SpringerBriefs in Computer Science

For further volumes:
http://www.springer.com/series/10028

Markus Jakobsson

Mobile Authentication

Problems and Solutions

Springer

Markus Jakobsson
PayPal
San Jose
CA, USA

ISSN 2191-5768 ISSN 2191-5776 (electronic)
ISBN 978-1-4614-4877-8 ISBN 978-1-4614-4878-5 (eBook)
DOI 10.1007/978-1-4614-4878-5
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2012942921

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

For A and Art.

Foreword

"Something you are; something you know; something you have." – I first heard these words as a graduate student studying computer security technologies and authentication. These three factors are all we have at our disposal to try to correctly identify other human beings.

In face-to-face interaction, familiar people use "something you are" to identify one another such as their facial structure or voices. When driving through an EZPass toll booth, one uses "something you have" to identify one's car, so that the appropriate account is billed. And when logging into most websites, users typically use "something you know" as the password. Using multiple factors in combination is known to increase security.

While most facets of technology have advanced exponentially, authentication of people to machines has stagnated for quite some time. Most people still use conventional passwords to log into websites for shopping, banking, and other sensitive transactions. We are starting to see small advances in practice, typically in the form of two-factor authentication instead of one, but we have not had the kind of revolution that other areas of technology have enjoyed.

In his new book featuring "duets" with several of his co-authors, Markus Jakobsson gives a fascinating look at current and potential future authentication technologies. He explains why the problem of authenticating users to machines is so difficult and gives a peek under the hood of some of the more promising techniques. For example, while many consider biometrics to be the holy grail for authentication, this book highlights the real benefits as well as the limitations of these techniques.

This book offers a deep understanding of password and PIN schemes and also covers such topics as visual authentication and defeating spoofing. Whether you are a practitioner who needs to understand your options for authenticating users, or a computer scientist who wants to perform research on this important and interesting topic, this book has plenty to offer you.

As a security professional, I began reading this book thinking that it would be a review of concepts I was already familiar with, but I found that I learned a tremendous amount, and think that this book is a must have for anyone in the security field.

Baltimore, May 2012 *Avi Rubin*

Preface

As a society, we have used different forms of authentication since ancient times – of people, documents, materials of value, etc. With the emergence of networked computers in the latter part of the twentieth century, authentication research flourished and many new techniques were developed. Among the central concepts developed or improved upon, we find PINs, passwords, various forms of backup authentication, techniques for device identification, and cryptographic techniques for message authentication.

While consumer habits and the use of legacy systems have hampered changes to authentication systems, we argue that systems designed with these issues in mind can be successfully deployed, and help address global security issues of increasing importance. In this book, we support this argument by describing a collection of new authentication technologies to address unmet authentication needs in a way that minimizes friction, and experimental evaluations of the technologies to quantify the benefits of deployment.

A handset is not just a small computer – it is a small computer with a different user interface. People use it differently. Therefore, mobile authentication is not simply authentication on a mobile device – there are other constraints and enablers. This book focuses on mobile authentication.

While this book provides a view of frontiers in authentication research, we certainly do not make any claims of covering all angles. However, we hope to convince the reader of the value of departing from the status quo and adopting new authentication methods.

Mountain View, California,
May, 2012

Markus Jakobsson
Principal Scientist of Consumer Security, PayPal

Acknowledgements

This book would not have been possible without the contributions of my co-authors – Ruj Akavipat, Mayank Dhiman, Debin Liu, Saman Gerami Moghaddam, Mohsen Sharifi and Hossein Siadati. We have benefitted from insightful discussions with Dirk Balfanz, Jeff Edelen, Aaron Emigh, Nathan Good, William Leddy, Brett McDowell, Jim Palmer, Garry Scoville, and Diana Smetters. Many thanks to Dahn Tamir, who helped in the execution of experiments involving Amazon Mechanical Turk, and to Hampus Jakobsson for assistance with recruiting subjects. We also appreciate the helpful feedback we have received from participants in the user studies underlying many of the chapters. Also, we wish to thank Jeff Hodges, M. Mannan, Netanel Raisch, and Chris Schille for feedback on earlier drafts; Hossein Siadati for LaTex assistance; and Eric Park for editorial guidance. Last but not least, thanks to Michael Barrett for recognizing the benefits of both basic and applied research at Paypal, thereby enabling much of this work.

Contents

Chapter 1
The Big Picture

Authentication is one of the issues at the heart of machine security, and there is a great array of authentication types. To begin with, one can classify authentication methods based on who is authenticating to whom, creating a breakdown into *machine-to-machine* authentication, *machine-to-human* authentication, *human-to-human* authentication, and *human-to-machine* authentication.

Practically speaking, machine-to-machine authentication is well understood. While there is a great number of types, properties, and designs of digital signatures and message authentication codes, there is little commercial demand for improvement. This might mean that we have achieved something close to perfection, or it might simply mean that we are currently satisfied with what we have. Similarly, there is rather limited activity related to machine-to-human authentication. This is a form of authentication that is relatively rare, and mostly used in the context of authenticating a service provider to an end user to avoid attackers from successfully impersonating service providers – as is done in a typical phishing attack. Human-to-human authentication is potentially of great interest in the context of recommendation systems, where the authentication is not of a particular individual as much as of an individual of a particular group; that, however, is outside the scope of this book. This book is about *human-to-machine* authentication, with a focus on the mobile scenario.

Human-to-machine authentication is a very active research area at the time of writing, with a tremendous number of publications related to passwords alone. But human-to-machine authentication is not synonymous to password research; one goal of this book is to make that point. Human-to-machine authentication is a startlingly complex issue. In the *old* days of computer security – before 2000 – the human component was all but disregarded. It was either assumed that people should *and would* be able follow instructions, or that end users were hopeless and would always mess up. The truth, of course, is somewhere in between, which is exactly what makes this so enticing. We cannot make progress with human-to-machine authentication without understanding *both humans and machines*. How can we design secure systems that take into consideration that end users are *people*? People like you and me, but also people *unlike* you and me.

M. Jakobsson, *Mobile Authentication*, SpringerBriefs in Computer Science,
DOI: 10.1007/978-1-4614-4878-5_1, © The Author(s) 2013

I am not making any claims whatsoever at covering the vast expanse of research done in the field of human-to-machine authentication. Instead, this book is written to highlight how many aspects there are to this problem. My aim is to convey the complexity of the issue, and the beauty of looking further into what things *mean* – as opposed to simply what they look like. You will find me peeking often into the mind of the end user throughout the book, arguing both about *what* people do and *why* they do it. I do not believe we can make progress without understanding the problem. The problem, here, is not only what we want to achieve – but equally important, how all the components work. And in particular, how the *human* component works. That said, this is not a psychology book. It is a computer security book that refuses to let go of the question, *"What do people do, and why?"*

The book consists of a collection of short chapters that I co-authored with some of my colleagues who are as passionate about these questions as I am. Each chapter is intended to bring light to one or more questions of relevance to human-to-computer authentication. Each chapter, you may say, argues a point that I feel is worth keeping in mind.

The book begins by describing a neglected aspect of passwords – *how they are made.* It seems embarrassingly obvious to state, but passwords are made by *people*. They are not random strings of characters. People use mental rules when they produce passwords – they concatenate, insert, and modify *components*. These components are typically words, sometimes numbers, and sometimes – typically due to demands from the service providers – other characters. But if that is so, why are we measuring strength in terms of the number of letters and digits? Should we not instead look at the commonality of the components, and the way in which these were combined? Chapter 2 shares some insights that can be gained by looking just at how passwords were made. While this chapter speaks of universal truths, it is well understood that people choose worse passwords in mobile contexts – it is so painful to enter complex passwords there. That makes it even more important to understand what passwords are strong enough, and which ones are not.

Chapter 3 takes a look at PINs. But perhaps not from the perspective you might expect. Instead, the question is asked: what can an organization do when its users have *passwords*, but *need* PINs? It presents a very simple approach by which an organization can bootstrap PIN ownership without any notable use friction. By doing so, it pays attention to maybe the most important issue of applied security: *If it is not easy to use, it will not be used.*

In chapter 4, several questions are asked and answered. Among these are *"How can we obtain more secure human-to-machine authentication?"*; *"How can we improve recall rates?"*; and *"How do we make it easy to authenticate using a handset?"*. As proposed in chapter 2, the search for answers start with the end user. What makes credentials secure? What makes them easy to remember? And what makes them easy to enter in devices with constrained user interfaces?

Chapter 5 pays special attention to the problem of memorability. It is believed that preferences are more stable than long-term memory. If you liked tennis two years ago, you probably still like tennis. And if you disliked olives last year,

that is probably still true. People change, but surprisingly little! An authentication scheme based on preferences can also be more resistant to data-mining efforts than knowledge-based approaches to password reset, in spite of the tremendous amounts of personal information that many typical Internet users share over social networks. This chapter shows how one can create a usable and secure authentication scheme based on preferences. In other words, it is not only about what people can do, but also about *who they are*.

One aspect that is often neglected when people speak about authentication is how to avoid abuse. Phishing, at its heart, is an *abuse* of authentication mechanisms. Chapter 6 describes how an understanding of spoofing can help us change how people authenticate, turning a liability into an asset by making a system in which lack of attention and dependence on habits no longer pose a threat, but become a benefit. This chapter therefore emphasizes the importance of understanding *how and why people fail* – and take that into consideration when designing secure systems.

In chapter 7, we ask: What problems and inconveniences can be addressed by a complete architectural redesign of a system? In particular, given a device with a biometric reader – such as a fingerprint scanner – what can we achieve? And what are the potential problems that may *arise* from a large-scale deployment of biometric readers – and how can these be proactively addressed? Society is rapidly moving towards an increased use of biometrics. We can use such devices the right way – or in less desirable ways. If we understand the issues at stake – issues as diverse as those arising from phishing attacks and replacement of devices – then we can produce a solution that does not simply replace passwords, but which creates new benefits.

Finally, we consider what can be done in a world inhabited by legacy servers. A complete redesign of the Internet and everything that connects to it may solve many of our problems in a very elegant way – but is this an approach that we can meaningfully expect? Chapter 8 describes how one can change the world while respecting legacy servers. I am *certain* that the approach I will describe cannot be used to achieve *any* change we may hope to make – but this book is about inspiration and first steps!

Chapter 2
The Benefits of Understanding Passwords

Markus Jakobsson, Mayank Dhiman

Abstract

In an effort to assess the strength of passwords, password strength checkers count lower-case and upper-case letters, digits and other characters. However, this does not truly measure how likely a given password is. To determine the likelihood of a password, one must first understand how passwords are generated – this chapter takes a first step in that direction. This is particularly important in a mobile context, where users already are tempted to use short and simple passwords – given how arduous password entry is.

2.1 Why We Need to Understand Passwords

While we do not think that passwords are the best way for people to authenticate to their devices and service providers, it is important to recognize the degree to which passwords are part of infrastructure, which makes them difficult to replace – even if we agree on what to replace them with.

This chapter describes a new method by which we can address two common problems relating to traditional passwords. The first problem is that of approximating the security of a given credential. Traditional password strength checkers plainly demand the presence of certain predicates – such as a combination of uppercase and lowercase; the inclusion of numerals; and that passwords do not match any of the most common passwords (such as "abc123"). This is not necessarily the optimal strategy, as it does not capture common transformations (such as from to common password "password" to the very similar "passw0rd"). Rather than extending the blacklist to all common variants of all common passwords, it is better to understand the underlying structure of passwords, and how people generate them. This allows us to score passwords based on how they were generated – doing this allows us to determine that "p1a2s3s4w5o6r7d" is somehow less secure than "a1d9o8g4." Without an understanding of how passwords are generated, the former password is likely to be believed to be the stronger of the two. (Of course, if mindless exhaustive search is the only path of compromise, the former password *is* the strongest of the two – so security must be seen in context of the most prevalent threat.)

M. Jakobsson, *Mobile Authentication*, SpringerBriefs in Computer Science,
DOI: 10.1007/978-1-4614-4878-5_2,

The second problem this chapter addresses is how to identify credential reuse – whether sequentially for one account, or consecutively between two or more accounts. Here, we do not only consider *verbatim* reuse, but also *approximate* reuse – such as "BigTomato" and "bigTOMATO1." If a person has two accounts with different passwords then loses one of the passwords to a fraudster, then the fraudster has a reasonable chance to get access to the other account as well. This is because the attacker may try all common transformations of the stolen credential, hoping that one of them will work for the second account. As a result, we need to identify and discourage both verbatim and approximate password reuse. While verbatim reuse can be detected without any understanding of the underlying credentials, detection of approximate reuse requires a structural understanding of passwords.

The two techniques described herein are closely related, and are both based on the parsing and decomposition of passwords, using rules matching those people are relying on when they *generate* passwords. Examples of such rules are insertion of one component into another component; concatenation of components; and common transformations of elements of a component.

2.2 People Make Passwords

A good password is hard to guess. Conversely, of course, a bad password is *easy* to guess. But what is it that makes something hard to guess, and how can we tell?

It is easier to tell that something is *easy* to guess than that it is hard to guess. For example, the following potential passwords are easy to guess: *fraternity* (a dictionary word); *$L* (a very short string); *qwertyuiop* (a string with a very predictable pattern); and *LoveLoveMeDo* (famous lyrics). Similarly, one can look at the commonality of passwords – any user who wants to use a password that has already reached the limit has to think of another password. This approach is taken by Schechter, Herley, and Mitzenmacher [81]. We can make a long list of reasons to consider a password to be weak – and this is what typical password strength checkers do – but how can we tell that we have not missed some?

To be able to determine what makes most sense, we need to understand how passwords are constructed. Passwords are constructed by *people*, and people follow guidelines and mental protocols when performing tasks. Therefore, a better understanding of passwords requires a better understanding of people – or at least how people construct passwords.

To gain a better understanding of this, we collected a very large number of actual passwords. We sampled and reviewed these, thinking carefully about how each password was constructed. It is meaningful to think of passwords as strings that are composed of *components*, where components are *dictionary words*, *numbers*, and *other characters*. When producing a password, a typical user composes a password from a small number of such components using one or more *rules*.

The most common rule is *concatenation*, followed by *replacement*, *spelling mistake*, and *insertion*. Here, an example of a concatenation is producing "passbay1"

from the three components "pass," "bay," and "1." Use of *L33T*[1] is a common replacement strategy, creating "s3v3nty" from "seventy" by replacement of each "e" with a "3." Misspellings may be intentional or unintentional, resulting in passwords such as "clostrofobic." Finally, insertion produces strings such as "Christi77na," where "77" was inserted into the name "Christina." (This was the least common type of rule among those we surveyed and the hardest to automatically detect in practice, so this rule was not used in the experiment we describe herein.)

The simple insight that *people* choose passwords suggests a new approach to determining the strength of a password: One can determine the components making up the password and the commonality of each such component; one could then consider the mental generation rules used to combine the components and make up the password – along with the commonality of these rules being used. The strength of the password, in some sense, depends directly on the commonality of the password, which in turn depends on the commonality of its components and password generation rules. Similarly, when determining the similarity of two passwords, one can compare the components that make up the two passwords along with the generation rules. It is therefore important to understand the commonality of components and rules.

2.3 Building a Parser

Components and Rules

To build a parser, we need to understand the components and the generation rules, and then "decompile" passwords into the components they were made from. We will therefore review how most passwords are formed, in order to understand how to invert this process.

Examples

- **Concatenating components.** The password "mylove123" consists of three components: "my," "love," and "123" combined with two occurrences of the concatenation rule. By determining the observed frequencies of the three components and the concatenation rule, it is possible to assess the "likelihood" of the password.
- **Order matters.** The password "my123love" has the same components and uses the same rules – except it does not attach the numerals at the end. A simple analysis of passwords shows that most passwords containing numerals have them at the end. And since the goal of parsing is to assess the likelihood of a given password – its believed strength – we see that order matters. Put a different way,

[1] L33T is pronounced "leet," and is a relatively common transcription of words in which letters are replaced by other characters with some resemblance to the replaced letter.

the "value" of the component "123" is greater when it is not at the end of the password.

- **Insertion.** Next, the password "mylo123ve" is even stronger, given that it does not only use concatenation but also insertion: the user has inserted the component "123" inside the component "love," which, in turn, was concatenated to the component "my."
- **L33T and multiple parsing possibilities.** The password "myl0v3" has two components, "my" and "l0v3," where the second component is L33T for "love." Here however, "l0v3" can either be seen as a dictionary word whose frequency is recorded and used to score passwords in which it occurs, or it can be seen as the result of a L33T-rule applied to a dictionary word. While the former approach is more straightforward, it does not benefit from knowledge of the observed frequencies of the underlying words. The latter approach, on the other hand, may result in errors where the L33T term is more common – relatively speaking – than the associated word. One approach to address this is to parse the password both ways and use the more conservative assessment.
- **Maximizing coverage.** Finally, the password "thinput" shows that there could be multiple paths. This could either be the result of concatenating the two character components "t" and "h" with the component "input," or it could be the result of concatenating "thin" and "put." Like in the previous example, it is possible to produce multiple parsing results and then select the most conservative. In the parser we describe, though, a greedy word-based approach was taken, where passwords are considered combinations of the words that "cover" the biggest portion of the strings. This is a result of the observation that most passwords are based on words, which surely is due to the fact that people relate better to words with meaning than to strings of random characters.

What the Parser Does

The parser takes a password as input and outputs the various components and rules which have been used to construct that password – or *appear* to have been used, to be specific. As we have mentioned, components include words, numbers, individual characters, and special symbols. Some components may be overlapping, e.g., a word is made up of number of characters. In such cases, those components are parsed first which have longer length. This minimizes the number of components that are used to "cover" the credential.

The parser has built-in rules to identify three major password creation rules: *concatenation*, *L33T*, and *misspelling.*

In order to parse an input, the parser uses an *input dictionary*. It is worth noting that language has strong influence on the creation of passwords and that words usually form the core around which various other operations are applied to make the password more complex. For simplicity, only English dictionaries are used herein.

The *input dictionary* contains words from different sources, such as a standard English dictionary, people's first and last names, geographical locations, technical terminology, and so forth. A vanilla English dictionary is not sufficient because many passwords contain words and phrases from the latter categories. To achieve a better and more comprehensive dictionary, we combined specialized dictionaries based on such topics. The final dictionary obtained upon merging these specialized dictionaries contained just below 670,000 words.

The parser contains algorithms capable of identifying password creation rules. The parser uses the input dictionary and the specified algorithms to identify the components and rules for the input password. The detection of components is directly dependent upon the choice of input dictionary.

Note that in many cases the passwords can be broken into components in more than one way. In such cases, the parser selects the components containing words with the *maximum coverage*. Maximum coverage means that the total length of the combined components, which are words, is maximized. For example, "thinput" will get parsed as "thin" and "put" rather than "t," "h," and "input." When two paths produce the same coverage, the path with the greatest probability of occurrence (as judged by the frequency of use of the rules and components) is chosen. Practically speaking, this typically corresponds to the path with the smallest number of components. Therefore, "password123" will be parsed as "password" and "123," rather than "pass," "word," and "123." The details of how the parser operates are given the next subsection.

Parsing Details

The three main subroutines which are required for parsing components are *generate-all-substrings*, *generate-max-substrings* and *generate-leftover-components*, which we will describe in detail next.

Consider a potential password "hello1." A call to *generate-all-substrings* will generate all possible substrings of a given string. Hence, a call to *generate-all-substrings* using this input generates "h," "e," "l," "o," "1," "he," "el," "ll," ..., "hello," "ello1," "hello1."

The *generate-max-substrings* subroutine takes in a password as an input and generates a set of substrings, which are words and can be found in the input dictionary. Only that set of substrings is returned which provides the *maximum coverage* of the input string. For example, using *generate-all-substrings* as a subroutine, and an input "hello2(world!35b," the subroutine *generate-max-substrings* returns a list containing "hello" and "world."

The third subroutine *generate-leftover-components* requires two arguments: the password and the list of substrings generated by *generate-max-substrings* using the password as an input. This subroutine will return the remaining components – i.e., consecutive numbers, characters and special symbols – after cutting the substrings generated by *generate-max-substrings* from the password. Thus, calling *generate-*

leftover-components with the inputs of “hello2(world!35b” and [“hello,” “world”] returns “2,” “(,”“!,” “35,” and “b.” Notice that the string is broken up into conceptually consistent pieces – e.g., “!35b” is split into “!,” “35,” and “b.”

Regular Components and Concatenation

In order to identify the components, the parser first checks if the password can be found in the input dictionary. In that case, the password contains only one component, the password itself. For example, “monkey” is one of the most commonly used passwords, which can be found in the dictionary. In the next step, the parser checks if the password contains numbers only. For example, keyboard patterns like “12345” and dates of birth are quite common passwords. The first two steps are performed in this order due to efficiency reasons. This is because an average of 25% of all passwords in most datasets are either dictionary words or complete numbers.

In order to detect concatenation, the parser uses the *generate-max-substrings* subroutine described above, which generates the set of substrings that provide *maximum coverage*. These results are combined with the results of *generate-leftover-components* to generate all the components of the password.

Identifying L33T

Since, L33T is not a proper language and many different dialects of L33T occur on the Internet, there is no perfect algorithm to convert between English and L33T. Since the rules for such conversion vary, one can use an exhaustive approach. One can construct a table containing all mappings from a token to be replaced (number or special symbol) to a list of all common replacements for that token, as seen in various L33T to English translators. For example, the character “0” is substituted for “o” and the character “|” can be substituted for both “i” and “l.” Hence, in the mapping table there will be an entry for “@” to be mapped to “a” and “|” to be mapped to both “i” and “l.”

Identification of L33T works in combination with the *generate-max-substrings* subroutine. The *generate-max-substrings* subroutine tries to replace the tokens with all possible characters found in the mapping of token in the L33T to English mapping table. All possible combinations are tried. For each combination tried, the parser tries to find the maximum covering strings. If the new substrings generated covers a larger length than the previously calculated substrings, then that character is replaced by the new character from the mapping. Consider the password “||ovey0u.” Each token which is not a letter is considered for replacement – in this example, there are three such characters. For each combination of “reasonable” substitutions, *generate-max-substrings* is run. The substitution with the greatest coverage is selected and output; in our example, “||ovey0u” would be converted to “iloveyou,” which then would broken into the three words it consists of.

Identifying Spelling Mistakes

Identification of spelling mistakes is more complex. Initially, the *generate-max-substrings* subroutine is called and the output of it is then used as input for the *generate-leftover-components* subroutine in order to examine leftover components. For example, consider the password "heilloworld." A call to *generate-max-substrings* will generate "world" which along with the initial password is used as an input to *generate-leftover-components* which will generate "heillo." This is the leftover string and potentially contains a spelling mistake. Before starting to parse the passwords, a training module is used to train for detection of spelling mistakes using an input dictionary. There is another component of this training module, which tries to detect and correct the spelling of each component, if possible. In case a spelling mistake is found, then a new string is generated in which the spelling mistake has been rectified, and the subroutine *generate-max-substrings* is called for a second time using this new string. This is because in certain cases, the incorrectly spelled word may be used to create longer words. For example, "jidgement" will initially generate the component "gem," but after the spelling mistake has been rectified, calling *generate-max-substrings* will generate "judgement."

2.4 Building a Model

The parser we have described can be used to process large sets of passwords, producing the associated components and rules used to produce the passwords in the sets. By recording the frequencies of each of the components observed – and the rules that were used – one can produce a stochastic model for how passwords are generated. This model can then be used to score passwords. In the following, we will describe how to build the model.

Approach

In order to build a model of passwords, one needs a large collection of passwords and an input dictionary. One can use any set of passwords to train the system. The larger the dataset is, the more accurate the resulting frequency database will be. If one small dataset is considered higher quality – i.e., more accurate – than a larger dataset, then one can perform "weighted" training using both. In the example we describe, however, we only used one dataset for training: The *RockYou dataset*. The RockYou dataset contains 32 million passwords, leaked in 2007. During training, we parse passwords from the RockYou dataset and generate various rules and components. In the process, we populate our input dictionary with the frequencies of occurrences of each rule and component.

After this training phase, we obtain three *trained dictionaries* containing the component/rule and the corresponding frequency of occurrence. These include a dictionary of words, a dictionary of numerals, and a dictionary of characters and symbols. Upon breaking down a password into the components and rules, the score calculator makes a search count for the components and rules in the *trained dictionaries*. Using these values, a score corresponding to the input password is calculated using the scoring algorithm.

Characteristics of Password Collections

After deriving a model by training the system with a large number of passwords, this model can be used to assess the strength of passwords. We test this by computing password scores from five datasets, each one of which corresponds to a collection of passwords. Each dataset has different characteristics, as will be shown, which influences the average strength of the passwords and hence, the password scores. We first describe three major characteristics of all datasets.

The first obvious characteristic is the *type of resource being protected*, i.e., what would be lost if a password is stolen. We consider only the losses as perceived by the user who produces the password, and not those potentially suffered by other users as a result of theft or the losses of the organization associated with the password. The rationale is this: users are influenced by the risk they associate with theft when they select their password. For financial service providers, money would be lost. For social networking sites, the user would lose face or be inconvenienced. For a porn site, the loss may be limited to the access to the site. In addition to this, some private information, such as user name and address, may be at stake for all of these types of sites.

A second characteristic is the *demographics of users*. While it is not known how demographics affect password strength, it appears likely that they do.

Finally, the *collection method* could introduce a bias in the dataset. For example, if a dataset is obtained by malware attacks, the dataset may have a greater percentage of passwords from people who are not security conscious than if it was obtained by site corruption. Similarly, a dataset associated with phishing is likely to have a greater percentage of passwords from people who are gullible than other datasets would.

In the following, we will describe results associated with five datasets. The first dataset, the *Rootkit dataset* contains 64498 passwords. It was obtained by a compromise of rootkit.com. This website has forums about discussions of advanced topics in computer security. The demographics introduce a bias as relative to an average user, as most people registering on this website are more aware about security issues than typical users. Second, the *Paypal dataset* contains 19053 passwords. Most of the users behind the passwords are adults. The method of collection is phishing, which introduces a bias towards people who are more gullible. Third, the *Justin Bieber dataset* contains 5091 passwords and was obtained by a compromise of a fan website. Hence, there is no bias due to the collection method. However, the demo-