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

As the Internet has matured, the ubiquitous use of passwords across the web to secure user accounts has resulted in passwords becoming a cornerstone of the Internets cybersecurity infrastructure. However, between the constant use of passwords required by web-based services and the steady increase of computing power, a sort of paradox has developed; a password needs to be long and random to be secure and a password needs to be easily memorizable. These things are mutually exclusive and are the reason most services now require users to create passwords with a minimum character count and special symbols to guarantee password length and complexity. These restrictions have resulted in a terrible user experience to most people, tech savvy or not. I end up resetting my passwords at least once every couple of weeks.

Passwords as a concept alone are not that bad. They actually have some amazing advantages in the world of authentication. They are marvelously familiar to users while being completely platform agnostic and relatively easy to implement[1] and reset. The positive aspects of passwords as an authentication method are not present in other forms of authentication such as biometric fingerprint scanning or hardware USB keys. Using hardware devices over the Internet is risky in general, and doing so reliably and securely just adds to that complexity. Not to mention, every hardware component you add costs money somewhere down the chain, in many cases hitting the users themselves. Passwords do the gargantuan job of web authentication better than anything else currently, yet, they are still so problematic.

The purpose of this project is not to expose the insecurities of passwords, as that is already well documented [**?**]; rather, the purpose of this project is to propose and validate a new knowledge-based authentication scheme as a replacement to the everyday use of passwords.

II. The Prevalence and Problems of Passwords

Weren't passwords supposed to die a decade ago [**?**, Need to cite]? Why are they more prevalent now than ever [**?**]? From my observations, along with the massive growth of the Internet, there are 3 main reasons:

1) Lack of a viable alternative
2) Familiarity of use
3) Ease of implementation

A. Lack of a Viable Alternative

Developing alternative to passwords is not a trivial process. There are many MANY factors to consider [**?**]; from user experience to cross-platform compatibility to ease of implementation, passwords will not be easy to root out entirely. It might actually be something that can never feasibly happen due to the vastness of their implementation. As I mentioned earlier, the main competitors to passwords work well for some use cases-smartphones for example-but they are not as widespread and frankly don't do the job as well as passwords in many situations. The only **actual** alternative to passwords at this point is a password manager application which isn't actually a replacement at all, just a bridge to fix some of the problems.

B. Familiarity of use

When accessing any sort of website, people expect passwords. If they were presented anything else[2], they would be confused and not understand what they were seeing. They wouldn't know how to use it, they wouldn't trust it, and they would most likely just leave altogether.

An interesting example of this is the process to authenticate into my credit card provider's online system. The password requirement are exceedingly stringent, requiring something like 14 characters with all the symbols and goofiness you would expect. On top of that, the system also requires you to choose an abstract image along with one of a dozen or so predefined hints[3] to assumedly make up for the complex password. However, a picture of the internals of a Swiss watch and the phrase, "Doubtful Cog" does absolutely nothing to prevent me from resetting that password once every couple of months.

[1]To developers who understand the complexities at least.

[2]With the possible exception of using a smartphone finger print scanner

[3]Some examples are: "similar washer", "unique clothier", "interesting tailor". I have yet to figure out the intended use for such abstract phrases...

C. Ease of implementation

Passwords are all software, and when it comes to websites, platform agnostic software at that. There are literally thousands of online tutorials depicting how to implement passwords correctly (ie. hashed and salted) and the technology used is almost always Open Source and free for commercial use. No other authentication method comes close to that level of support.

D. Need for a New Method of Authentication

While passwords do have some wonderful qualities, their widespread use has resulted in some catastrophic user experience issues:

1) Most users reuse the same password(s) over and over [**?**]
2) Most users write all of their passwords down [**?**]
3) All users forget their passwords at some point [**?**]

There are also many easy to execute attacks that can render passwords useless or worse:

1) Bruteforcing, ie. guessing all possible combinations until a match is found[4]
2) "Smart"forcing[5], ie. guessing a friends password because it's their dogs name and anniversary date
3) Social engineering ie. phishing/spear phishing or convincing someone to remotely give you access
4) Keylogging, you don't know what's installed on someone else's computer[6]
5) Shoulder surfing, ie. seeing someone type in their password

Both the user experience issues and security flaws that are inextricably tied to passwords have gotten to such a head that it is fairly well agreed upon [**?**] that something needs to be done.

E. Basic Traits Necessary to Replace Passwords

This needs to be filled out still, a lot of the information is available at [1]

III. Enter: Dynamic Authentication

"Dynauth"[7], is a portmanteau of dynamic and authentication, and will be used colloquially to refer to dynamic authentication for the rest of this paper.

A. Introduction to Dynauth

Dynauth is designed to be a password replacement authentications scheme that fixes many of the problems inherent to passwords, while being similar enough to be adopted easily. The end goal for dynauth is to implement the latest OAuth protocol and over a service to authenticate other apps over the Internet and prevent needing multiple logins.

[4]A password can ALWAYS be guessed.

[5]I made this up

[6]Many people don't know what is installed on their own computer

[7]Thanks for the idea Microsoft

Locks	Keys
1	ant
2	beetle
3	cat
4	dog
5	eagle
6	fish
7	goat
8	hare
9	ibis
10	jackal

TABLE I
A SAMPLE TABLE OF LOCKS AND KEYS

B. Overview of Usage

A user configures a table of numbered "locks" that correlate to strings of text (typically plain English words) known "keys". These locks and keys are similar to a password in that they are the user's "secret", and must remembered for authentication. Table 1 depicts what a lock and key setup might look like.

Note: this table is not necessarily an example of what a typical configuration may be like since complexity requirements for keys have not been nailed down yet. It seems that typical dictionary words with a minimum length of 3 are sufficient for most use cases, but that has not been confirmed.

Once the user configure their locks and keys, they are ready to login. A typical login sessions goes like this:

1) The user enters in their username (a valid email address, for this implementation)
2) The user is presented 4[8] of their locks in a random order, without repeat
3) The user inputs, as one long string without spaces or delimiters, the keys that correlate to the randomly chosen locks in the same order

C. Example Usage 1: Successful Authentication

Using the same locks and keys as depicted in Table 1, here is what a successful login session might look like:

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

D. Example Usage 2: Failed Authentication

Using the same locks and keys as depicted in Table 1, here is what a failed login session might look like:

1) Please enter your email:
   - cpete4@brockport.edu
2) Your locks are: 3 - 6 - 8 - 9
   - catfishharejackal[9]

[8]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.

[9]Notice the word "jackal" is for lock number 10, not 9 as required

3) INCORRECT: Your keys do not match, not authenticated. Please try again
4) Your locks are: 9 - 4 - 2 - 1[10]
   - ibisdogbeetleant
5) Correct! You are now authenticated

E. Usage Details

The user's locks will initially not be presented to them until a registered email is submitted since the locks are grabbed from the server, not generated client side. Also, every time there is a failed authentication attempt, the locks will reshuffle themselves, making another server call. For this implementation, users were given the ability to "refresh" their locks at will and no maximum number of failed attempts. These features are easily implemented, however, more research will need to be done before the necessity of additional restrictions like these is clear.

F. Hashing

For all the same reasons as passwords [**?**], the locks and keys need to be hashed and stored in a safe and secure manner. It was decided that all the information sent between the client and server not only needs to be TLS encrypted, but also hashed itself to keep the integrity of dynauth as an authentication scheme[11]. The latest and still secure as of June 2018 hashing schemes of SHA2-256 and SHA3-256 were used in this implementation since the 256 bit models provides a great amount of security while taking up less data.

G. Information Storage

One of the main problems with the usage of passwords is the fact that no matter what hashing and salting scheme is used, there is always a one-to-one relationship of username/email to password somewhere in a database. This allows any attacker who gains access to compile massive tables of passwords quickly to crack through at a later time.

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.

An example of the permutations being generated:

1) The user configures the locks and keys outlined above
2) The first permutation that is generated is the one for the first 4 **keys**: "antbeetlecatdog"
3) That permutation is then hashed[12] into: "0fba3db0d135d8db6798011404194e398cf5d9de5ff3be25937d911cc279cf56"
4) The associated **locks** are then prepended onto the hash: "**1234**0fba3db0d135d8db6798011404194e398cf5d9de5ff3be25937d911cc279cf56"
5) That new string with the prepended locks is then hashed yet again to ensure that the user can't just enter any of their keys in response to the locks[13]: "06d9cd066d2d6fe9849b2a76b3bc90c28c5e2ba43c4dbb57b6bfac793e6b20df"
6) That final hash is then stored in the database along with the hash salt, and the next permutation would being the same process until there aren't any left.

H. Benefits of Dynauth

a) Crack time greatly increased: The largest benefit dynauth provides is how much longer it would take to successfully crack[14]. 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[15]. 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[16].

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.

[10]Notice that the locks changed automatically upon failure.

[11]This has resulted in the unintended but welcome effect that anytime a user needs to reset one key, they must re-enter all the others previously since the system cannot pick and choose what is changed in the hashes due to the design.

[12]As mentioned above, using SHA2-256

[13]This time, the hash is SHA3-256 with a 64 character salt that is stored right next to the final hash in the database. NOTE: The salt is not represented in the hashes in this example.

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

[15]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

[16]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.

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.

I. 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 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.[17]

IV. 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[18] and the frontend was written in TypeScript using the Angular 5[19] framework. I hosted the backend and frontend on separate AWS VMs and installed free signed HTTPS certificates using Let's Ecrypt[20].

B. Backend

The backend of the system was designed as a REST-like[21] API that issues JSON Web Tokens[22] 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.

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[23].

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[24].
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.

[17] Humans are stubborn

[18] https://golang.org

[19] https://angular.io/

[20] https://letsencrypt.org/

[21] 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.

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

[23] 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.

[24] Citation Needed

V. 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.

C. Method of Usability Testing

Unfortunately, I was only able to perform a small-scale usability test at this point[25] 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[26] 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[27]. 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...

VI. 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*

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

[26] 10 total keys, 4 locks displayed during each authentication session

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

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."

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