We believe that Typing-Proof is a strong alternative to 2D-2FA, and it could be used in tandem with 2D-2FA. In such a scheme, a user might log in normally, with their device listening to their keystrokes. If they are in a loud or busy environment, such as a café, [9] found that users are less likely to user Typing-Proof. In this case, it would be convenient to use 2D-2FA as an alternative. Additionally, Typing-Proof has a measured false acceptance rate of about 0.015, and during cases where the device is unsure about the validity of the keystrokes it heard, it could use 2D-2FA instead.

References

[1] C. Jacomme and S. Kremer, “An extensive formal analysis of multi-factor authentication protocols,” *ACM Trans. Priv. Secur.*, vol. 24, Jan 2021.

[2] M. Eiband, M. Khamis, E. von Zezschwitz, H. Hussmann, and F. Alt, “Understanding shoulder surfing in the wild: Stories from users and observers,” in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI ’17, (New York, NY, USA), pp. 4254–4265, Association for Computing Machinery, 2017.

[3] L. Bošnjak and B. Brumen, “Shoulder surfing: From an experimental study to a comparative framework,” *International Journal of Human-Computer Studies*, vol. 130, pp. 1–20, Oct 2019.

[4] M. Saha, “NIST shouted, who listened? Analyzing user response to NIST’s guidance on SMS 2FA security,” Dec 2016.

[5] M. Jubur, P. Shrestha, N. Saxena, and J. Prakash, “Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications,” in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS ’21, (New York, NY, USA), pp. 447–461, Association for Computing Machinery, 2021.

[6] M. Shirvanian and S. Agrawal, “2d-2fa: A new dimension in two-factor authentication,” in *Proceedings of the 2021 Annual Computer Security Applications Conference*, ACSAC ’21, (New York, NY, USA), pp. 482–496, Association for Computing Machinery, 2021.

[7] National Institute of Standards and Technology, “The keyed-hash message authentication code (HMAC),” Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 198-1, National Institute of Standards and Technology, Gaithersburg, MD, 2008.

[8] N. Karapanos, C. Marforio, C. Soriente, and S. Čapkun, “Sound-proof: Usable two-factor authentication based on ambient sound,” in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC ’15, (USA), pp. 483–498, USENIX Association, 2015.

[9] X. Liu, Y. Li, and R. H. Deng, “Typing-proof: Usable, secure and low-cost two-factor authentication based on keystroke timings,” in *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC ’18, (New York, NY, USA), pp. 53–65, Association for Computing Machinery, 2018.