Acknowledgements

I want to first thank God for always being with me.

I am very grateful to my advisor Abel Sanchez for guiding me through my research and providing me with interesting readings that helped shape the ideas behind this thesis. To John Williams for providing his support and ideas that pushed this project much further.

Thanks to my girlfriend Rosita who has been the closest person to me during all my time at MIT, even though we are thousands of kilometers apart. Without you, I probably wouldn’t even be here!

My parents who always believed in me and have helped me in my quest of every goal I work for. To Ingrid and Fernando who invited me to their home constantly and showed me everything Massachusetts has to offer. And my godparents that are always exited to participate in what I do.

I will forever remember all the friends that I met here and made my experience in SDM an amazing adventure. They gave me perspective and can turn any day into a fun and wonderful one. And all the SDM staff that always made me feel welcome in our home away from home.

I greatly appreciate my friends from home that have been here sharing the beautiful cities of Cambridge and Boston: Sofia and Carlos for all that we lived together and for making sure I always had desserts at home, and Malena for being a good friend, bringing years of experiences together and being my #1 fan and cheerleader.

I want to send a special mention to CONACYT and I2T2 who helped me fund this dream.

Thanks to everyone involved in the NAS project. To IARPA for all their generous funding. To Eric and Bruce for helping make sure our code worked, for sharing your knowledge and listening to our ideas. To the team at the University of Memphis: Denise, Dr. Dasgupta, Sanjib, Kul and Abhijith that started working on negative authentication way before I did.
Password Secured Systems and Negative Authentication

by

Alvaro Madero

Submitted to the System Design and Management Program
on May 23, 2013 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Engineering and Management

Abstract

Today's industry, government, and critical infrastructure are dependent on software systems. In their absence, our modern world would come to a stop. Given our dependence, the mounting cyber threat is of critical concern. In the majority of our systems, passwords are the keys to the system. Unfortunately, there has been little innovation and vulnerabilities keep mounting. Even with established and well known defenses, no authority has emerged to establish policies or laws that guarantee their implementation. The response has been more complex passwords. This is not working. This thesis presents the state of the practice in password systems and introduces work in Negative Authentication and its implementations.

Thesis Supervisor: Abel Sanchez

Title: Executive Director Geospatial Data Center
Table of Contents

Acknowledgements ........................................................................................................ 2
Abstract ......................................................................................................................... 3
List of Figures .................................................................................................................. 6
List of Tables .................................................................................................................. 6
Chapter 1 Introduction ..................................................................................................... 7
Chapter 2 Password security .......................................................................................... 9
  2-1 What is password security ....................................................................................... 9
  2-2 Brief history of password security .......................................................................... 11
  2-3 Methods used to secure passwords ....................................................................... 12
    2-3a Plain-text databases ......................................................................................... 13
    2-3b Hashing ............................................................................................................ 14
    2-3c Salting ............................................................................................................. 15
    2-3d Stretching ....................................................................................................... 17
    2-3e Policies ........................................................................................................... 19
  2-4 Layers of security ................................................................................................... 21
  2-5 Other methods ...................................................................................................... 22
    2-5a Two-factor and Multi-factor authentication .................................................... 22
    2-5b Threshold Cryptography .................................................................................. 24
    2-5c Negative Authentication .................................................................................. 26
Chapter 3 What is happening in the real world? .............................................................. 28
  3-1 Security Reports .................................................................................................... 28
  3-2 Policies, Regulations and Guidelines .................................................................... 33
  3-3 Cases where password databases have been leaked ............................................ 34
  3-4 Other advances in password cracking .................................................................. 35
Chapter 4 Negative authentication ................................................................................. 37
  4-1 Our approach to Negative Authentication ........................................................... 38
  4-2 Different implementations .................................................................................... 38
List of Figures

Figure 2-1 Architecture of a Password Security System .......................................................... 9
Figure 2-2 Flow diagram of a basic Password Security System: User Validation ................. 10
Figure 2-3 Process of salting and hashing a password........................................................... 16
Figure 2-4 Relative Size of Password Keyspace. ................................................................... 21
Figure 2-5 Sample of RSA SecurID Hardware and Software Authenticators .................. 23
Figure 2-6 RSA Distributed Credential Protection ................................................................. 25
Figure 2-7 Negative authentication layers .................................................................................. 27
Figure 3-1 How data breaches occur .......................................................................................... 29
Figure 3-2 Variety of hacking actions in data breaches ............................................................ 29
Figure 3-3 General Malware Threats: New Password Stealers Samples ............................. 30
Figure 3-4 Variety of social actions in data breaches ................................................................. 30
Figure 3-5 Data Breaches Made Public ..................................................................................... 31
Figure 3-6 Variety of compromised data in data breaches ....................................................... 31
Figure 3-7 Timespan of data breach events .............................................................................. 32
Figure 4-1 Negative Authentication - Positive and Negative Space ....................................... 37
Figure 4-2 Negative Authentication: Real space – Detector Creation .................................. 40
Figure 4-3 Negative Authentication: Binary - Hamming Distance ........................................ 41
Figure 4-4 Negative Authentication: Grid - Detector Creation .............................................. 43

List of Tables

Table 2-1 Plain-text and Encrypted Password examples ......................................................... 14
Table 2-2 Hashed Password example ....................................................................................... 15
Table 2-3 Salted Hashed Password example .......................................................................... 17
Table 2-4 Stretched Salted Hashed Password example ............................................................ 18
Table 2-5 Possible Keypaces by Password Length and Character Set Size .......................... 19
Chapter 1 Introduction

Passwords are used in almost every digital system. Everyone is familiar with logging-in to a PC operating system, accessing their email or signing-in to a social network, but passwords protect access to many other systems like banking terminals and websites, network devices like switches and routers, corporate applications and even entertainment services hooked up to our TVs. Most vendors have been able to implement their own approach to safeguarding those passwords and this has resulted in a mix of safe and insecure systems.

Maintaining password security is a problem of ever increasing importance. With the increase in online services, the amount of credentials per person has also increased dramatically. News reports of hackers stealing credentials are increasing and so are notices to users requesting password changes. Through this research I’m trying to find how attackers are able to obtain these passwords. The research focuses on the state of the practice and the effectiveness of defense of the password security implementations in preventing the disclosure of the password to an unauthorized party. This does not go into other system vulnerabilities in network, operating system or database.

Although comprehensive security is needed, I'll focus on why it's possible to break the passwords and not on how they gained entrance into the system to get the password data. To properly understand some of these issues I will first describe what password security is. Afterwards I will continue with a brief history of password security and evolution in recent years.

Once the big picture has been established, I will detail the most common components and methods used today in implementing password security. I will go over some of the benefits and disadvantages of each method and how they are linked with each other as a measure taken to try to overcome their major drawbacks. To further strengthen the level of security that the usual methods provide, there have been developments in other approaches to authenticate the users. I will go over some of the most common ones that are already available as well as introduce a new approach called negative authentication.

The third chapter presents a review of policies, regulations and recent security reports by important cyber-security labs to establish the current situation and its impact on the
industry. Followed by specific high profile cases where password security has been compromised and some of the reasons why this happened.

Having reviewed the situation, I will detail the implementation of negative authentication we have been working on. There are different approaches that can be tried and each can have some advantages and disadvantages. This will allow you to better understand its potential advantages.

Finally, I will present my conclusions of the analysis, with some recommendations on how to increase the safety of password security systems. I will review some of its implications and the future research that can be done in this subject.
Chapter 2 Password security

2-1 What is password security

Password security is the set of mechanisms or approaches used to protect the characters that compose a password and keep them secret from others. According to Morris & Thompson “It is of course only one component of overall system security, but it is an essential component.” (Morris & Thompson, 1979)

A good password security system will reliably grant access to a user that provides the correct set of characters and make it very hard for unauthorized users to obtain the original password. “Good system security involves realistic evaluation of the risks not only of deliberate attacks but also of casual authorized access and accidental disclosure”. (Morris & Thompson, 1979) To achieve this, the system architecture involves many layers and components that all contribute to guard the personal information of the user. The whole system needs to handle the following aspects of a user's credentials:

- Length and characters allowed
- Login prompt
- Transmission method
- Validation process
- Secure storage

![Character policies diagram](image)

![Login prompt and User Validation diagram](image)

![Credentials Table diagram](image)

Figure 2-1 Architecture of a Password Security System
For this thesis the focus is going to be mainly on how the credentials are stored and validated. Weaknesses in any other part of the system present a risk for the system to be compromised allowing unauthorized access or for data to be stolen. Figure 2-1 shows a simplification of the different components involved in handling passwords. In this diagram some of the best practices are represented like password character policies, hashing and salting the password. In the following chapters each of the relevant components will be discussed along with some of the benefits and disadvantages of their use and common practices. Figure 2-2 shows the User Validation diagram in detail. This has been the basic architecture of password security for many years.

![Flow diagram of a basic Password Security System: User Validation](image)

One of the most important aspects of the storage is the hashing function used. Because this is a complex calculation, this is the main component that determines the time it takes to store and compare passwords. Since there is no way to reverse a password hash to get the original string, an attacker must try many possible combinations with the hope that in time he’ll find the correct one. Therefore the amount of time it takes to hash each attempt becomes the critical factor determining the speed of hacking a password database. (Morris & Thompson, 1979)

Before going into further detail, I will talk about the history of password security to get the overall perspective through time.
2-2 Brief history of password security

Password security has been an issue for many years. The same problems that we are facing now with the speed at which attackers can test passwords have existed for decades. Every round of development and innovation is an incremental change to the complexity of the algorithms and the speed it takes to calculate them.

The encryption algorithms used are too complicated for a human to calculate, but computers can do it in fractions of a second. They can do it so fast, that checking many combinations per second is feasible with available computers. Whole dictionaries are used as input to try to guess the passwords. Random salts are added to the user input to prevent multiple uses of the same password to produce the same digest. Networked systems that divide and conquer the work and hardware accelerators reduce the time needed to crack the database by more than a 100 times. Even though these statements reflect the realities in 2013, all of these concerns have been around for 35 years. (Peslyak & Marechal, 2012)

At least since the 70’s we have known of the problems with fast hashing and encryption algorithms and the benefits of using salts. The first UNIX algorithms derived from WWII encryption machines, took milliseconds to calculate. Dictionaries of 250,000 words could be used to crack passwords in minutes. Introduction of more advanced encryption algorithms dramatically increased processing time and addition of a random salt forced checking of each hash on the list to be done independently. In post-1976 versions of UNIX the DES encryption algorithm used one way functions, 12 bit salting and stretching with 25 iterations. (Klein, 1990)

Klein went on to describe advances in the then current technologies of 1989 where computers were readily available and networks allowed the cumulative processing power to be harnessed. The advance in speed of processors allowed once again checking the whole dictionary in around 5 minutes. Hardware processing of algorithms could decrease encryption time to microseconds allowing going through the whole dictionary in 1.5 seconds.

“The problem we find with cryptographic hashing functions is that whilst computing power continually increases over time, password entropy (or randomness) does not.” (Dunstan, 2012) As time goes on, Moore’s Law suggests that computing power doubles every two years.
We have seen every generation of one way hash functions become less useful because they can be calculated too quickly.

In 1999 Provos and Mazières (Provos & Mazières, 1999) proposed an adaptable hashing algorithm that could increase in complexity with a variable parameter they labeled as cost. In their work they proposed a new key derivation function named bcrypt that implements salting and stretching into the hashing function. There are currently many implementations of bcrypt for the different programming languages and it could become one of the next generation of commonly used hashing algorithms. (Dunstan, 2012)

Decades after the initial concerns appeared, major system vendors have not taken the precaution of investing in implementing and promoting password security best practices. “Windows 95 did not support case-sensitivity in its passwords. This meant that if the user set his password to “PassWord1”, you could use password1, PASSWORD1, PassWord1, and so on to log on.” (Microsoft Support, 2006) It wasn’t until the 1996 release of Service Pack 2 for Windows NT 4 that it introduced password policies to require at least 6 characters with at least one character from three of four types (lowercase, uppercase, numbers, symbols). (Microsoft Support, 2006) Even in recent systems like Windows 7, we find that the implementations lag on some features. Windows still uses old hashing algorithms without salting or stretching. “Neither the NT hash nor the LM hash is salted... Windows has never stored hashes in human-readable form, so there has never been a need to salt them.” (Microsoft, 2012)

2-3 Methods used to secure passwords

As computer systems became more prevalent, the need to protect passwords became clear. There are a few methods that are commonly used to protect these systems. To properly store a password in a file or database, one or many of these methods are applied to the original password string. The resulting output is what is ultimately written to the storage and to properly authenticate, the system must apply the same methods to a user’s credentials when they try to log in. Every one of these approaches may be used in conjunction with others to strengthen the security that they provide, but some of them are not useful if used by themselves.
2-3a Plain-text databases

A plain-text database is one type of database where the information is stored in a way that can be read back with the original meaning. The main benefit of this approach is that storing passwords in plain-text is the simplest method. It essentially means writing the characters to a file or database. When a user logs in, the password that is entered is compared to the string stored and if they match, the user is allowed access.

One feature of this means of storage is that the password is recoverable if the user forgets it. For a long time, online services would email the password back to the user if requested. A potential security threat is that the mechanism used to retrieve the passwords could be maliciously employed to gain access to them. Additionally, the method of sending the forgotten passwords to the user was most commonly text emails that could be intercepted and it would reveal the password to an unauthorized party.

The main disadvantage from a security perspective is that the data for everything stored can be read back. After gaining access to the file, there is nothing preventing the reader from obtaining the password information.

Table 2-1 shows an example of how the plain-text database looks like. You can clearly see the passwords that the users have chosen. The second table shows what an encrypted database might look like. Since this example is encrypted by each entry, those that share the same password have the same encrypted string. The passwords that are very similar to each other in this example also share a large portion of the encrypted string. Even though it looks like gibberish right now, if the encryption algorithm and key are known, they can be decrypted to reveal the original values.

Some services employ encryption in their databases to prevent unauthorized access to the passwords. In some cases the whole database might be encrypted and in others, it's only the password field. While this changes the data into what may seem unreadable at first sight, the resulting output can always be decrypted and turned back into the original text. Therefore encrypting the database is not as secure as other methods since anyone can decrypt it if they have access to the right encryption key.
2-3b Hashing

A hash is a one way function. It is a complex set of operations that regardless of input, produces a fixed length result called a digest. A good one way function will be practically impossible to reverse, meaning that obtaining the input from the digest cannot be done.

When a password database implements hashing, the entries in the database contain the digests for the passwords and the plain-text passwords should not be stored anywhere. When a user needs to authenticate, the password that he supplies will be hashed using the same algorithm that the database used. The resulting digest is compared to the one stored and if they match, the user is allowed access. In essence one-way functions can be used to hide real passwords but retain the ability to authenticate a user. This is the main benefit of using hashes and is the reason why they are the most valuable part of password security.

An important feature of a hashing algorithm is that small differences in the input should create a very different digest. This will make the deciphering of the digest much more difficult since there should be no relationship between input and output.

One of the weaknesses of using hashing just by itself is that given the same input and same algorithm, the digest will be the same. (Nielsen, 2012) In all the implementations that use
the same algorithm, when two or more users choose equal passwords, the digest resulting from that password will be identical. Anyone with access to the digests can compare them and know which ones represent the same password. This could be done between different users in one system or between different systems. Every user that shares the same password will also share the same digest. With a sufficiently large database, the list can be sorted by popularity of digests and then compared to commonly used passwords.

Table 2-2 shows the original set of usernames and passwords followed by the table with the digest for each entry. The first and second users share the same password and therefore the digest for each one is the same. The third and fourth users have very similar passwords, but good hashes produce very different outputs with even small changes in the input. In this example you can see that the resulting digests are very different from each other. There is no way to revert to the original password strings by using the digests. To find the original passwords we would need a brute force attack trying different combinations until one resulted in a match.

<table>
<thead>
<tr>
<th>User</th>
<th>Original Password</th>
<th>User</th>
<th>SHA-1 Hashed Password (Digest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rose</td>
<td>bluehouse</td>
<td>rose</td>
<td>5b3d84a20f0d70d023f9a56f8d8e73f8471885d</td>
</tr>
<tr>
<td>mary</td>
<td>bluehouse</td>
<td>mary</td>
<td>5b3d84a20f0d70d023f9a56f8d8e73f8471885d</td>
</tr>
<tr>
<td>alvin</td>
<td>helloworld</td>
<td>alvin</td>
<td>6adfb183a4a2c94a2f92dab5ade762a47889a5a1</td>
</tr>
<tr>
<td>jack</td>
<td>hellowormd</td>
<td>jack</td>
<td>7fc7fb5565ca20ea97a7916175f897bbd6e29976</td>
</tr>
</tbody>
</table>

Table 2-2 Hashed Password example.

2-3c Salting

In cryptography a salt represents a set of data added to the message before it is hashed. When applied to passwords, it is generally a set of characters or bits that can be appended or prepended to the password before going through the hashing function. The salt makes the password longer than it originally was and makes the input string to the hashing function different from each other in systems or users that chose the same password. The characters can be static, meaning they are the same for every entry, or random, meaning generated
differently by the server for each user. Adding variability to passwords before hashing differentiates the digest and will hide when users share passwords.

The password database will include the digest of the password that was incremented with the salt. To obtain the same digest, the authentication process needs the same salt. Therefore the salt must be stored in an accessible place so that when a user tries to authenticate, the process will use the salt on the supplied password before hashing and then compare the resulting digest to the one stored. Figure 2-3 shows the basic process that is followed when salting and hashing a password.

![Figure 2-3 Process of salting and hashing a password](password-hash-salt.png)

A salt is not intended to be secret and is normally stored with the digest. This results in the digest field in the database being longer than the digest length of the hashing function. An important benefit of using salts is that they increase the amount of time it takes to guess a password database, since each salt has to be tested independently.

The main issue with the use of salts is that there is a lot of confusion and misunderstandings around them, how they are used and how they are stored. But in reality they are not hard to implement and provide enhanced security without much processing and space const.

Table 2-3 shows the salted and hashed password how it would be stored in the database. The salts were randomly generated of 24 bits in hexadecimal format. Since not all
hexadecimal values map to viewable ASCII characters, the whole process is shown in
hexadecimal format. After hashing the password with the salt, the salt is appended to the
digest so that it can be read and used when authenticating a user. The length of the SHA-1
digest is 40 hexadecimal characters (160 bits). The database now has 46 characters so the
remaining 6 (24 bits) are the salt.

<table>
<thead>
<tr>
<th>User</th>
<th>Original Password</th>
<th>Salt</th>
<th>Salted Hexadecimal Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>rose</td>
<td>bluehouse</td>
<td>f75982</td>
<td>626c7565686f757365f75982</td>
</tr>
<tr>
<td>mary</td>
<td>bluehouse</td>
<td>eb5070</td>
<td>626c7565686f757365eb5070</td>
</tr>
<tr>
<td>alvin</td>
<td>helloworld</td>
<td>9333dc</td>
<td>68656c6f776f726c649333dc</td>
</tr>
<tr>
<td>jack</td>
<td>hellowormd</td>
<td>7c9c62</td>
<td>68656c6f776f726d647c9c62</td>
</tr>
</tbody>
</table>

User Original Password

<table>
<thead>
<tr>
<th>User</th>
<th>Hexadecimal Password</th>
<th>Salt</th>
<th>Salted Hexadecimal Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>rose</td>
<td>626c7565686f757365</td>
<td></td>
<td>626c7565686f757365f75982</td>
</tr>
<tr>
<td>mary</td>
<td>626c7565686f757365</td>
<td>eb5070</td>
<td>626c7565686f757365eb5070</td>
</tr>
<tr>
<td>alvin</td>
<td>68656c6f776f726c64</td>
<td>9333dc</td>
<td>68656c6f776f726d649333dc</td>
</tr>
<tr>
<td>jack</td>
<td>68656c6f776f726d64</td>
<td>7c9c62</td>
<td>68656c6f776f726d647c9c62</td>
</tr>
</tbody>
</table>

Table 2-3 Salted Hashed Password example

Hashed and Hex with: (FileFormat.Info) Random with: (Random.org)

2-3d Stretching

When using hashing algorithms for password protection, there is another issue that
decreases the effectiveness of the security. These algorithms were developed to be very fast so
that they could also be used to compare larger messages or files. (Nielsen, 2012) The amount of
time it would take to calculate a hash was measured in milliseconds, therefore thousands of
hashes could be calculated per second. This enables someone to check many different
combinations of input and compare it to the digest to find passwords. (Peslyak & Marechal,
2012)

Stretching is the technique involving repeated use of a hashing algorithm to increase the
time taken to calculate the digest. The password would be used as input and a digest is
calculated, then the digest would become the input for the next iteration and this process is
repeated many times. Some algorithms implemented a fixed number of iterations and some allow for a variable number to be given as input to the function. The first implementations of stretching involved 25 iterations, but today the number can be in the thousands. (Klein, 1990) The main advantage of using stretching is that it can help protect passwords from being guessed. By stretching the hashing algorithm, the authentication process is slower, but still within an acceptable range to the user while the time it takes to guess a password is greatly increased and becomes very inconvenient for the attacker.

One issue with stretching is that it will task the resources of the system it’s running on. Depending on the hashing function chosen, it may consume a lot of processor resources or a lot of memory. This would mean that a lower number of users can authenticate in a given period of time or more resources will have to be installed.

Table 2-4 shows the result of stretching the salted hashed password 5 times. The first iteration is just like the last example. From then on, the resulting digest is fed as input to the hashing function and run again. The same process is run until we get to the fifth iteration and then store that digest with the appended salt. If salts hadn’t been used, the same issues with repeating digests would occur for users that share passwords.

<table>
<thead>
<tr>
<th>User</th>
<th>Original Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>rose</td>
<td>bluehouse</td>
</tr>
<tr>
<td>mary</td>
<td>bluehouse</td>
</tr>
<tr>
<td>alvin</td>
<td>helloworld</td>
</tr>
<tr>
<td>jack</td>
<td>hellowormd</td>
</tr>
</tbody>
</table>

Only this table stored in database -->

<table>
<thead>
<tr>
<th>User</th>
<th>Stretched (5 times) Salted SHA-1 Hashed Password (Digest+Salt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rose</td>
<td>ee21504acbe68ea89b833288c4213b9ac4a9c056f75982</td>
</tr>
<tr>
<td>mary</td>
<td>32fc8105275ce3456b20d98bed0717e30abb589eb5070</td>
</tr>
<tr>
<td>alvin</td>
<td>af000ea5836578af3bfeb2104e092e5398af6da59333dc</td>
</tr>
<tr>
<td>jack</td>
<td>d4638ebba6eb6a1c67f3868303a7b18e63fd7c9c62</td>
</tr>
</tbody>
</table>

Salts have been bolded for easy identification

Table 2-4 Stretched Salted Hashed Password example

Hashed with: (FileFormat.Info)
2-3e Policies

Password policies are restrictions to the length and complexity with which the users can choose their passwords. A system administrator will implement a set of rules over what users can choose as their password. These rules can work to force the user to increase the length of the password to fulfill at least a set number of characters like a minimum of 8 for example. The policies can also force the user to include numbers, upper case letters or symbols in their password.

Every time the number of characters is increased or the type of characters used grows, the possible keyspace from which the password is part of increases. Table 2-5 shows the corresponding number of possible combinations based on the type of characters that are chosen for the password as well as the total number of characters in the password, also known as password length. Most users won’t use ASCII characters other than what is on their keyboard since they would need to know the code for that special character.

<table>
<thead>
<tr>
<th>Char. Set Size</th>
<th>Character Types</th>
<th>Password Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Digits</td>
<td>Letters</td>
</tr>
<tr>
<td>10 Decimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 Hexadecimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 Case-insensitive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 Decimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46 Case-insensitive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52 Upper and lower</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62 Decimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72 Upper and lower</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95 Decimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>222 Decimal</td>
<td>Upper and lower</td>
<td>All symbols on standard keyboard</td>
</tr>
</tbody>
</table>

Table 2-5 Possible Keyspaces by Password Length and Character Set Size

(Scarfone & Souppaya, NIST Special Publication 800-118 (Draft) Guide to Enterprise Password Management, 2009, p. 20)
In some cases, policies limit the maximum number of characters or the type of characters that can be used in a password. In these cases, not all characters can be used and only a maximum number of characters can be entered. Different implementations support the use of only a subset of the symbols and have a hard limit on the maximum number of characters. There can be many reasons why policies limit the maximum characters and different symbols. Sometimes it can depend on the legacy application and its ability to work with older systems, while other times, it’s just not implemented as well as it could be out of ignorance from the administrator.

Not all policy restrictions help to make the passwords harder to guess. Entropy is used to measure the randomness in the system. (Scarfone & Souppaya, NIST Special Publication 800-118 (Draft) Guide to Enterprise Password Management, 2009) In Figure 2-4 we can see that the password length is the parameter that impacts the keyspace the most, well over the impact that can be attributed to including all symbols or ascii characters. When users are faced with more restrictions on the amount of special characters (not letters) that they can use, they tend to fall back on predictable patterns to construct their passwords. The result is that the entropy of the password is negatively affected. It would be better to encourage the user to use a longer password than to make sure they include at least one of every type of character. (Komanduri, et al., 2011)

Close attention must be paid to the policies that are put in place because their main advantage of helping users choose strong passwords can turn to a disadvantage if it’s not implemented properly.
2-4 Layers of security

The password database is not the only layer of security in authentication systems. Every implementation is slightly different, but all of them share layers of security. The whole system is a group of components that work together to provide services to the users. The database has its authentication mechanisms to allow or deny access to the data. The portal through which users log in has to process requests and at some point compare the supplied credentials with what is stored in the database. The operating system in each of the servers involved plays a critical role in running the applications and preventing unauthorized access. The network enables communication between the interfaces of all the components and can also be configured to monitor or block certain communications.

All the layers in the system must be designed to prevent any unauthorized use at any level. For every layer, there are multiple ways of achieving the desired operational result. However, not all of those options prevent users from engaging in activities not intended by the system design. A proactive approach must be used to design the whole system to minimize...
threats. The NIST Guide to General Server Security (Scarfone, Jansen, & Tracy, NIST Special Publication 800-123 Guide to General Server Security, 2008) includes in chapter 2-4 a good list of information security principles that can be applied here. Some of these guidelines are summarized here; the whole list is included in Appendix A.

When implementing the system, the design and configuration may be tightly guarded secrets, however, it’s important not to confuse real security with “security by obscurity”. In the latter, the system relies on external parties not knowing how the system works as a measure of security. This will work until the secrets are discovered. Only by making sure that each component provides appropriate security even when the attacker has complete knowledge of its functionality can we have a secure system.

Complicating security algorithms can make the system harder to operate, diagnose and improve over time. Simple elegant solutions provide better overall security than solutions that complicate human understanding of the security but represent no greater challenge to computers.

2-5 Other methods

With the increased risk of password compromise, many other approaches have been developed and researched to try to provide greater security. Some methods include additional items needed to authenticate, redesigns of the password storage system, extra layers of authentication, among others. Following is a description of some of these methods, but there are many others that can seem weird, strange or novel. There have been proposals about systems where you should pick the color of your password, others where you trust your friends to authenticate you or where you must recognize your friends to get access. We will focus on the ones that are more relevant to this study because of their similarity to my research or for their broad utilization. Each one has its benefits and its weaknesses, and analyzing many of them can help understand how a more robust system can use a combination of approaches.

2-5a Two-factor and Multi-factor authentication

Multifactor authentication means that the user will need more than one factor to authenticate. There are three categories of authentication factors: something we know,
something we have or something we are. (Burr, et al., 2011, p. 20) In recent times we have also seen the rise in the idea there could be a fourth type of factor: something we do. (Peslyak & Marechal, 2012, p. 63) Normally the password is the first factor of authentication that most of us are used to. The password represents a secret that we know, and like this there are many other things that fall into this category. For example the questions we get asked when we forget a password: “What was your first pet’s name?”, “Where did you grow up?”, etc.

A typical two factor authentication that not many people realize they use is an ATM card and its PIN. To get access to your account, you need to insert the card (something you have) and input the PIN (something you know). A more sophisticated two-factor mechanism is the use of tokens. These may come in many shapes and sizes as seen in the sample of RSA SecurID products Figure 2-5.

Tokens like these are “something you have” and they work by generating a sequence of numbers on a regular time interval. This means that the number on the device changes every minute and the server expects the user to enter the appropriate code at the time they are logging-in. Some devices offer additional level of protection like the need to input a PIN or
scanning the fingerprint of the user (therefore using two types of factors) to get the code. The
tokens can also be software based. In this case they run on your PC or on a smartphone. Other
implementations don’t generate the codes on the device, but transmit it through another
means like a cell phone network also called “out of band”. This still relies on the user having the
device, in some cases being as simple as an SMS message, in others authentication can just
involve replying or accepting the message.

The third type of factor is who you are and it was mentioned in passing as the
fingerprint, but it may encompass any number of biometric readings like voice recognition, iris
scanning, facial recognition, etc. The fourth type is what you do and is based on behavior, which
can mean the schedule you usually log-in to a service, in the internet can mean the IP address
or location you are logging from.

There are definitely benefits to increasing the level of confidence that the user is who
they say they are. When properly used, multi-factor authentication helps foil attempts by
attackers that have gotten a user’s password. The drawback of these systems is that depending
on what the other mechanism of authentication is, it can be inconvenient for the user because
they have to carry something around with them. In other cases if the “out of band” system is
not secure enough, the second-factor authentication may also be compromised. An example of
this could be a man-in-the-middle attack or a Trojan in the system. (Schneier, 2005)

2-5b Threshold Cryptography

Current authentication systems like the ones we have been looking at throughout this
paper are based on one database containing all the data necessary to authenticate a user.
Another approach is to split the data between servers so that authentication requires multiple
parts of a key to get access. (Simonite, 2012) The user does not need to change anything, as
far as they are concerned, they are inputting their username and password the same way they
have always done. Behind the curtain, in the datacenter, the credentials are treated differently.
This system uses complex math so that the data in only one of the servers is useless, but when
enough servers cooperate, they can authenticate the password.

RSA has built a system using this technology. In its Distributed Credential Protection
(EMC Corporation, 2012), they use two servers where they store two values. One of the servers
holds the password masked with extra data. Another server holds a copy of that extra data. By themselves, each server holds gibberish that is useless to figure out the password. You need both parts to be able to decipher it.

When it's time to authenticate, the user inputs its password and it is masked with the same process as before, but with a new random set of data. Both servers then subtract their input from what they had stored. If the password entered right now is equal to the one that was originally used, the result from the subtraction on both servers will be the same and the user is allowed access. Figure 2-6 shows a diagram and a brief explanation of this process.

Figure 2-6 RSA Distributed Credential Protection

Source: (EMC Corporation, 2012)

If an attacker can get hold of one of the pieces of data, he needs to get the other one to be able to figure out the passwords. It would just be a matter of time before he could get it. To minimize this issue, the servers periodically generate additional data that is added to what is already stored. This changes what is inside so that an old version of the data won't work with the current data. The active servers will need to keep synchronized and always change their data together. By doing this, attackers would be forced to get a hold of both parts of the data before the next window of changes happens.
Threshold cryptography might be very beneficial if the two servers can be kept secured so that an attacker can’t get access to both in the timeframe needed before the next update of the data. A very important aspect of the whole system will be the need for fast mechanisms to detect whenever a breach has happened and therefore trigger the data change.

**2-5c Negative Authentication**

Up to this point, all the authentication mechanisms are based on storing the credentials of the user and if they match what is being entered to log-in, they are granted access. This is called positive authentication, because it positively identifies the user. Negative authentication takes a different approach to making sure the user is who he says he is. Negative authentication as proposed by Dr. Dasgupta (Dasgupta & Azeem, An Investigation of Negative Authentication Systems, 2007) takes the credentials that are sent by the user trying to log in and compares them to everything that it knows is not a valid username and password combination.

If the negative authentication implementation has everything that is not a valid username and password, then anyone could invert the information and find everything that is a valid username and password. They would essentially have the same thing as the positive authentication database. To fix this, negative authentication leaves holes in the data of negative authentication which translate to false sets of username and password data.

Now a valid user with the right credentials will go through negative authentication and the system will let him through. Additionally, there will be a number of combinations of username and password that will falsely pass through the negative authentication. For this reason, negative authentication is normally thought of as a two layer solution, where the negative database is the first layer of security and a positive database will be the second layer. After users pass the first layer, they still have to pass the positive authentication to be 100% sure they are who they say they are. Figure 2-7 shows a simple diagram of how the layers interact with each other and complete the full authentication. Security layer one is the Negative Authentication layer and includes the Anti-P detectors that are the areas that are not valid user password combinations. The hole in the middle is everything that that is a valid password. In a real life implementation the spread of the valid credentials would mean that they would be really small holes scattered in the available space.
There are benefits to having this two layer approach. First, it helps in the detection of attacks trying to guess passwords. Most of those tries will fail the negative authentication and the system can send alarms or signal security systems to take measures to block off the attack.

Another benefit is that there are a lot of possible combinations of username and passwords. The number is so large, that even with the holes in the negative data, the probability of finding one by chance is extremely small. This way, the requests that pass the first layer are very few and very likely to be correct. This acts as a barrier of protection so that the positive authentication system is not attacked as often.

The main weakness in negative authentication as it has been proposed up to now is that it still requires a positive authentication server to be sure of the user’s identity and to be able to handle new users or changes in their credentials. If an attacker gets a hold of the positive authentication database, he has enough data to try to guess the passwords.
Chapter 3 What is happening in the real world?

3-1 Security Reports

The past few years have been eventful in data breaches where password information has been compromised. In here I present relevant findings in several security reports by done by major security labs. All of these reports gather information from multiple sources and then sort and quantify real world issues that happened during the period of the report. All of them apply to all or part of 2012 and are the most recent reports available from each of the organizations.

The main findings reflect some big issues with password security and the repercussions that this has on further attacks. Companies are disclosing more incidents where data was stolen. The misuse of passwords ranks very high as the main vector that hackers use to gain access to a system. From there the next step may be financial (trying to acquire illicit funds) or gathering data (including sets of passwords that may or may not be hashed.) To get the original passwords to access the system, perpetrators leverage password reuse where employees have their passwords compromised elsewhere and because they use the same password for work, the attacker can log-in with their privileges. Another popular method is with malware designed to steal passwords from the user’s computer. Once the user’s computer is infected with malware, it can capture information on that system and relay it outwards. Even if the original code for the malware didn’t have that function, it usually contacts outside servers to get more commands and install other malicious software.

According to Verizon’s 2013 Data Breach report one of the main causes of data breaches was passwords that were too easy to guess or stolen passwords. “The easiest and least-detectable way to gain unauthorized access is to leverage someone’s (or something’s) authorized access... [authentication based attacks] factored into about four of every five breaches”. (Verizon RISK Team, 2013, p. 34) We see that in Figure 3-1 it is the largest cause contributing to breaches and then focused only on incidents that involved hacking in Figure 3-2. In this figure we can see that the percentage attributed to unsafe credentials either stolen or susceptible to brute force, reaches over 80%.
How do breaches occur?

- **52%** used some form of hacking
- **76%** of network intrusions exploited weak or stolen credentials
- **40%** incorporated malware
- **35%** involved physical attacks
- **29%** leveraged social tactics
- **13%** resulted from privilege misuse and abuse

The one-two combo of hacking and malware struck less often this round, but definitely isn’t down for the count. Filtering out the large number of physical ATM skimming incidents shows exploitation of weak and stolen credentials still standing in the ring.

The proportion of breaches incorporating social tactics like phishing was four times higher in 2012. Credit the rise of this challenger to its widespread use in targeted espionage campaigns.

Correlated with the 1.4% of breaches tied to insiders, privilege misuse weighs in at 13%. Insider actions ranged from simple card skimming to far more complicated plots to smuggle corporate IP to competitors.

---

**Figure 3-1 How data breaches occur**

*Source: (Verizon RISK Team, 2013, p. 6)*

**Figure 3-2 Variety of hacking actions in data breaches**

*Source: (Verizon RISK Team, 2013, p. 34)*

In at least one incident pertaining to Dropbox, hackers took an obtained password from an employee and used it to get access to that employee’s account. From there they got access to Dropbox user’s account information. (Check Point Software Technologies LTD, 2013, p. 27)

One of the methods used to obtain the initial illicit credentials is malware. According to
McAfee, as shown in Figure 3-3 there has been an almost continuous rise in the number of detected malware samples that steal passwords.

![Figure 3-3 General Malware Threats: New Password Stealers Samples](image)

*McAfee Labs, 2012, p. 12*

The other main source of unauthorized credentials has to do with social hacking where the user is tricked into revealing their password. By far the most common form of social action is Phishing to which most reports agree and Verizon puts in at 77% in Figure 3-4. This is especially prevalent in breaches involving espionage type activities.

![Figure 25: Variety of social actions](image)

*Source: (Verizon RISK Team, 2013, p. 35)*

Another trend that is alarming is that the number of incidents that are made public is rising. According to McAfee in Figure 3-5 we can see a constant increase after 2009. It is unclear how many events remain secret and if these are the companies being more outward facing and
transparent, if there is a rise in the amount of incidents happening or if the companies are finding out about them with more frequency. In all likelihood, it’s a combination of all three.

![Figure 3-5 Data Breaches Made Public](source: McAfee Labs, 2012)

In the events that have been publicly shared, the most sought after piece of data are the ones that are most easily turned into profits. This turns out to be payment transactions or credentials into financial systems as shown in Figure 3-6. The attackers then use these to withdraw funds from banks or individuals and funnel them into their own accounts. In the rest of the events, the amount of credentials stolen is still amongst the top most stolen data.

![Figure 3-6: Variety of compromised data](source: Verizon RISK Team, 2013, p. 46)
A big warning to everyone in the industry is the amount of time it takes for the victim to realize that they have been compromised. In Verizon’s report Figure 3-7 the timespan shows how attackers normally require very little time to find and extract what they need, but companies usually take months to realize what happened, and in sometimes even years. After that, most breaches are contained within days, but some can take months.

There is no single point of failure in the system that is causing all the problems. But the issues affect almost all its parts. “All in all, 2012 reminded us that breaches are a multi-faceted
problem, and any one-dimensional attempt to describe them fails to adequately capture their complexity.” (Verizon RISK Team, 2013, p. 4)

3-2 Policies, Regulations and Guidelines

Laws and guidelines from authorities that dictate minimum security requirements are not easy to find. There is no organization that rules over how a service provider handles the security of the passwords for its users. Organizations like NIST offer a set of recommendations on how to implement several systems, but they are not very detailed when it comes to passwords. Their primary document addressing password security is still in draft form since 2009. (Scarfone & Souppaya, NIST Special Publication 800-118 (Draft) Guide to Enterprise Password Management, 2009) In it, they describe the main attacks against password security and how to mitigate them. The recommendations reflect the commonly accepted best practices of using a good hashing algorithm, salting and stretching.

Other organizations offer similar documents with varying amount of detail into each topic. ENISA offers a short guide for users and system administrators pointing out the main issues and why it’s so urgent that they are addressed. (ENISA, 2012) However, there are no regulations that force anyone to implement password security the right way.

Another role of these institutes is to help establish standards. For years, NIST has worked to establish standards on hashing algorithms. It defined the SHA family and has been updating it whenever there is a reasonable indication that the current ones are no longer enough. The contest to define SHA-3 took place after attacks were discovered against SHA-1. “In 2004-2005, several cryptographic hash algorithms were successfully attacked, and serious attacks were published against the NIST-approved SHA-1.” (NIST, 2013) Even though SHA-2 remains secure, the new standard will provide added security measures.

On top of the standards and guidelines, there are sometimes politics involved and there’s a lot of confusion around password security and how it should be done. This has caused misunderstandings and oversights from policy makers and their constituents. A case like this happened in France where a law passed obliges sites to provide user data upon request, including passwords. (BBC, 2011) At the time, several sites stated that the law required sites to hand over unhashed or plaintext passwords, forcing them to go back on security policies that
kept them secure. Large companies backed organizations that challenged the law. In reality they were only required to store the passwords for a period of two years in the way they were currently stored and not go back on implemented policies. The confusion between all the stakeholders shows that there is a lack of understanding on how these password security systems should be implemented and ultimately results in unclear or misunderstood laws and insufficient guidelines.

3-3 Cases where password databases have been leaked

The past few years seem like cyber-attacks happen every other week. In the summer of 2012 ENISA published that just in a couple of months ~18.4 Million passwords had been stolen. (ENISA, 2012) Many big organizations were hacked and some events surrounding the breach made it public. In some cases the attackers were the ones that publicized the breach, in others the company suffered some discontinuity in its service, and in the rest the company decided to contact its users to inform them of the event and to recommend the actions they should take to protect their information.

One of such breaches was the very public release of 6.5 Million passwords from LinkedIn on June 6 2012. Most of the passwords released were still hashed, but some of the easy passwords had been already discovered. (Silveira, An Update on LinkedIn Member Passwords Compromised, 2012) Immediately after the events, LinkedIn disabled accounts of the compromised users and recommended everyone change their password frequently. This would allow them to take advantage of the newly implemented salting of their password database. The salt would help LinkedIn attain an “extra layer of protection that is a widely recognized best practice within the industry”. (Silveira, Taking Steps To Protect Our Members, 2012) The lack of salts in the password file made it much easier to crack them, and sometime later close to 90% of the passwords had already been found.

In 2011 Sony faced attacks to its PlayStation Network and to Sony Pictures. During April of that year, Sony took down the PlayStation network in response to attacks and latter recognized that records of 77 million user including passwords and probably credit card information were taken. (Roberts, 2011) While they didn’t get into much detail of the technical implementation, Sony informed the public that the passwords were hashed. (Seybold, 2011)
For many days service to PlayStation network was down to prevent more breaches and to improve security. However, the same policies were not implemented across the whole company. Two months later, in June, hacker group LulzSec, using SQL injection vulnerabilities got access to around 1 million passwords from Sony Pictures that were stored in plaintext. (Vijayan, 2011)

On March 2nd 2013 Evernote wrote to 50 million users to tell them their passwords need to be reset after a break-in where attackers stole the credentials database. (Lunden, 2013) Fortunately, they seem to have taken at least some measure of security, and although details were not released, they did say that the passwords were hashed and salted. (The Evernote team, 2013) This means that they would be difficult to decipher, but not impossible. If users change their passwords right away (assuming they choose a different password), then the risk of attackers gaining access to the Evernote accounts is minimal. However, as in anyone of this type of attacks, the database is already out there in the open and it’s just a matter of time before most if not all the passwords are found. If users reused the same password elsewhere, they need to change it too. One of Evernote’s recommendations is that, as always, passwords should not be the same in different websites.

This is not meant to be an exhaustive list of the incidents that have happened, but a sample of the type of events and what different organizations have done before and after. In some cases, companies are well prepared and have implemented all the security policies therefore buying more time for their users to change their passwords and take appropriate measures to prevent further consequences. In the majority of cases, there has been a clear lack of at least one of the suggested components from the best practices. This puts the users at more risk of their information being revealed soon.

3-4 Other advances in password cracking

The speed with which password hashes can be cracked is a constant struggle between new techniques with increased computing power and the complexity of the implemented hashing algorithms. In all cases, it’s just a matter of time before the right password is found to match the stored digest. Since most current hashing algorithms were developed to be fast and light, they are good candidates to run on GPUs. A GPU or Graphic Processing Unit was originally
designed to perform 3D calculations normally used in games or simulations. Over time these processors have become more complex and the number of cores per processor has increased dramatically. While regular CPUs inside the computer have somewhere between 2 and 8 cores, the GPUs have hundreds of simpler cores. Each one is capable of processing in parallel of its companions. This allows GPUs to be well suited for brute force attacks against password hashes.

One of the latest publicized exercises of using GPUs to crack password hashes utilized several GPUs per server and then interconnected servers to each other via software virtualizations. Using Virtual OpenCL, Jeremi Gosney linked 25 GPUs and managed to achieve 348 Billion hashes per second using NTLM hashing algorithm. MD5 and SHA1 were also very fast in the tens of billions of hashes per second. The performance depended on the characteristics algorithm used, and newer algorithms that consume a lot of memory like bcrypt did not benefit as much from this approach, achieving just 71 thousand hashes per second. (Gosney, 2012)

The tremendous power available to crack hashed passwords, using GPUs is no longer just available to very experienced hackers that have resources to buy the hardware required. Taking advantage of cloud computing resources allows the hacker to rent computational power by the hour from services like EC2, paying just a few dollars per hour. (Roth, 2011) Now the hacker can work from anywhere and pay for the instances just when required.

The improvement of the hacker’s toolbox will be an everlasting problem. As long as new technologies and research is being done, there will be shortcuts and speed gains to break those passwords faster. For every hashing algorithm or its variants, there will come a time when it is no longer recommended for use. That is why every few years, algorithms need to be changed. However, there might be a point where just swapping algorithm won’t be enough and we need to start thinking now about alternative designs for the whole system.
Chapter 4 Negative authentication

Negative authentication system also abbreviated NAS, refers to a system of authentication that compares the submitted credentials against a database of values that are known to be invalid combinations of username and password. The total space of valid username and password combinations is very large, and it depends on the method of storage for each particular implementation. For the implementations that we have been working on, the combination of usernames and passwords are stored using 256 bit hashes which means there are $2^{256}$ possible combinations. Every pair of username and password will map onto one point on the negative space, and the hashing algorithms produce a uniform distribution. Beyond this, any set of credentials that’s sufficiently large could produce hash collisions.

Figure 4-1 represents the possible space of username password combinations and in a) shows a set of 25 pairs of username and password already mapped onto the space, while on b) the resulting negative space. Notice the negative space is not the exact opposite of the positive, but still every valid pair of credentials maps to a hole.

Because the difference between the total possible combinations and the valid set of usernames and passwords is big enough, we can add some uncertainty to the data of negative detectors. The number of valid username and password combinations depends on the amount of users. A system with 10,000 users will have less than $2^{10}$ combinations which would leave
2^256-2^10=2^246 values covered by the negative detectors. In a system with 1,000,000 users we would still have a detector space of at least 2^236. With such a large space, we can have several times the amount of valid credentials as holes or ambiguity in the detectors (also called the confusion parameter). This serves the purpose of hiding the positive database by adding noise to the inverse of the data.

4-1 Our approach to Negative Authentication

The implementations that we have worked on stemmed from Ji’s and Dasgupta’s papers and research. (Ji & Dasgupta, 2004) The basis of the system is the sets of valid credentials that are stored in the positive authentication database. From there, the user and password for each account are combined and then hashed and the resulting digest becomes the point in the space that must remain as holes, also named the self-points. One of the advantages of negative authentication is that for it to work hashing the password is mandatory. This will guarantee that at least that level of security is implemented in the system.

Depending on the particular implementation algorithm, the digests go through a varying transformation processes so that they become coordinates in the negative authentication space. Each implementation will have different characteristics for the space and it can be represented in many dimensions. For every dimension, there needs to be a value, therefore the digest is split and each part is assigned to a dimension. Our tests have focused on 2, 4 and N dimensions where N represents the number of bits in the digest. However, the same theory can be applied to any number of dimensions, even 1 dimension.

4-2 Different implementations

Structuring data across two dimensions allows for a visually clearer representation of the negative space. This way it can be shown in flat diagrams presented in this thesis. Details on each method of implementation are explained in the next sections. All of these implementations work as Negative Authentication layers that rely on a second layer of positive authentication to authenticate a user with 100% accuracy.

Three parameters affect the accuracy of negative authentication: the area covered by the detectors, the area of the self-points and the resolution of the self-points vs. the actual
digest. If the area of the detectors is very close to covering everything but the coordinates resulting from the digest, then it will be easier to inverse the data to get the positive authentication database. If the area of the detectors is small, then it will be very hard to determine the positive database, but the certainty that the users are using the correct credentials to log in is diminished. Each of the implementations offers a varying degree of certainty that can be adjusted by tweaking the aforementioned parameters.

To authenticate users through the negative authentication layer, the supplied credentials are processed in the same manner that was used to generate the self-points for each method. Then the resulting coordinates are checked against the detectors. If there is no match, it means that layer 1 is passed and there is a level of certainty that the credentials are valid. The authentication is then passed to layer 2. In the opposite case, where the point matches a detector, the layer 1 authentication fails and it's known that the supplied credentials are wrong. This allows the first layer to stop invalid requests from reaching layer 2 and also provides a way of triggering advanced warnings if there are many failed attempts to log-in.

Since negative authentication is still a new design to implement authentication, there is no dominant architecture of how it is built or how it works. The following are examples of approaches that we have been testing so far.

4-2a Real Space

The Real Space implementation uses hyperspheres in 4 dimensions to represent the areas that are not valid usernames and passwords. These hyperspheres are the detectors and are defined by a point in the 4D space which is the center of the sphere and a radius that is the distance from the center which the detector covers.

To generate the detectors, the process starts with the set of valid usernames and passwords and from there gets the self-points. Every self-point is composed of four segments of equal length. The digest is split into four equal segments that map to the 4 dimensions. From each segment a piece is discarded to add the uncertainty or ambiguity. This is the same thing as lowering the resolution of the space. Each of the self-points then represents a point in a space that is smaller than $2^{256}$. 
The next step is to generate the detectors. The space between the self-point is filled with hyperspheres in random coordinates with radii that do not overlap with any self-point. The process continues until a certain coverage is achieved. Then the self-point information is discarded and only the detectors remain as a set of coordinates and radii. Figure 4-2 shows the steps in the creation of the detectors. To be able to present it clearly, the diagram shows only 2 dimensions, while the Real Space implementation uses 4. Step a) is the representation of the self-points in the available space. Each one is marked by a point enclosed in a small box. The box represents the minimum resolution that the implementation allows depending on the amount of data that's discarded from the split digest. In step b) the self-points are shown with the hypersphere detectors and an approximation of the area covered by them in light gray. None of the detectors must cover any self-point. Step c) shows just the detectors remaining as hyperspheres. This resulting database of detectors will be used to authenticate the users in the Negative Authentication level.

Real Space implementation provides a coordinate space that is complicated to understand at times because it has to be visualized in 4 dimensions. However, computers have no trouble in handling the added complexity. The distance between the points is calculated by getting the Euclidean distance which is the square root of the sum of the distances between the two points squared. This is very similar to calculating the hypotenuse of a triangle, but in 4 dimensions.
One issue with the current implementation is that it takes a long time to generate random points and check them against all the self-points and previous detectors before creating a new detector. As the number of detectors increases, so does the time to check them all.

4-2b Binary

The Binary implementation is similar to Real Space in that for the detectors it uses the concept of hyperspheres with the coordinates of its center and a radius. However this implementation goes to N dimensions which is the same as using each bit in the digest as a dimension. If we were using a hashing algorithm of 256 bits, the dimension is 256. This allows a simplification on calculating the distance between two points. Since every bit is a dimension, the maximum distance or difference there can be in any dimension between two points is 1. When both points share the same value for a bit 0,0 or 1,1 there is no difference. When they have different values 1,0 or 0,1, the distance is 1. This is called the Hamming Distance. Figure 4-3 shows the calculation for two strings. The binary boxes show every bit of the first string stacked on top of the second string bits. The third row shows a 1 in every column where there's a difference in the strings. When all the 1's in the third row are added, the result is 13 which means we obtain a Hamming Distance of 13. Below the strings is the hexadecimal representation of each which is easier to read than long strings of binary digits.

```
0010010010011011101100110101101010101101010001100011
00101011001000110000110001010001000000000011000001110
```

Hamming Distance: 13

In Binary space, we don't discard some of the data when generating self-points. Each self-point uses all the bits in the digest. To add the ambiguity or confusion to the set of detectors, the radii of the self-points is increased. The random creation of detectors and their
radii makes the self-points very hard to get from only the negative authentication data. The binary implementation shares the issue of slow creation of detectors as the number of these increases because they have to be checked to every previous one created. Grid based

4-2c Grid based

Originally suggested by John Williams, the Grid based implementation splits the available space into a grid. (Williams, 2013) The detectors are the cells that do not contain self-points. Since the cells are part of the total area, this is the same as the reduction of resolution and automatically provides ambiguity proportional to the size of the grid cell chosen. The self-points only need to map to a grid cell, all the extra data that determines in which position in the cell they are, is discarded. Additional ambiguity can be added by selecting random cells and treating them as if they were also self-points. However, because of the normal distribution of hashing algorithms, choosing random fake cells or increasing the size of the cells produces the same effect. Each time we double the amount of fake cells we could alternatively increase the dimension of the cells by one bit to obtain the same level of ambiguity.

Using a grid pattern in two dimensions allows a much simpler method of finding all the space that is not part of the self-point cells. Therefore Grid is the fastest method we have tested of creating the detectors. For our implementation, the detectors are rectangles that are one cell high and cover the horizontal area from the beginning of the row to the first cell with a self-point or from the previous cell with a self-point to the next one. Figure 4-4 shows the steps in the process. In step a) the self-points are mapped to cells. Step b) introduces random fake cells and then fills up all the remaining space with detectors shown as light gray rectangles. Step c) shows the resulting set of detectors where it's impossible to distinguish the real from the fake self-points. Additionally every cell that is left blank represents the equivalent of cell-size worth of possible digests that map onto this space. Only one digest is needed to mark a whole cell as self-point, but each cell can contain millions or more possible digests.
The main advantages of grid space are that it is easy to understand and implement, it is also very fast to create the detectors and therefore can work in almost real-time when there is a user creation or password change.

4-3 Server Environment and Layers

The different implementation methods describe the behavior of the detectors and the database. A very important part of the system is how it's organized to take advantage of the negative authentication layer. The system is deployed in two layers with different security levels. The first layer is the one that faces the users and is the negative authentication layer. A server receives requests from users to authenticate and does the initial verification. The second layer contains the positive authentication server that will make sure the user credentials match 100%. In the same security level of layer two, there's an application server in charge of generating the list of detectors and then sending them to the layer one server.

By implementing this layered approach, the communication between the layers may be inspected and reinforced. With the added warning mechanism that negative authentication provides, the whole system can be better equipped to detect irregular behavior that may be a sign of an attack. Knowledge of an attack in progress can help the administrators to deploy countermeasures earlier than without a warning system. This could lead to less instances of compromising the user's passwords. It is important to highlight that the security of the system will depend on its weakest link, and if an attacker were to get access to the layer 2 positive

Figure 4-4 Negative Authentication: Grid - Detector Creation
authentication database, it would be no different as it was a traditional one layer positive authentication system.

NAS has some distinct characteristics like pre-screening requests and getting 99.9% or higher certainty that the user is who they say they are, but without having a database that can be reversed to reveal the user’s passwords. Current implementations still have room to grow and mature to a point where systems like this may become more valuable.
Chapter 5 Results

So far the research has shown that password security is still a grave issue. There is no indication that the overall safety of the user’s credentials is getting better, in fact, as time goes by and computing power increases or becomes cheaper, the situation worsens. The techniques used to optimize password guessing are making brute force attacks much more efficient. Where previously thought that the amount of work needed to crack a password would take years, now takes seconds or minutes.

The sophistication of the tools available to hackers is increasing in every generation; meanwhile the system architecture of a password security system hasn’t changed in decades. All of the security enhancements done to password security have been in increasing the length of digests and the time it takes to calculate them. The next generation hashing algorithms are starting to become more common and when well implemented will help keep passwords hidden for another generation. The adaptable stretching of the algorithms may help extend their life, but it requires attention to keep the iteration numbers high enough to represent a decent computational challenge. For this to work, they need to be ubiquitously implemented.

Even with a password system design that has been around for many years, there are constantly public disclosures of large corporations where the best practices were not followed. Current policies do not enforce many of these best practices, but instead just offer them as guidelines or recommendations. The big software vendors are not doing a good enough job to make sure they keep up to date with password security implementations and promote them with their customers. For many system administrators this represents a confusing landscape and creating a lack of knowledge on how to implement their system.

New system designs like negative authentication and others, offer only slightly better security than the systems that are implemented with the current design to the best of its capabilities. However, the new approaches force the use of hashes and can take the opportunity to define their system as also requiring salting and stretching. This clean slate approach may be the largest short-run benefit to the overall system security.

Negative authentication by itself in the implementations we have seen so far is not going to solve the big problems with password security that we have now. Going forward, it
certainly provides an interesting vector to search for new approaches to keeping passwords safe. The change in design of password security systems can benefit from the advances we have made up until now by using the same components (hashing, salting, stretching) but changing the way they work with each other and how users are validated. Other components like multifactor authentication can be more integrally tied into the system to add versatility and thoroughness. A new security system design that proves to be safer than the current one will allow us to step away from just making more complex hashing algorithms.

Although we can keep increasing the computational expense of calculating hashes almost without limit, the human aspect of choosing passwords does have its limits. Every day there are more services online that require passwords, and as password become longer, they are harder to remember. Users are now also forced to follow character policies that may not have been well thought out. The combination of all these factors makes password reuse more tempting and in many cases users rely on weak passwords. We have seen how one compromised password that is reused between personal sites and work access, can lead into a serious security breach.
Chapter 6 Conclusions

6-1 Recommendations

Institutional measures need to be taken to ensure systems administrators implement defenses for all known exploits. Currently, the gap between solution and vulnerability is too large. It is important to recognize security requires constant vigilance. Security vulnerabilities, like technology, are constantly changing. Security needs to be engaged in a continuous cycle of self-improvement.

Unfortunately, as has been discussed, many organizations fail to implement even the most basic measures. Today, there should be no password security system that does not take advantage of hashing, salting and stretching to protect its passwords.

Companies and users alike should push software vendors to provide easier to implement security in their systems. Most administrators will have enough work to worry about and they want their ROI to come as soon as possible. Vendors and providers should make their software easy to use and offer a clear way of managing password security implementations and updates to it.

Users should hold companies accountable for data breaches, and push for more stringent laws and regulations. They should also insist that companies implement security measures that conform to current best practices. In this age of social networks, if online service providers feel the pressure from its users, they'll be more inclined to implement secure policies, even when it's difficult to verify that they are actually doing it.

6-2 Implications

Password security has been a concern for many years, but the mechanisms to protect them have not changed very much. The techniques used to secure password databases have been around since the 70's. Through time we have seen the complexity of the algorithms increase to cope with advances in hardware speed and optimizations in the approaches used to calculate hashes. The fact remains in that in the best implemented cases we are still using almost the same security that was designed over 35 years ago.
Many companies are not implementing best practices in password security. Even with such old concepts we find that large corporations have trouble implementing all security measures correctly. Part of the corporation may have recently faced a cyber-attack, but that doesn’t immediately cause all the organization to revise their systems and fix them right away.

Other than ignorance and confusion, there are no reasons why large systems can’t implement best practices in keeping their passwords secure. When users start pushing companies for accountability and regulators to police the environment we’ll finally see changes that impact their bottom line and consequently drive improvements.

As with any complex system, especially where humans are involved, all developments need to take into account all the components and realize that the weakest one might compromise the whole system.

### 6.3 Remaining questions and future work

There are still many approaches that can be taken to design new systems to handle password security. In particular negative authentication could provide a solution that is good enough to authenticate users and does not provide the data needed to trace back to the original passwords. Two approaches for future research come to mind. The first: use only Negative Authentication without Positive Authentication, and the second to link the two layers of negative and positive in a way that obtaining only one of the parts is not enough to work out the passwords.

- Where negative authentication becomes the only layer in which passwords are involved, this would not guarantee 100% accuracy, but it would be very close. Augmented with other systems in multifactor authentication can provide the added certainty required for a functional deployment.

- When linking both layers, the system should need something obtained from successfully traversing layer 1 for it to be able to process layer 2. Currently the easiest component to manipulate in such a way may be the salt.

Apart from the required technical aspects, a more detailed study into why system administrators don’t know the appropriate security measures may help determine the root causes. From there the best course of action to instruct them in the future can be developed. In
the hope that there may come a time when everyone involved understands enough to make a safe system.
References


Williams, J. R. (2013). Professor of Information Engineering, Department of Civil and Environmental Engineering and Engineering Systems Division at Massachusetts Institute of Technology.

52
Appendix A: General Information Security Principles

Extracted from: (Scarfone, Jansen, & Tracy, NIST Special Publication 800-123 Guide to General Server Security, 2008, pp. 2-4)

2.4 Server Security Principles

When addressing server security issues, it is an excellent idea to keep in mind the following general information security principles:

Simplicity—Security mechanisms (and information systems in general) should be as simple as possible. Complexity is at the root of many security issues.

Fail-Safe—If a failure occurs, the system should fail in a secure manner, i.e., security controls and settings remain in effect and are enforced. It is usually better to lose functionality rather than security.

Complete Mediation—Rather than providing direct access to information, mediators that enforce access policy should be employed. Common examples of mediators include file system permissions, proxies, firewalls, and mail gateways.

Open Design—System security should not depend on the secrecy of the implementation or its components.

Separation of Privilege—Functions, to the degree possible, should be separate and provide as much granularity as possible. The concept can apply to both systems and operators and users. In the case of systems, functions such as read, edit, write, and execute should be separate. In the case of system operators and users, roles should be as separate as possible. For example, if resources allow, the role of system administrator should be separate from that of the database administrator.

Least Privilege—This principle dictates that each task, process, or user is granted the minimum rights required to perform its job. By applying this principle consistently, if a task, process, or user is compromised, the scope of damage is constrained to the limited resources available to the compromised entity.

Psychological Acceptability—Users should understand the necessity of security. This can be provided through training and education. In addition, the security mechanisms in place should present users with sensible options that give them the usability they require on a daily basis. If users find the security mechanisms too cumbersome, they may devise ways to work around or compromise them. The objective is not to weaken security so it is understandable and acceptable, but to train and educate users and to design security mechanisms and policies that are usable and effective.

Least Common Mechanism—When providing a feature for the system, it is best to have a single process or service gain some function without granting that same function to other parts of the system. The ability for the Web server process to access a back-end database, for instance, should not also enable other applications on the system to access the back-end database.

Defense-in-Depth—Organizations should understand that a single security mechanism is generally insufficient. Security mechanisms (defenses) need to be layered so that compromise of a single
security mechanism is insufficient to compromise a host or network. No “silver bullet” exists for information system security.

Work Factor—Organizations should understand what it would take to break the system or network’s security features. The amount of work necessary for an attacker to break the system or network should exceed the value that the attacker would gain from a successful compromise.

Compromise Recording—Records and logs should be maintained so that if a compromise does occur, evidence of the attack is available to the organization. This information can assist in securing the network and host after the compromise and aid in identifying the methods and exploits used by the attacker. This information can be used to better secure the host or network in the future. In addition, these records and logs can assist organizations in identifying and prosecuting attackers.
Appendix B: Tips to end users

Extracted from: (ENISA, 2012, p. 5), except where noted.

- Test the sites you use the most. Use the “forgot password” link. If they send you back your password, then they are not storing it properly. Make sure to express your concern to them. (From author)
- Do not reuse the same password for multiple accounts. Attackers often try to re-use compromised passwords to access other services.
- If a password is stolen, it must immediately be changed. In case the same password or a variation of the same password has been used for another online account, they must also be changed.
- Use complex passwords longer than 8 characters which contain alpha-numeric and special characters (e.g. characters a-z, A-Z, 0-9 along with ',@:?(!)$#/
- A long password does not mean it is hard to remember: four random common words mixed with special characters make a password strong and easy to remember.
- Regularly change passwords for online accounts.
- Make use of passwords managers. Passwords managers are software that chooses and manages passwords for you.
- Take advantage of service providers that offer two-factor authentication. A two factor authentication involves two things, such as something you know (like a password) and something you have (e.g. a one-time password sent to a mobile phone). When possible, use this secure form of authentication to access important services.