Dynamic Authentication: Developing an Alternative to Passwords (https://dynauth.io)

Connor Peters
Dept. of Computer Sciences
The College at Brockport
Brockport NY, USA
cpete4@brockport.edu

Dr. Ning Yu
Dept. of Computer Sciences
The College at Brockport
Brockport NY, USA
nyu@brockport.edu

Dr. Christine Wania
Dept. of Computer Sciences
The College at Brockport
Brockport NY, USA
cwania@brockport.edu

***Abstract*—Abstract here after the entire paper is finished**

I. Introduction

The reliability of passwords as a secure authentication scheme has been degrading rapidly since their digital inception in the very first multi-user operating systems. Due to Moore's Law and the general expansion of the Internet, passwords that were once secure are no longer, and the bar for what is an acceptable password is continually being pushed farther down a dark path.

The user experience when authenticating with a password has followed an identical route, becoming more and more unbearable with time. In short, as computers get faster, passwords must be more complex[1], and therefore user requirements more stringent. However, as noted by [**?**], the recall ability of humans has not changed at all with time, and most people's passwords are insecure regardless of the requirements [**?**].

This brings us to an interesting place in the world of cybersecurity. Users need to be authenticated of course, yet alternate schemes that would attempt to replace the password, such as biometric readers, have still not gained widespread prevalence across the Internet despite their growing ubiquity in end-user computing devices. Two factor authentication is just starting to gain traction, but users in general hate it [**?**] and adoption rates are low. Passwords are increasingly unfit for the job, yet there is no replacement.

But it's not all bad. Passwords *have* worked, at least enough to protect the majority of Internet users from malevolent cyber threats [**?**]. Let's give passwords a break for just a second to appreciate the benefits they provide.

1) Passwords are platform agnostic. Any device with an input method can take advantage of them.
2) Passwords do not require any additional hardware pieces for the users or the developers.
3) Passwords are free to use.
4) Passwords are relatively easy to implement for developers.
5) Passwords can be reset.
6) Passwords are marvelously familiar to users due to their prevalence on the Internet.

[1]By complex, I really mean "provide more bits of entropy".

When framed like that, passwords actually seem to be quite competent at their job of authentication. Yet time and time again, the opposite is proven to be true. The reality is that the critical flaw to passwords are humans themselves, not the technology they are built on. But technology is here to suit humans; if it doesn't work for us, it just doesn't work.

II. Dynamic Authentication

Dynamic Authentication, colloquially referred to as "dynauth" for this paper, is an authentication scheme designed to replace passwords. The core concept behind dynauth is to create an authentication scheme that attempts to retain the aspects of passwords that are beneficial, while eliminating the parts that aren't. Therefore, dynauth should be thought of as a sort of iterative improvement on typical passwords, rather than a new thing entirely.

Before I explain how dynauth functions, a brief description of how password-based authentication works is necessary to prevent confusion in the upcoming sections.

Firstly, password-based authentication relies on two main elements:

1) A publically known identifying "username", most often an email
2) A protected "secret", known only to the identifier, most often a password

The difference between a "secret" and a "password" is an important one to highlight. A user's secret is any piece of data that can be used to prove someone should have access to something. This can be a password, PIN, or security question. A password is a secret, but a secret isn't necessarily a password. The distinction is important to keep in mind as you read the upcoming description of dynauth.

Secondly, the password-based authentication scheme referred to in this paper is assumed to be following all of the industry standard password security guidelines as described by [**?**]. This is important to note because all comparisons to dynauth and passwords made in this paper assumes that the password is implemented perfectly. After all, it is far too common for an inexperienced developer to not properly implement authentication in their application [**?**].

A. Introduction to Dynauth

Dynauth, just like a password-based authentication scheme, requires an identifying username as well as an authenticating secret. The difference is that the secret the user must remember for authentication is not a password. Instead, the secret is a list of words (typically plain English words), known as "keys", associated with numbers, known as "locks".

A typical list of locks and keys might look like this:[2]

1) rough
2) mountain
3) biking
4) large
5) rocks
6) resulted
7) lengthy
8) costly
9) repairs
10) jeff

The numbers 1 through 10 being the locks, and the following words being the associated keys.

However, unlike password-based authentication, the user does not enter in the entirety of their secret to authenticate. Instead, after they enter in their username, the user is presented with 4^3 of their (numbered) locks in a random order, without repeat. The user then inputs the keys that are associated with the presented locks as one long string, in the same order as shown, without spaces or delimiters. Should the keys and locks match the ones on the list, the user will be authenticated.

If the inputted string does not match the keys and the locks, the presented locks will be randomly selected again and the user is not authenticated.

To further illustrate how dynauth functions, here are 2 example login sessions:

B. Example Usage 1: Successful Authentication

Using the same locks and keys as depicted in the list above, here is what a successful login session might look like:

1) Please enter your email:
   Input: cpete4@brockport.edu
2) Your locks are: 7 - 4 - 2 - 10
   Input: lengthylargemountainjeff
3) Correct! You are now authenticated

C. Example Usage 2: Failed Authentication

Using the same locks and keys as depicted in the list above, here is what a failed login session might look like:

1) Please enter your email:
   Input: cpete4@brockport.edu
2) Your locks are: 7 - 4 - 2 - 10
   Input: lengthlargemountainrepairs
3) Incorrect, please try again
4) Your locks are: 9 - 10 - 1 - 4
   Input: repairsjeffroughlarge
5) Correct! You are now authenticated

D. Memorization

Before getting into the technical intricacies that make dynauth a more secure alternative to the traditional password, I think it's best to address the issue of memorization. I can already imagine many people starting to consider tearing their hair out at the thought of *adding* complexity to an authentication process that is starting to fail *because* of increasing complexity. Requiring users to remember not just *one* secret but *ten*, in the proper order, that also changes every time they fail – well that's ridiculous.

The reality is this: a password with stringent requirements designed to make it more secure results in passwords that are far harder to remember than a simple list of 10 words. Humans are actually pretty good at remembering ordered information [**?**]. They aren't so good at remembering if their Apple ID password used their "normal" password with the uppercase at the beginning or the one with the exclamation point at the end. Due to this, people have successfully created passwords that are easy for computers to crack and hard for people to remember [**?**]. Dynauth navigates around this issue by not requiring users to use symbols and numbers at all, but to instead use common dictionary words.

Also, the fact that at face value it seems a more difficult task to remember 10 words in order than a passwords just encourages users to try harder to commit them to memory. As you'll see later, a simple practice feature developed for the usability test made a huge difference in how easily users were able to memorize their locks and keys.

Lastly, since dynauth is more complex than passwords, it is much less likely that a developer will incorrectly implement it and subsequently comprise the security of their users[4].

E. Design

This is where things get interesting. For all the same reasons as passwords [**?**], all user's locks and keys need to be hashed and stored in a safe and secure manner to prevent attackers from accessing cleartext entries of them if they can somehow get in the backend of the system. There also needs to be a way guarantee the security of each hash is greater than that of a typical password because otherwise, what's the point?

This presents a problem: any normal dictionary word cannot be hashed by itself and referenced later as it would be far too insecure. Therefore, the locks and the keys can't just be hashed and stuck in the database to be compared to the individually entered locks and keys later.

[2]This list of locks and keys is not necessarily an example of what a highly secure configuration may be like since complexity requirements for locks and keys can be altered depending on the use case, just like with passwords.

[3]The number 4 was a completely arbitrary number chosen for this implementation simply because it seemed reasonable, both in terms of memorization and security. This number is not set in stone, and more testing will have to be done to determine what the optimal number would be.

[4]This assumption comes from the fact that the implementation of dynauth isn't immediately apparent to most developers like passwords are. Therefore, they are more likely to research methods of implementation urging them to use an official library or OAuth (once they are built that is).

The core difference that allows dynauth to operate more securely, even in the event of a database breach, is the **hashed** storage of *all possible lock and key permutations*. This means that if a user configures 10 total keys, and are presented 4 total locks at the time of authentication (the base level configuration I chose), there will be a total of 10P4 (10 * 9 * 8 * 7 = 5040) permutations generated and stored.

Here is an example of a user's permutations being generated and hashed:

1) The first 4 keys of the user's configuration are concatenated.
   Example: roughmountainbikinglarge
2) That string is then hashed **Example:** 1CF0B384D1D52133255970AE0B091D5BDFCB627F EA9048D1FBC265BBF00137B7
3) The locks that those 4 keys are associated with are prepended to the hash string **Example:** 12341CF0B384D1D52133255970AE0B091D5BDFCB 627FEA9048D1FBC265BBF00137B7
4) That entire string is then hashed again, and the result is what is stored in the database as a single permutation **Example:** 0E60D213A1055A3F3D49BF4611D3307542615E53A 638751BAF50CF9E187228C9
5) This process would continue until all possible permutations of the user's configuration are generated and stored.

Here is the process for a user to authenticate:

1) The user enters in their email on the client side and the email is sent to the server.
2) The server randomly selects the locks appropriate for the user and stores them in a database with an expiration date and time.
3) The server then sends the same locks it stored back to the client for the user to view.
4) The user enters in the keys associated with the locks.
5) The locks are hashed client side and the hash is then sent to the server.
6) The server then prepends the keys stored previously in step 2 to the hash received from the client and hashes that entire string again.
7) The resulting hash is then used to iterate over the user's database of lock and key permutations until a match is found. If any permutation matches, the user is authenticated.

The reason this design was chosen was...

F. Benefits of Dynauth

a) Crack time greatly increased: The largest benefit dynauth provides is how much longer it would take to successfully crack[5]. The average password provides about 2^{22} bits of entropy [**?**]. Considering a worse case scenario, each key present in dynauth provides between 2^{11} and 2^{14} bits of entropy, depending on the words present in the dictionary used for a Dictionary Attack[6]. With a 10x4 schema, that means the average dynauth setup provides between 2^{44} and 2^{56} bits of entropy. These bits of entropy would also provide more protection than a typical password since they are hashed twice, first on the client side, then on the server side with the locks . This would mean an attacker would need to perform twice as many operations per guess, doubling the average amount of computation time needed to crack a single hash. On top of that, three distinct hashes would have to be cracked before an attacker retrieves all 10 keys[7].

b) Social engineering is less effective: It is much more difficult to phish a user since the system would have to know exactly how many locks and keys they use beforehand. Even in the event of a successful phishing attack, the attacker does not necessarily have immediate access since they would only have 4 our of the 10 possible keys.

c) Keylogging is significantly harder: Due to the fact that the keylogging system won't know which keys they retrieved are associated with which locks are displayed on the screen, keylogging is much more intensive. It is still possible, but does not provide immediate access to the user's account.

d) Possibility of infinite loop: **Not sure if I should mention this.** This is the idea that if a user is bruteforcing it, they can't assume that a wrong answer was actually not correct since it just wasn't correct for the displayed locks and keys while it still could be correct for another pair. This could result in the attacker never being able to get in.

e) It's new software: As with everything new, people need to adapt. That includes attackers; brand new software would have to be made to attempt to crack user's accounts. While this is similar to security by obscurity in that it isn't actually secure at all, it will slow attackers down initially.

f) Potentially strong enough to be used alone: Due to the additional bits of entropy provided, I would be more confident to say that a user could either reuse their locks and keys between accounts (as long as the accounts implement it correctly), or a single service could authenticate other services using this method with more assurance that there isn't a weak link of a master password.

G. Additional Possibilities

There are many additional little features that could be implemented with relative ease.

- A refresh button to reshuffle the user's locks on demand

H. On The Name

I am willing to admit that "Dynauth" or even "Dynamic Authentication" might not be the most ideal name for such a mechanism due to the ambiguity around the word "dynamic". The intention was to convey that the login process is dynamic

[5]By "crack" I mean guess all the possibilities against a hash, NOT crack the hash itself

[6]It is assumed the attacker will be using a dictionary attack here because that is the worse-case scenario. Trying to guess the final hash character by character results in excess of 2^{120} bits of entropy

[7]This is the case since each hash contains only 4 of the 10 words. The attacker would need to crack hashes for 1234, 5678, and 87910.

in the sense that the user types a different thing between sessions and that the secret is changed on a failed login attempt. It was suggested to name it "Active Authentication" due to the fact that the user needs to "actively" think about the process every time they authenticate, reinforcing the memorization of the locks and keys. Despite the nice alliteration, I decided to keep the "dynamic" due to the fact that I had already bought the domain name "dynauth.io" and I did not want to change it for a small difference.[8]

III. Implementation

The obvious next step after developing the framework of dynauth was to implement it in a usable and extensible way to use as a testbed for further research. A live example utilizing the code written during this study is available online at https://dynauth.io.

A. Method of Implementation

As is common practice in software development, I broke dynauth into two de-coupled sections, the "backend" and the "frontend". The backend was written entirely in Golang[9] and the frontend was written in TypeScript using the Angular 5[10] framework. I hosted the backend and frontend on separate AWS VMs and installed free signed HTTPS certificates using Let's Ecrypt[11].

B. Backend

The backend of the system was designed as a REST-like[12] API that issues JSON Web Tokens[13] to users after a successful authentication attempt to identify the user to the API in a stateless way.

The backend of this implementation is perhaps the most important aspect of this project because:

1) It provides a testbed to analyze the real world security benefits of dynauth
2) It provides a testbed to benchmark the performance and compare it to other authentication schemes
3) The REST-like design forced me to consider every HTTP request sent over the Internet and refine the authentication process

Using Golang as the sole server-side language provided some huge advantages during the actual development cycle. Specifically, having a strongly typed language and garbage-collected language helped ensure that the processing was fast and reliable. The speed of development also helped a lot since I had a very limited amount of time to pull this project together.

[8]Humans are stubborn

[9]https://golang.org

[10]https://angular.io/

[11]https://letsencrypt.org/

[12]I describe it as "REST-like" due to the fact that the API is not entirely stateless. Once the user initially sends a login request to retrieve the random locks from the server, those locks are stored in order to be used again during authentication.

[13]JWT landing page: https://jwt.io/, JWT RFC: https://tools.ietf.org/html/rfc7519

C. Frontend

The frontend of the system was just a relatively simple JavaScript application that made asynchronous requests to the API and presented a clean and interactive form with for users. The app stored a cookie with basic user information to allow the user to maintain authentication for a set period of time, just like a normal web based service.

The only thing worth talking about when it comes to the frontend was the client-side hashing. To round out dynauth, it was decided between me and Dr. Yu to have the client hash all lock and key information despite it being encrypted with TLS as well. This did not cause any noticeable load on the user's device as long as the permutation number was kept to a sane amount[14].

D. Challenges During Implementation

1) The permutation generation could cause lots of server load. The reason I did not use a scheme like Bcrypt or Scrypt for hashing was because of the insane amount of load that would result in on the server when servicing multiple users.
2) Authenticating every HTTP request using JSON Web Tokens was a new thing for me, and took a bit of time to understand correctly. I ended up using a free middleware package written in Golang to authenticate every request[15].
3) I designed each hash permutation to be inserted into the MySQL database as a single huge insert statement rather than X amount of permutations as separate statements. This worked wonderfully from a speed point of view, however, if the number of user keys exceeded 13, MySQL would reject the insert statement for being too large.

IV. Validation

A. Analysis of Implementation

B. Usability Testing

When it comes to authentication, user experience is of paramount importance; It would be trivial to make passwords secure by simply requiring them to be 20+ characters. However, we have learned over time that doing so would not actually result in anyone being more secured due to the compromises that would inflict upon the users [**?**]. Having some sort of authentication scheme that integrates well across domains, across different user demographics, and provides consistent security is the goal.

Dynauth was designed to be similar to passwords, yet more secure and more extensible.

[14]I tested schemes up to 20x5 and did not have any issues on my laptop or mobile device. More testing is needed before this can be confirmed to be a good idea though.

[15]Citation Needed

C. Method of Usability Testing

Unfortunately, I was only able to perform a small-scale usability test at this point[16] due to timing issues.

The test was run as a part of Dr. Christine Wania's Human Computer Interaction (CIS404) class at SUNY Brockport and consisted of 18 college students split into two groups of 9. Group 1 was the control group representing typical password usage and group 2 was representing dynauth and using a 10x4[17] schema.

Each user was preregistered with their student emails and randomly assigned to a group until the groups were evenly assigned.

1) Each preregistered user would initially login without any sort of authentication (just their email) and read through a brief tutorial with memorization tips
2) Each user in Group 1 would then configure their password[18]. Each user in Group 2 would then configure their 10 keys associated with numbered locks
3) Each user was asked to practice logging in 10 times
4) Each user was asked to logout and fill out a questionnaire regarding this activity
5) They were then asked to login again and fill out a similar (but different) questionnaire for 2 more consecutive weeks

Every user interaction was tracked including the length of their passwords/keys, how many times they failed during login, how long it took them in milliseconds to login, and if they refreshed their locks at all.

D. Results of Usability Testing

Still working on this...

V. Conclusion

Still working on this... The main takeaway for dynauth as a new authentication scheme is this: *it forces users to use what are essentially secure passwords in a memorizable way*

Acknowledgments

A thanks is in order to the wonderful faculty at SUNY Brockport who were willing to jump in and help guide me on this project:

- Dr. Ning Yu, for providing your positive spirit and cybersecurity chops to this project. "Just implement it first, don't worry about the paper yet. Just implement it."
- Dr. Christine Wania, for guiding me when it comes to usability testing and allowing me to hijack an assignment in your class for my own personal gain. "This is normal human behavior."

[16] A larger "Alpha" test is in the works. More information at https://dynauth.io

[17] 10 total keys, 4 locks displayed during each authentication session

[18] It was required to be "strong" as decided by the helpful ZXCVBN library https://github.com/dropbox/zxcvbn

References

[1] Bonneau, Joseph, et al. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. 2012 IEEE Symposium on Security and Privacy, 2012, doi:10.1109/sp.2012.44.

[2] Citation needed