Analyzing Breach Data to Uncover Student Exposure & Password Trends

By

George S. Major

Bachelor of Science
in
Information Security & Intelligence
Ferris State University, 2017

Advisor:

Dr. Greg Gogolin
Full Professor
College of Business

Fall, 2018
Ferris State University
Big Rapids, MI

Table of Contents

Chapter 1 9
Introduction 9
Background 10
Purpose of the Research 11
Rationale 11
Research Questions 12
Chapter 2 13
Early Password Cracking & Analysis 13
Breach Password Data 15
Password Reuse & Modification 19
Chapter 3 21
Description of Methodology 21
Research Focus 21
Design of Study 22
Chapter 4 32
How many student records can be identified within the source breach dataset? 32
What password creation & reuse trends exist among the breached student passwords? 32
Central Michigan University 37
Eastern Michigan University 42
Ferris State University 47
Michigan State University 52
Northern Michigan University 57

Western Michigan University 62

Wayne State University 67

University of Michigan 73

Chapter 5 79

Summary of Findings 79

Conclusions 80

Recommendations 81

Recommendations for Further Study 82

Appendix A 86

Appendix B 95

List of Tables

Table 1. Breach Records Recovered by Institution. 33
Table 2. Accounts Breached Multiple Times by Institution. 34

List of Figures

Figure 1. Distribution of passwords by length for all recovered passwords. 34
Figure 2. Average Levenshtein Distance by count of user passwords for all recovered data. 35
Figure 3. Count of Average Levenshtein Distance less than 4 by number of passwords per user. 35
Figure 4. Top ten passwords across all datasets. 36
Figure 5. Top ten root strings across all datasets. 36
Figure 6. CMU users by number of times breached. 37
Figure 7. Bar Chart: CMU passwords by length. 38
Figure 8. Pie Chart: CMU users by times breached. 38
Figure 9. CMU passwords by length. 39
Figure 10. Most common passwords within the CMU dataset. 40
Figure 11. Distribution of Levenshtein Distance averages for CMU. 40
Figure 12. Bar Graph: Distribution of Levenshtein Distance averages for CMU. 41
Figure 13. Most frequent password strings within the CMU dataset. 41
Figure 14. EMU users by number of times breached 42
Figure 15. Pie Chart: EMU users by times breached. 43
Figure 16. Bar Chart: EMU passwords by length. 44
Figure 17. EMU passwords by length. 44
Figure 18. Most common passwords within the EMU dataset. 45
Figure 19. Distribution of Levenshtein Distance averages for EMU. 46
Figure 20. Most frequent password strings within the EMU dataset. 47
Figure 21. Bar Graph: Distribution of Levenshtein Distance averages for EMU. 46
Figure 22. FSU users by number of times breached. 48
Figure 23. Pie Chart: FSU users by times breached 48
Figure 24. Most common passwords within the FSU dataset. 49
Figure 25. Bar Chart: FSU passwords by length. 49
Figure 26. FSU passwords by length. 50
Figure 27. Distribution of Levenshtein Distance averages for FSU. 50
Figure 28. Bar Graph: Distribution of Levenshtein Distance averages for FSU. 51
Figure 29. Most frequent password strings within the FSU dataset. 51
Figure 30. MSU users by number of times breached. 52
Figure 31. Pie Chart: MSU users by times breached. 53
Figure 32. MSU passwords by length. 54
Figure 33. Bar Chart: MSU passwords by length. 53
Figure 34. Most common passwords within the MSU dataset. 55
Figure 35. Distribution of Levenshtein Distance averages for MSU. 55
Figure 36. Bar Graph: Distribution of Levenshtein Distance averages for MSU. 56
Figure 37. Most frequent password strings within the MSU dataset 56
Figure 38. NMU users by number of times breached. 57
Figure 39. Pie Chart: NMU users by times breached 57
Figure 40. Bar Chart: NMU passwords by length 58
Figure 41. NMU passwords by length. 59
Figure 42. Most common passwords within the NMU dataset. 60
Figure 43. Distribution of Levenshtein Distance Averages for NMU. 60
Figure 44. Bar Graph: Distribution of Levenshtein Distance averages for NMU. 61
Figure 45. Most frequent password strings within the NMU dataset. 61
Figure 46. WMU users by number of times breached. 62
Figure 47. Pie Chart: WMU users by times breached. 63
Figure 48. WMU passwords by length. 64

Figure 49. Bar Chart: WMU passwords by length. 64
Figure 50. WMU passwords by length. 65
Figure 51. Distribution of Levenshtein Distance averages for WMU. 66
Figure 52. Bar Graph: Distribution of Levenshtein Distance averages for WMU 66
Figure 53. Most frequent password strings within the WMU dataset. 67
Figure 54. Pie Chart: Wayne State users by times breached. 68
Figure 55. Wayne State users by number of times breached. 68
Figure 56. Bar Chart: Wayne State passwords by length. 69
Figure 57. Wayne State passwords by length. 70
Figure 58. Most common passwords within the Wayne State dataset. 71
Figure 59. Bar Graph: Distribution of Levenshtein Distance averages for Wayne State. 71
Figure 60. Distribution of Levenshtein Distance averages for Wayne State. 72
Figure 61. Most frequent password strings within the Wayne State dataset. 73
Figure 62. U of M users by number of times breached. 74
Figure 63. Pie Chart: U of M users by times breached. 74
Figure 64. U of M passwords by length. 75
Figure 65. Bar Chart: U of M passwords by length. 76
Figure 66. U of M passwords by length. 77
Figure 67. Distribution of Levenshtein Distance averages for U of M. 77
Figure 68. Bar Graph: Distribution of Levenshtein Distance averages for U of M. 78
Figure 69. Most frequent password strings within the U of M dataset. 78

Abstract

This study attempted to quantify the data breach exposure of students at select universities in Michigan, and to analyze their password creation behavior. A large, 1.4 billion-record corpus of breach data was examined for student email addresses corresponding to eight prominent Michigan universities. Institutions included within this study were: Central Michigan University, Eastern Michigan University, Ferris State University, Michigan State University Northern Michigan University, Western Michigan University, Wayne State University, and University of Michigan. A total of 144,083 email/password records were found, corresponding to 121,215 distinct email addresses. We found that university users more frequently employed password mutation techniques as opposed reuse of identical passwords. Common passwords within each dataset were predictable and often university-centric, however a majority of users employed unique passwords.

Chapter 1

Introduction

Sitting behind a computer terminal in a non-descript apartment just outside of Lebanon Kansas is the "security expert". He's earned that title less through academics and more through experience: breaking into computer systems is his favorite past time, and recently, it's been keeping the lights on. He's discovered here's a lot of money to be made selling personal information on the Internet, especially if it's fresh. Names, dates of birth, social security numbers, college transcripts…the more he can bundle together, the more it goes for.

He's staring at the login page of his latest target of interest, your university. Impressively—annoyingly—it seems that the security team has done a pretty good job at locking down the public-facing portions of the university's website. He wonders…has the same attention to detail been applied internally, from the perspective of a credentialed user? As luck would have it, he as a couple sets of university user credentials that might work—40,000 sets actually—sourced from a number of major data breaches of entertainment, news, and gaming websites, on his hard drive. A flash of the keyboard later, and he's pulled up a promising candidate: this user has been breached on three separate websites using his university email. On each website, it seems that he's used the same password. If he's used it repeated elsewhere, perhaps he's used it for the official university site as well. "Hmm…worth a shot anyway", the hacker thinks.

Returning to the university login page, he types "Football23" in a steady cadence, followed by the return key. The web page hesitates for just a second, as if thinking it over, then promptly produces a concise landing page with the university logo in the top-right corner. "Too easy", the hacker smiles inwardly.

Although a fictious story, the scenario is not impossible; there are many users out there

following less-than-optimal practices, such as password reuse. We know this to be the case, in part because every year databases containing login credentials are breached, and that data is made public and readily available to anyone who cares to analyze it. Once a data breach occurs and the data is made public, the damage is done. The information will likely be online somewhere and accessible forever by the bad guys, but also the good guys. The silver lining is that we can leverage breach data for the common good: by using it to determine our level of exposure, and to uncover exploitable password trends, we can make our accounts, and our networks more secure.

Background

Passwords remain the most common form of authentication for online services (Haque et al., 2013). They are the lynch pin that allows a user, and only that user, to access a particular online account, or to assume a particular online identity. By providing the correct password, an online service can be assured that the user is who they say they are. In a perfect world, it would be impossible for anyone else to access that your user account. Unfortunately, as an industry we've yet to create an ideal, fool-proof system. Attackers continue to find ways to compromise user passwords, assume online identities of victims, and to use victim's privileges to perform some malicious activity.

Passwords by their nature are meant to be kept secret, so what happens if the password for, say, your email account is breached and available for the whole world to see? You may think: "So what, I hardly use it anyway". This may be the case, but if a user has set the same password for a critical service, such as their online banking? What if the user's password contains identifiable trends that attackers could allow attackers to guess other passwords? The effect of a trivial data breach may be magnified greatly. By analyzing breach data, we can uncover these potential problems and devise ways to minimize their potential negative effects.

Statement of the Problem

Michigan university students are not immune to password breaches, nor are they immune to attacks against passwords. As a subsect of the general population of Internet users, they may be uniquely susceptible to compromise due to specific password creation strategies or predicable password trends employed. If such trends indeed exist, they be discovered by attackers and used for malicious activity. Furthermore, universities and affect students may not fully realize the scope or threat that data breaches pose. This research attempts to address a lack of information related to student breach exposure and student password heuristics.

Purpose of the Research

The purpose of this research will be to aggregate and analyze the publicly-exposed credentials of students that attend(ed) one of several well-known universities in Michigan. The goal of the research is to quantify student involvement in data breaches, to analyze student passwords for exploitable trends, and to leverage that information in order to inform and improve account security. To date, the author is unaware of any research which attempts to ascertain, compare, and analyze student breach involvement in this manner. This work is intended to contribute to the body of knowledge related to institutional information security by aggregating, analyzing, and reporting such statistics.

Rationale

Numerous breach datasets exist in the public space. Locating, collecting and aggregating these datasets may allow them to be examined using big data tools. By searching breach credential sets that contain email identifiers, students from universities of interest to this study may be identified. For example, Ferris State University students can be identified by searching for FSU's unique email domain "@ferris.edu". After accounts of interest have been identified, they may be extracted and complied into a special dataset. This dataset may be used to quantify,

compare, and contrast the number of breached accounts among various universities. The structure of passwords in the dataset may then be analyzed for content and trends. Statistics and trends discovered during the course of the study can be reported and may be used to improve account security and breach awareness among university students.

Research Questions

This work attempts to answer the following research questions with respect to the universities studied:

1. How many student records can be identified within the breach dataset(s) examined?
2. What password creation & reuse trends exist among the breached student passwords?

Chapter 2

This literature review presents, categorizes, and discusses existing research related to password analysis. Information and findings from past research in the field provide much of the basis for this study, and they provide critical reference for interpreting the methodology used chosen.

Early Password Cracking & Analysis

Password recovery and analysis as a vector for assessing account security can be traced back at least to the seminal work of Morris and Thompson (1979). The pair used a 3,289-password dataset collected from Unix users over a "long period of time", to perform password content analysis (Morris & Thompson, 1979, p. 596). Morris and Thompson determined that 71% of passwords collected were six characters or less (1979). They also reported that 84% of passwords in their sample were recoverable using a password guessing attack (Morris & Thompson, 1979). Much of the subsequent research produced since Morris and Thompson's (1979) article expands upon password analysis concepts that were first explored by Morris and Thompson.

Klein's (1991) research utilized a larger Unix sample than that of Morris and Thompson and introduced rudimentary transformation techniques in an attempt to improve the effectiveness of password guessing. Klein compiled a list of nearly 15,000 Unix-type account records from survey respondents located in Great Britain and the United States (1991). Using various methods, was able to crack 21% of his dataset (roughly 3,150 passwords) within one week (Klein, 1991). Klein also attempted content analysis against the passwords recovered, finding 52.2% were found to be six characters or less (1991). Around the same time, Spafford (1992) studied length and content of university passwords. Spafford compiled a plaintext database containing around 19,100 password change records at Purdue University during a ten-

month collection period (1992). After preforming a statistical analysis of password length and character content, Spafford used a dictionary attack against an all-lowercase version of the sample (1992). 2754 passwords (20%) were “quickly found using simple dictionary and wordlist lookups…” by Spafford (1992, p. 11).

Within the same decade, Wu explored offline cracking and password analysis of Kerberos authentication systems (1999). In a two-week experiment utilizing a Kerberos realm of around twenty-five thousand users, Wu was able to guess 2,045 (8%) of the passwords (1999). After analyzing the recovered passwords, Wu reported that nearly 96% of recovered passwords were eight characters or less, and that 49% of passwords were recovered without the use of any transformation rules (Wu, 1999).

Early password analysis attempts were largely limited by the datasets available for study. Researchers had to rely upon voluntary submission as a collection method, resulting in relatively small, (and potentially biased), datasets (Morris & Thompson, 1979; Klein 1990; Spafford, 1992). When users are asked to submit passwords for use in the study, the knowledge that their password choice will be scrutinized could alter their behavior. A knowledgeable user might submit a “better” password for the study than one they would typically use. Result bias is another common potential problem with such studies. In studies where the entire dataset of passwords is not known, (Klein, 1990; Wu, 1990) researchers could only analyze passwords that could be cracked using then-current technology (21%; 8% of the datasets). If the hashed password was strong enough to withstand cracking attempts, researchers could not analyze it, therefore the information gleaned from such studies was inherently limited; the usefulness of the data set was limited to the offensive potential of research team. Given the cracking speed that early researchers could achieve, massive recovery of passwords proved difficult within

reasonable periods of time, which might explain the modest recovery success rates. To some extent, cracking speed may have hindered researchers' ability to fully demonstrate the effectiveness of devised password recovery methods. Wu (1990) discussed this issue at length:

> "It is estimated that our password cracker verified slightly over 100 million candidate passwords, distributed over the eight different workstations…it was only a small fraction of the total number of candidate passwords available to the password cracker. Had the experiment been allowed to continue for a greater length of time, a larger portion of the total password space could have been searched…" (Wu, 1990).

If those early studies were replicated today using modern equipment capable of recovering larger portions of the dataset, it is possible that very different results would have been produced.

Breach Password Data

> The authors of the 2006 version of NIST SP 800-63 described the problem succinctly: "Unfortunately, we do not have much data on the passwords user choose under particular rules…NIST would like to obtain more data on the passwords users actually choose [however] system administrators are understandably reluctant to reveal password data to others" (Burr et al., 2006, p. 47).

To overcome the sample collection limitations apparent in earlier password research (Morris & Thompson, 1979; Klein, 1990; Spafford, 1992), some researchers began to use datasets sourced from online data breaches—passwords stolen by attackers or otherwise exposed to the public. There are two obvious research advantages for the use of breached password datasets: realistic data can be gathered which with a reduced possibility for research bias, and researchers do not have to spend long periods of building a data set. When polling or informing users of password collection there is a chance that they will alter their password creation behavior, which could

adversely affect any research conclusions. Breach password data is unquestionably realistic when properly vetted because it comes from actual user accounts. Since these breach datasets are often readily accessible online, aggregation of hyper-realistic data for research analysis can be performed quickly and with relative ease.

When considering the use of breach data, one cannot overlook ethical concerns; it is important to ensure any study conducted is morally and ethically sound. The APA Publication Manual describes three primary ethics goals for academic research: "to ensure accuracy of scientific knowledge"; "to protect the rights and welfare of research participants"; "to protect intellectual property rights" (2010, p. 11). Furthermore, the first General Principle of the American Psychological Association (APA) Ethics Code is "Beneficence and Nonmaleficence". It reads in part:

> "…psychologists seek to safeguard the welfare and rights of those with whom they interact professionally and other affected persons …scientific and professional judgements and actions may affect the lives of others…guard against personal, financial, social, organizational, or political factors that might lead to misuse…." (American Psychological Association, 2017, para. 12).

Therefore, use of breach data should not increase the risk of harm to the users being studied. To ensure that is the case, researchers should take prudent steps to ensure that research methods are consistent with ethical guidelines. It is common for researchers to obtain the approval of ethics committees prior to conducting password related research (Das et al., 2014, Mazurek et al., 2013, Wang et al., 2017). Various techniques may be employed to minimize risks, such as restricting the use of breach data to datasets that are completely publicly available (Wang et al. 2017) or the use of anonymization techniques to prevent user identification and hash reversal

(Bonneau, 2012).

In 2005, Narayanan and Shmatikov pioneered use of Markov models—algorithms typically used for natural language processing—to reduce password cracking keyspace to probable human-memorable passwords. Narayanan and Shmatikov utilized this approach as part of a hybrid cracking attack against a small real-world dataset consisting of 150 leaked passwords sourced from the website Passware (2005). They reported successful recovery recovered 67% percent of passwords using a search space of $2x10^9$ (Narayanan & Shmatikov, 2005).

Another early example of breach dataset analysis was conducted using Myspace.com username and password pairs, which were initially obtained by attackers through a phishing attack and later published on the Internet (Schneier, 2006). Notably, Schneier discovered that 65% of passwords contained eight characters or less, while 17% were made up of six characters or less (2006). Dell'Amico et al. (2009) performed subsequent password strength analysis against the Myspace dataset.

Expanding upon the work of Narayanan and Shmatikov, Dell'Amico et al. (2009) used Markov modeling-based attacks to exploit regularities that might exist in passwords, such as pronounceable substrings. Through the use of Markov modeling, Dell'Amico et al. (2009) concluded that password strength cannot be determined strictly by examining password length. "A short password containing infrequent characters and/or sequences thereof can actually be stronger than a noticeably longer one" (2009, p 10). The concept of identifying vulnerable substrings is discussed often in later research.

Much contemporary research has focused on understanding user password-generation behavior among multiple webservices. By aggregating breach data from numerous sources,

scholars could correlate user accounts among multiple services and study password trends on a larger scale. Liu et al. (2014) conducted a large-scale password analysis study using some twenty million records sourced from four popular Chinese web sites. Liu et al. found that among the four sites, passwords were often comprised of digits only; datasets from two of the four sites consisted of 60% and 70% digit-only password distributions (Liu et al., 2014). Examining password substrings from the Chinese datasets revealed that the incorporation of usernames, email addresses, phone numbers, or birthdates was common (Liu et al., 2014).

Wang et al. (2017) utilized 61.5 million passwords aggregated from 107 password services to study the password re-use behavior of 28.8 million users. Wang et al. determined that 38% of users included in their study had reused at least one password, with a particularly high rate of password reuse among email accounts (60.4%) (2017). Wang et al. examined the prevalence of password modification through use of substrings, determining 21% of users had used a modified password for a new service at least once (2017). Wang et al. concluded that users with five or less passwords were more likely to reuse passwords whereas users with more than five demonstrated a tendency to modify passwords (2017).

Das et al. (2014) performed a similar study on password reuse which utilized breached datasets to locate 6077 users with at least two sets of breached credentials. Analysis of the user credential sets found 43% used an identical password, while 19% contained some substring of one another (Das et al., 2014). Bailey, Durmuth, and Paar measured the re-use and strength of breached passwords obtained from four online sources, with an emphasis on financial accounts (2014). One list obtained from a malware source contained approximately 3,500 account credentials for 1,700 users, to include 177 identified as “high value” financial accounts (2014). Bailey, Durmuth, and Paar found 14% of passwords were reused among online accounts, while

an additional 19% contained were substrings of other passwords within an edit distance of 0.2 (passwords were at least eighty percent similar) (2014). Among financial accounts, Bailey, Durmuth, and Paar found that reuse rates of 21% and 26% respectively, which would indicate higher password reuse among financial passwords (2014). Results from both Das et al. (2014) and Bailey, Durmuth, and Paar (2014) seem to be consistent with that those from Wang et al. (2017): password reuse and password modification seem to be common methods for creating new passwords.

Password Reuse & Modification

Notable research has been conducted surrounding user password strategy across multiple accounts. Several studies which utilized breach datasets have already been discussed, (Bailey, Durmuth, & Paar, 2014; Das et al. 2014; Wang et al., 2017), however there are others non-breach studies which also warrant inclusion. [Research on number of accounts] found that older and more experienced internet users frequently have more online accounts than younger or newer users. It stands to reason that as the number of user accounts increases, so must the number of passwords that users must remember. Therein lies a vulnerability: users might be tempted to reuse passwords among multiple accounts to reduce the number they have to keep track of. Bailey, Durmuth, and Paar postulated that if a rogue site can collect a user's password, it may be may tested against other sites (2014). If the user has set the same password for multiple services, an attacker could easily breach every such service.

Florencio and Herley collected voluntary data on password usage from 500,000 Microsoft users over a three-month period (2007). After examining the data collected, Florencio and Herley postulated that strong passwords were shared less frequency among multiple websites (a mean of 4.48 sites) compared to weaker passwords, which were more frequently reused (6.06 sites) (2017). They also theorized that users tended to create stronger

passwords for sites deemed more important: on average users created stronger passwords for Paypal and Microsoft's corporate website than for the New York Times newspaper website (Florencio & Herley, 2007). Notoatmodjo et al. also examined this phenomenon, theorizing that users tend to group various online accounts into "security levels" (2009). In a small experiment consisting of 26 university student participants, Notoatmodjo et al. discovered that among 68 "high importance" accounts, 63% were assigned unique passwords (2009). Among 253 "low importance" accounts, only 32% were assigned unique passwords (Notoamodjo et al., 2009).

Haque et al. conducted a follow-up laboratory experiment based on the work of Notoamodjo et al. (2009) to determine if users were more likely to assign stronger passwords to "important" websites (2013). Eighty university students were asked to create passwords for mock representations of popular websites in four categories: financial websites, identity websites (such as facebook.com), content websites, and "sketchy websites" (such as those offering suspect deals) (Haque et al., 2013). Haque et al. found the mean length and complexity of the student passwords to be highest among financial and identity websites (2013). Haque et al. attempted to crack the "higher level" student passwords using John the Ripper with mangling rules and a dictionary file containing four of the "lower level" student passwords (2013). Using this technique, it was possible to crack 19.1% of the students' higher-level passwords (Haque et al, 2013). Using a dictionary file consisting of the lower-level passwords and the Cain & Abel wordlist, Haque et al. reported a 33% success rate (2013). These findings would indicate a high degree of syntactic similarity between the higher and lower-level passwords; it is possible that password modification techniques using substrings, as evidenced in Das et al. 2014 and Wang et al. 2017, were used by some of the students.

Chapter 3

Description of Methodology

This research is a technical-based, descriptive study which seeks to quantify and analyze the breach data of students from eight prominent Michigan universities. The reasons for choosing a descriptive methodology for this work were twofold: password analysis as a research topic has been studied extensively—it is not an emerging field—thus, a broader, more comprehensive exploratory study is not warranted; the primary goal of this research is not to determine causality, but rather to answer specific questions as they relate to a narrow, targeted population (Michigan university students).

This research was inspired predominantly by that of Robinson (2018), Spafford (1992), and Wang, Jan, Hu, and Wang (2017). Wang et al. studied password reuse among online services by aggregating data from a number of data breach sources (2017). Robinson demonstrated the use of Apache Spark for detailed analysis of breach password data (2018). Spafford collected plain-text university passwords in order to analyze password structure and detect password weakness (1992). Components of these three pieces of research form the conceptual basis for this study: this research attempts to leverage breach data from a number of sources to study scope of student breach involvement, and to analyze password creation behavior of university students. Apache Spark was used as to perform much of the technical password analysis and to necessary data mining tasks related to this study.

Research Focus

The research focus of this study is current and former students from the following Michigan-based universities:

- Central Michigan University (CMU)
- Eastern Michigan University (EMU)

- Ferris State University (FSU)
- Michigan State University (MSU)
- Northern Michigan University (NMU)
- Western Michigan University (WMU)
- Wayne State University (WSU)
- University of Michigan (U of M)

Design of Study

This study consists of two phases: data discovery and password analysis. Data discovery focuses on identification, enumeration, and data cleaning of student records and is used to quantify student breach involvement. The password analysis phase focuses on semantic analysis of student passwords and is used to identify password trends.

Breach dataset criteria & selection. Prior to describing the design of the study in detail, it is necessary to explain the criteria used for evaluating breach data for inclusion in the study, and to introduce the structure, format and origin of the breach dataset selected. Any dataset included within the study needed to publicly available, readily accessible, and had to contain email addresses so that records within the scope of the study could be easily identified and attributed to the proper university. To avoid limiting the analysis of password data to records that could be cracked, datasets which included plain-text passwords were sought over datasets wherein the passwords were hashed or otherwise obfuscated. While searching for suitable datasets, a large (forty-one gigabyte, 1.4 billion record) “breach compilation” archive was discovered. The archive was comprised predominately of email address/plain-text password combinations, which were optimal for use in this study. Due to the size and scope of the breach

compilation dataset, it was not necessary to include any additional datasets.

Resources. To accomplish the objectives of this study, the following hardware and software resources were used:

- Hardware:
  - MSI GE63VR 7RF Raider
    - Intel Core i7-7700HQ CPU @ 2.80GHz
    - 32 GB RAM
    - Nvidia GeForce GTX 1070
- Software:
  - Windows 10 Education 64-bit, Version 1803, OS Build 17134.345
  - Ubuntu Linux 64-bit, Version 18.10
  - VMware Workstation 15 Pro, Version 15.0.0, build-10134415
  - Apache Spark Version 2.3.2
  - Anaconda, Version 5.3.0, 64-bit architecture
  - Deluge Version 1.3.15
  - Jupyter Notebook Version 5.6.0
  - Python 3.7.0
  - Plotly 3.4.2

Data discovery. The data discovery phase of the study focused on answering the first research question: "How many student records can be identified within the breach dataset(s) examined?" To accomplish this task, a lab environment was created, relevant data was parsed from the breach compilation dataset, and that data was cleaned and normalized.

Lab environment. Deluge, a bit-torrent client, was used to download the breach

compilation dataset from the Internet using a Windows Ten host. Next, an Ubuntu Linux virtual machine (VM) was created using VMWare. Python 3 was updated to the latest version (3.7.0 at the time of writing) for use with Apache Spark. Apache Spark was installed and configured on the virtual machine. To interact with Apache Spark, the built-in application programming interface for Python, called "PySpark" was used. Python code for Pyspark was executed via Jupyter Notebook, an interactive web-based platform for programming. To facilitate this, Jupyter was installed via the Anaconda distribution of Python data science tools. After the virtual machine was set up and the above software packages were installed, a copy of the breach dataset was transferred to the VM for analysis.

Locating University Email Domains. To properly attribute breach records to each university, it was necessary to know the email domain used by each. A quick open source intelligence (OSINT) investigation was conducted on each of the eight universities of interest to this study. The Google search engine was used to identify the website for each university. Email domains were located on each university website by searching available public contact lists, contact pages, and email help documentation. It was possible to confirm that the email domains found for Ferris State University, Central Michigan University, and Michigan State University were correct by contacting current or former students from these institutions that were personally known by the author.

The following email domains were discovered for each university:

- Central Michigan University: @cmich.edu
- Eastern Michigan University: @emich.edu
- Ferris State University: @ferris.edu
- Michigan State University: @msu.edu

- Northern Michigan University (NMU): @nmu.edu
- Western Michigan University (WMU): @wmich.edu
- Wayne State University (WSU): @wayne.edu
- University of Michigan (U of M): @umich.edu

Sorting using Linux utilities. The breach compilation archive consisted of a several informational folders (containing log files, scripts, or other information), as well as a primary "data" folder where the breach records were stored. Within the data folder, subfolders divided files containing breach records alphabetically by the first character in the email address. Each subfolder contained a data file representing a portion of the archive. For example, a folder named "a" might contain a data file with all email addresses that began with the letter "a". In some cases, letters were further sub-divided by subsequent characters, such as "aa", or "ab".

As no singular file containing all of the university email records existed, it was necessary to search for instances of university email addresses across every file, then aggregate the results to isolate university data. In order to accomplish this, the terminal command "Grep -rih '@emaildomain.edu' > output.txt" was used to recursively search all files and folders within the archive data folder, ignoring character case. The grep command was executed for each email domain, and the results were saved to separate text files for each university. Next, "Grep -vi 'Binary'" was used to remove search headers that were inserted into the output files by the previous grep command. At this point, the datasets for each university were saved in a colon-separated list format, such as "user@email.com:password". To make the files easier to parse automatically within Pyspark, "sed -e 's/:/,/' Universitylist.txt > Universitylist.csv" was used to replace the colon delimiter between email and password with a comma for each row.

Pyspark data cleaning. After the eight files containing university records were created

and converted into a comma separated value (CSV) format using the Linux command line utilities, they were imported one at a time into a Jupyter notebook for data cleaning. Bad data stemming from several common issues was identified in the university datasets. Common filtering rules were written to isolate and exclude bad data using Python and SQL. Each query return was manually reviewed so that good records would not be categorically eliminated. Custom rules were written to address issues unique to an individual university list, when needed. The actual rules used for data cleaning can be found in Appendix A. The most common issues encountered during data cleaning were missing values within the source archive, missing line returns between records, and bad password data. Missing values or line returns in the source file occasionally led to an amalgamation of records on one line consisting of irrelevant data, such as email addresses from multiple, unrelated email domains. Bad password data manifested as hex records, password hashes, and suspicious values. Hex records and password hashes were easily identified by length and character composition. Notably, a small percentage of records in every data set contained a strict fifteen-character pseudo-random numeric string in the password field. These records appeared highly suspicious due to their uniformity and strong contrast with the other password data in each dataset and were removed with other bad data.

Data Analysis. The data analysis phase of the study focused on answering the second research question: "What password creation & reuse trends exist among the breached student passwords?" In order to answer this question, normalized data sets from the data discovery phase were analyzed in a virtual lab environment using Pyspark.

The code and data transformations used within Pyspark for the Data Analysis portion of the study were designed to be applicable to each of the eight university datasets produced by the first phase. A duplicate copy of the Jupyter Notebook used for dataset analysis was created for

each of the eight individual datasets. Each dataset was analyzed sequentially within its own Pyspark instance, and the results observed, noted, and reported in Chapter four of the study. In order to analyze the data for broader trends, a ninth dataset consisting of all data from each of the universities was created. An additional Juypter Notebook instance was created for analysis of this data.

Lab environment. The lab same lab environment consisting of an Ubuntu Linux VM loaded with Apache Spark and the Anaconda data science tool distribution, as described in the previous section, was utilized for the data analysis portion.

Pyspark analysis overview. Data analysis using Pyspark was divided into two sections: Preprocessing and Dataset Analysis. In order to provide a logical layout for the Dataset Analysis section, it was organized further into five sub-sections. Each subsection contains a collection of data transformation code which provides useful analysis based on a general analysis goal. For example, the "Length, Min, & Max" section provides information such as average password length, longest password, and shortest password. Much of the code is designed to run sequentially: earlier transformations provide data which is used in later, more complex transformations. Each subsection will be identified and discussed later in the study. The actual code used for dataset analysis can be found in Appendix B.

Preprocessing. After reviewing the university datasets that were created during phase one of the study, a problem with email address capitalization variance was identified. In some instances, multiple records containing the same email address used differing capitalization schemes, such as all lowercase, all uppercase, or "camel-case". While it is easy for a human to identify emails of varying case as the same source account, Pyspark would categorize these as unique emails. In order to solve this problem, all email addresses were transformed to lowercase

using the Python function “lower()” which returns input strings in lowercase. This is a suitable solution because unlike passwords, email addresses are not case sensitive.

Dataset analysis. The five subsections used for dataset analysis are as follows: “Length, Min & Max”, “Password Recurrence, Reuse by User”, “Calculate Levenshtein Distance per User”, “Calculate Levenshtein Distance Between All Passwords”, and “Regex Transformations to Identify Common Root Words”. The scope and methodology of each section is described in detail below:

Length, min & max. This section used built-in Pyspark functions to calculate the number of unique email addresses, the number of email addresses with more than one record, the average password length within the dataset, as well as the shortest and longest passwords in the dataset. The “count()” function was used to determine the number of rows within the dataset. The “distinct()” function calculated the number of non-recurring records within the dataset. Combining “length()” and “avg()” returned the average length of passwords in the dataset. “Min” and “max” were used to determine the range of password lengths within the sample.

Password recurrence, reuse by user. Queries written in this section calculated the number of reused passwords within the dataset, displayed the most common passwords, and determined users with the greatest number of breached accounts. Additionally, the number of passwords by length and accounts breached grouped by instances per user were calculated and displayed. These transformations were accomplished primarily by migrating the datasets into SQL views, then running SQL queries against the data.

Calculate Levenshtein Distance per user. “Levenshtein Distance” (sometimes called “edit distance) is a measure of the similarity of two strings, in terms of the number of deletions, insertions, or substitutions required to transform the first string into the second string (Gilleland,

n.d). Levenshtein Distance has been widely used in areas of computer science, linguistics, bioinformatics for tasks such as spell checking, plagiarism detection, speech recognition, and DNA analysis (Klein, n.d.; Gilleland, n.d.). The theory and algorithm were originally developed by Russian scientist Vladimir Levenshtein in 1965, hence the name (Gilleland, n.d.).

In order to calculate the average Levenshtein Distance per user, a number of steps were required. First, a list of email addresses and their appearance frequency was created by grouping the primary dataset by email in conjunction with the count function as an aggregator. This list was filtered to only include records where the count was greater than one. Next, an SQL inner join was performed against the primary dataset where email addresses existed in both datasets. This resulted in a sorted list of all users with more than one breached password. The Pyspark aggregation function "collect_list" was used against this sorted list to collect all passwords for a given user as a Python list object. Instead of multiple records for each email containing one password, the result was a single record for each user which contained all of the user's passwords.

While Apache Spark includes a built-in function for calculating Levenshtein Distance, it was intended for use within an aggregation function and thus not independently callable. For this reason, it was necessary to create a custom version in Python which accepts two strings as input and returns a string format number representing the Levenshtein Distance between the two input strings.

To calculate the average Levenshtein Distance for each user, it was necessary to find the Levenshtein Distance between each of the user's passwords, add all values together, then divide by the number of passwords. The formula to calculate the number of unique combinations possible in a given list, where N represents the number of values in the list is available below as

Equation 1.

$$\boldsymbol{N} * \frac{\boldsymbol{(N-1)}}{\boldsymbol{2}} \quad (1)$$

A second Python function was designed which accepts a list object as an input and returns a user's average Levenshtein Distance. The function, named "userLev", used the above formula to generate all possible unique combinations for an input list of *N* length. As combinations were generated, the values were passed the values to the Levenshtein Distance function. The spot Levenshtein Distance was saved to a second list object, which was read off at the end of the function to calculate the user average. The data produced from the calculation of Levenshtein averages was used to create several other datapoints, including an aggregate average Levenshtein Distance for all multi-record users, average Levenshtein Distance for multi-record user by frequency, average Levenshtein Distance by number of passwords and frequency, and aggregate average Levenshtein Distance by number of passwords.

Calculate Levenshtein Distance between all passwords. By calculating the Levenshtein Distance between all passwords within the dataset, the overall homogeneity of the dataset could be measured, and the most highly correlated passwords as measured by low Levenshtein Distance could be determined. The methodology that was used in the previous section was carried over for this set of Levenshtein calculations, however some adaptation was necessary to accommodate the larger scale of the data.

The "collect_list" aggregate function was used to collect every password record in the dataset as a single list object. Much of the code written for the userLev function for calculating unique combinations was reused to calculate unique combinations among the dataset. Due to the sheer number of possible combinations that can be produced using even modest dataset, several modifications were required; a 40,000-record dataset produces nearly 800,000,000

unique combinations! A new function called “uniqueCombos” was defined which accepts a list object as input. To avoid overflowing data objects when working with large data, output was directed to a CSV file within the current working directory rather than saved as a list object, as was done before. Next, the CSV file was imported back into Pyspark as a new dataframe. It was then possible to run calculate Levenshtein Distance within Pyspark using the built-in function. After the Levenshtein Distance for each password combination was calculated, that information was used to find the aggregate average Levenshtein Distance for the entire dataset as previously discussed.

Regex Transformations to Identify Common Root Words. This section expanded on the concept of identifying common or homogenous strings within a dataset. By using the Pyspark “regexp_extract” function in concert with the “lower” function, it was possible to remove all instances of numbers and special characters while converting all alphabetic values to lowercase. This technique was applied to the passwords aggregated by count of Levenshtein values transformation in order to produce a list of root words among the most highly correlated passwords. The technique was also applied to the list of passwords sorted by frequency generated within the “Password recurrence, reuse by user” section. This transformation resulted in the most common root words within the dataset.

Chapter 4

How many student records can be identified within the source breach dataset?

A total of 144,083 breach records representing the eight institutions of interest to this study were identified within the source dataset. Analysis of the data collected revealed a total of 121,215 unique university email addresses from the above-referenced institutions. See Table 1 below for a breakdown of records recovered for each university. Detailed statistics for each institution can be found in the dedicated sections below.

Table 1

Breach Records Recovered by Institution

Institution	Total Records Recovered	Unique Email Addresses
Central Michigan University	12755	10822
Eastern Michigan University	13584	11409
Ferris State University	1906	1694
Michigan State University	43678	35527
Northern Michigan University	6688	5689
Western Michigan University	11511	10079
Wayne State University	13932	11866
University of Michigan	40029	34129

What password creation & reuse trends exist among the breached student passwords?

Breach Exposure. More than one breach record was found for 18,551 email accounts, or approximately 21% of the population of breached accounts. A majority (roughly 70%) of

accounts with more than one record had been breached twice. Table 2 summarizes the number of multi-breach accounts per institution.

Table 2

Accounts Breached Multiple Times by Institution

Institution	Breached Twice	Breached > 2 Times
Central Michigan University	1339	248
Eastern Michigan University	1511	283
Ferris State University	144	68
Michigan State University	5387	1,194
Northern Michigan University	714	124
Western Michigan University	1025	174
Wayne State University	1393	274
University of Michigan	3965	754

Average Password Length & Range. The average password length for all university passwords was 8.21 characters. Overall, passwords ranged from one character to 29 characters in length. The frequency of 8-character passwords within the data collected was nearly double that of any other password length (See Figure 1). Approximately 91% of passwords collected were

between 6-10 characters in length.

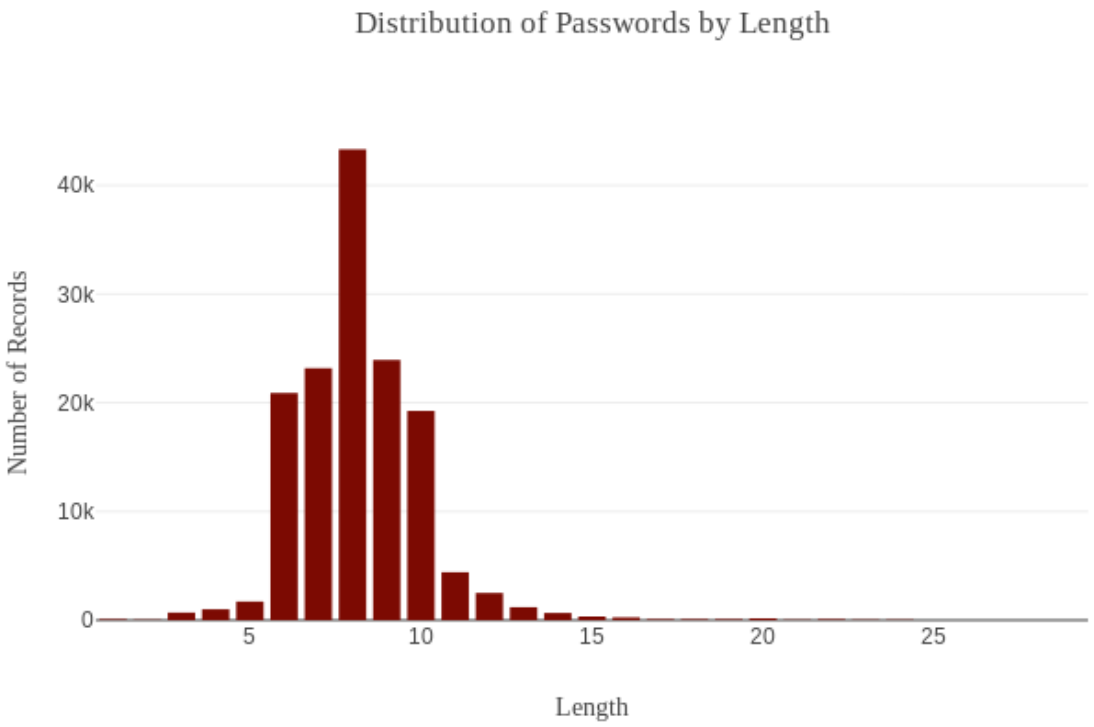

Figure 1. Distribution of passwords by length for all recovered passwords.

Password Reuse. Password reuse was measured among the 18,551 accounts where more than one password was available. 1,857 (10%) of users were found to have reused the same password twice, while only 36 users (< 1 %) were found to have reused the same password three times. Among the eight institutions studied, zero instances of password reuse greater than three times were recorded.

Password Similarity. The average Levenshtein Distance among passwords was calculated for each multi-record user. This data was used to plot the Levenshtein Distance average by number of passwords (See Figure 2). 6,113 of the 18,551 of multi-record accounts (33%) averaged a distance of three or less (See Figure 3). The Levenshtein Distance for each unique password combination within the dataset was calculated, then averaged. For each dataset,

the average was approximately 8.

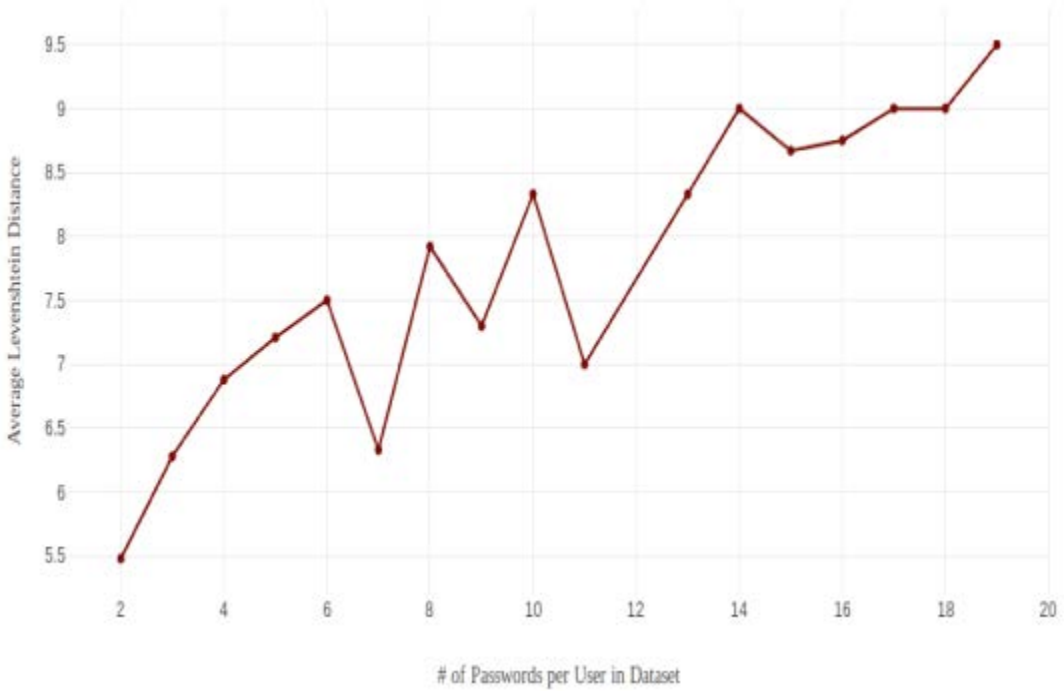

Figure 2. Average Levenshtein Distance by count of user passwords for all recovered data.

No_of_PWDS	Average_Lev_Dist	Count
2	0.0	1439
2	1.0	2450
2	2.0	1175
2	3.0	613
3	0.0	28
3	1.0	169
3	2.0	90
3	3.0	95
4	1.0	22
4	2.0	12
4	3.0	11
5	1.0	2
5	2.0	1
5	3.0	2
6	2.0	1
9	3.0	3

Figure 3. Count of Average Levenshtein Distance less than 4 by number of passwords per user.

Common Passwords. The most common passwords and the most common root password strings were calculated for the entire dataset. The single most common password was used 205 times across all eight university datasets, whereas the most common root string appeared 570 times (See Figures 4 & 5). The ten most common passwords accounted for seven-tenths of one percent of recovered passwords, whereas the ten most common root strings were present in approximately two percent of records.

Count	PWD
205	password
191	password1
121	123456
112	michigan
87	myspace1
81	goblue
75	abc123
74	cme2012
64	michigan1
63	football

Figure 4. Top ten passwords across all datasets.

rootPWD	Appearances
password	570
soccer	349
michigan	349
hockey	272
goblue	262
football	233
spartan	223
myspace	195
love	193
july	189

Figure 5. Top ten root strings across all datasets.

Central Michigan University

Number of records. After data cleaning, 12,755 email/password records were extracted the source dataset corresponding to 10,822 distinct “cmich.edu” email addresses. 1,587 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users.

Breach exposure. The number of passwords recovered for each user ranged from one to fifteen. In other words, within the CMU dataset, a single email could be associated with as few as one, or as many as fifteen breach records (See Figure 6).

TimesBreached	Count
1	9235
2	1339
3	202
4	31
5	4
6	3
7	1
8	4
10	2
15	1

Figure 6. CMU users by number of times breached.

Figure 7 represents the frequency of users by the number of times breached observed within the university dataset.

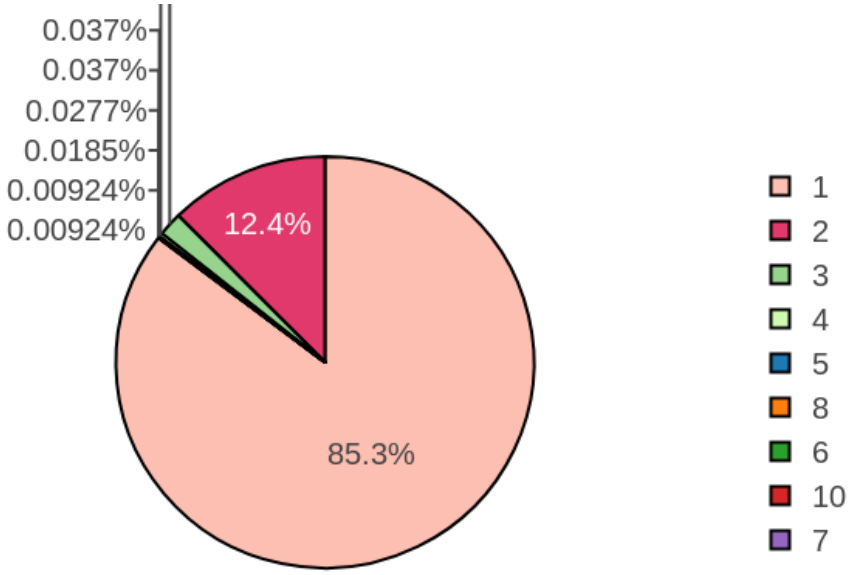

Figure 8. Pie Chart: CMU users by times breached.

Length, min & max. Passwords from the CMU dataset ranged from one character to 27 characters in length; the average length of passwords within the dataset was found to be 8.35 characters. A majority of passwords (92 percent) ranged from six to ten characters (See Figures 8 & 9).

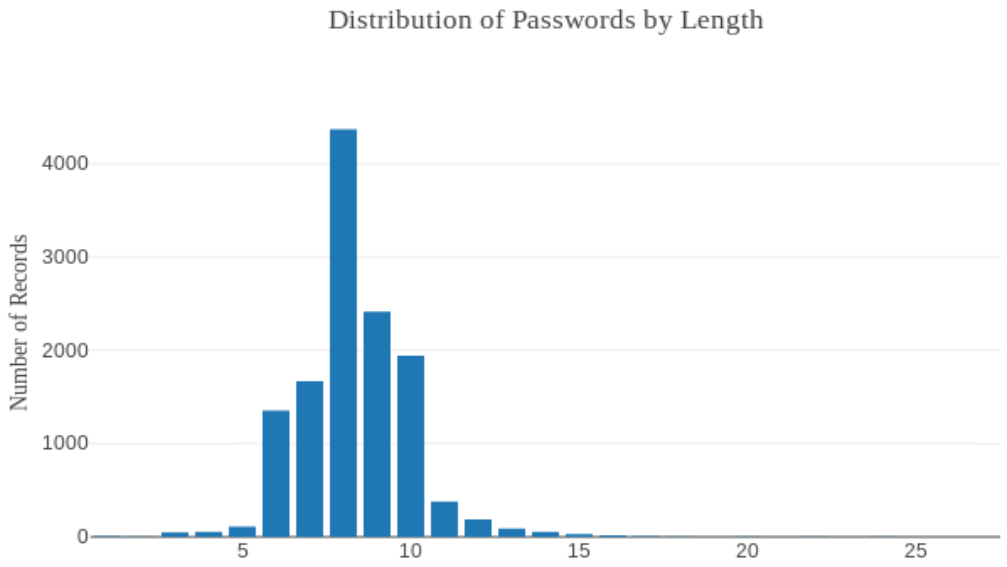

Figure 7. Bar Chart: CMU passwords by length.

PWDLength	#_of_Passwords
1	9
2	5
3	48
4	54
5	110
6	1353
7	1667
8	4368
9	2413
10	1941
11	378
12	187
13	89
14	54
15	30
16	16
17	9
18	6
19	2
20	3
21	2
22	3
23	1
24	3
25	2
27	2

Figure 9. CMU passwords by length.

Password recurrence, reuse by user. 2,787 passwords—approximately 21% of passwords observed— appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 10. For users with more than one breached password, password reuse was measured. Of the 1,587 users with multiple breached passwords, 317 (20%) were found to have reused the same password twice, while 4 (0.2%) reused the same password three times.

PWD	Count
password1	33
central1	29
football	17
password	15
abc123	12
michigan	11
123456	10
baseball	10
baseball1	9
central10	8

Figure 10. Most common passwords within the CMU dataset.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user's passwords was calculated. 588 users (37%) averaged a distance of 3 or less, indicating passwords that vary within three characters (Figures 11 & 12).

Average_Lev_Dist	Count
0.0	252
1.0	193
2.0	94
3.0	48
4.0	45
5.0	68
6.0	109
7.0	186
8.0	248
9.0	185
10.0	103
11.0	20
12.0	14
13.0	11
14.0	2
15.0	5
16.0	1
17.0	1
18.0	1
22.0	1

Figure 11. Distribution of Levenshtein Distance averages for CMU.

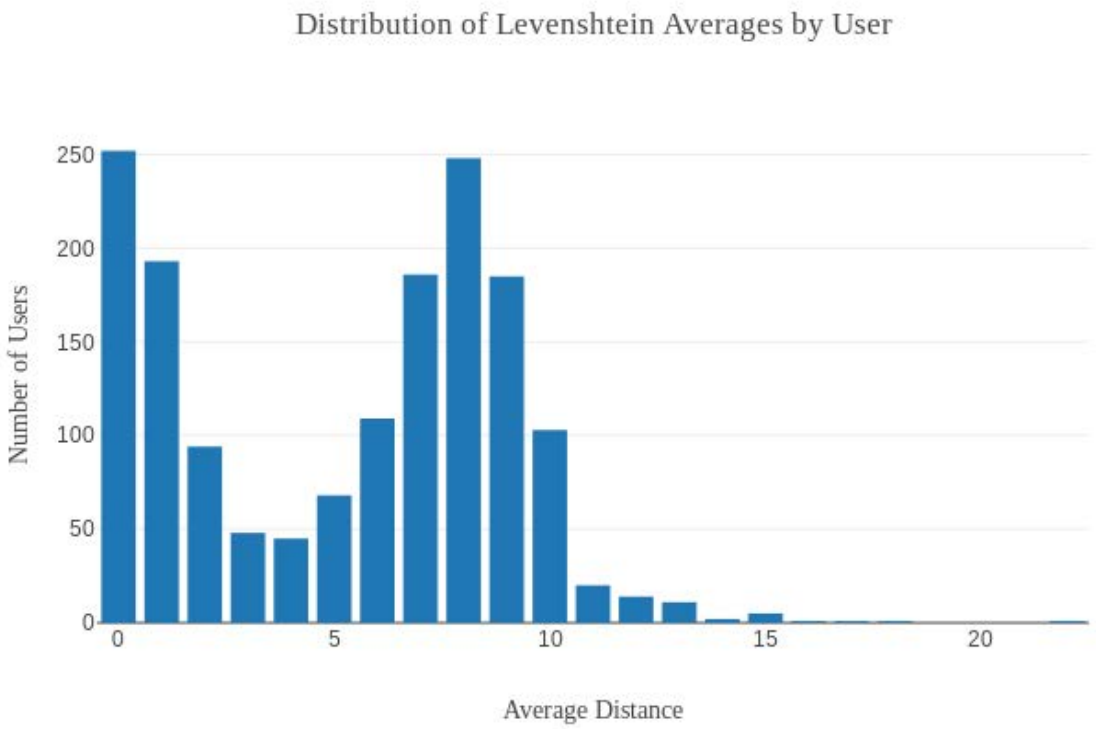

Figure 12. Bar Graph: Distribution of Levenshtein Distance averages for CMU.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 13 shows the most common root strings within the CMU dataset.

rootPWD	Appearances
central	139
soccer	69
password	66
football	58
baseball	42
hockey	29
michigan	23
softball	21
march	19
lucky	17

Figure 13. Most frequent password strings within the CMU dataset.

Eastern Michigan University

Number of records. After data cleaning, 13,584 email/password records were extracted the source dataset. These records corresponded to 11,409 distinct “emich.edu” email addresses. 1,794 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users.

Breach exposure. The number of passwords recovered for each user ranged from one to sixteen. In other words, within the EMU dataset, a single email could be associated with as few as one, or as many as sixteen breach records. Approximately 84% of users discovered within the EMU dataset had been breached a single time, while 13% had been breached twice (Figures 14 & 15).

TimesBreached	Count
1	9615
2	1511
3	229
4	38
5	8
6	2
7	3
8	1
11	1
16	1

Figure 14. EMU users by number of times breached.

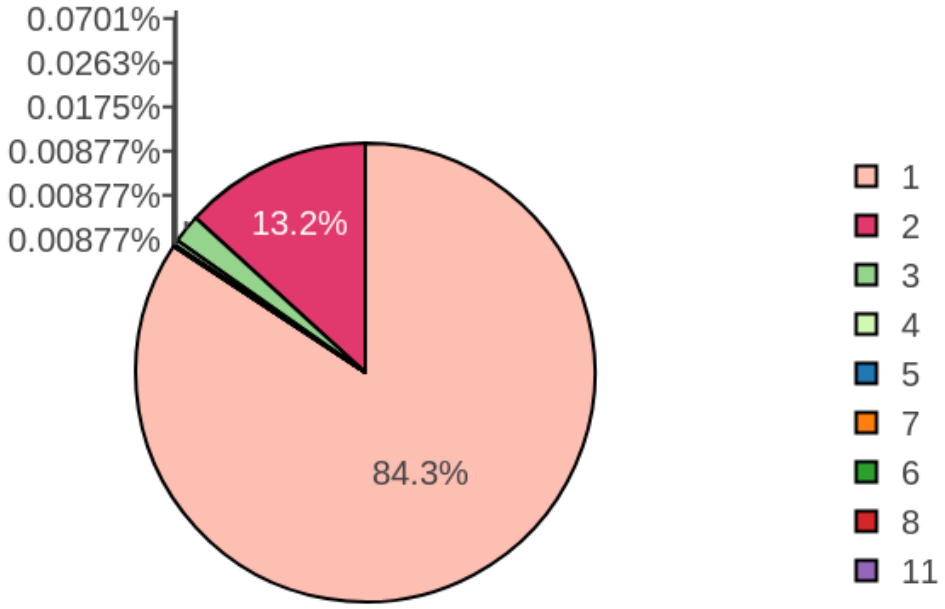

Figure 15. Pie Chart: EMU users by times breached.

Length, min & max. Passwords from the EMU dataset ranged from one character to 28 characters in length; the average length of passwords within the dataset was found to be 8.23 characters. A majority of passwords (91 percent) ranged from six to ten characters (Figures 16 & 17).

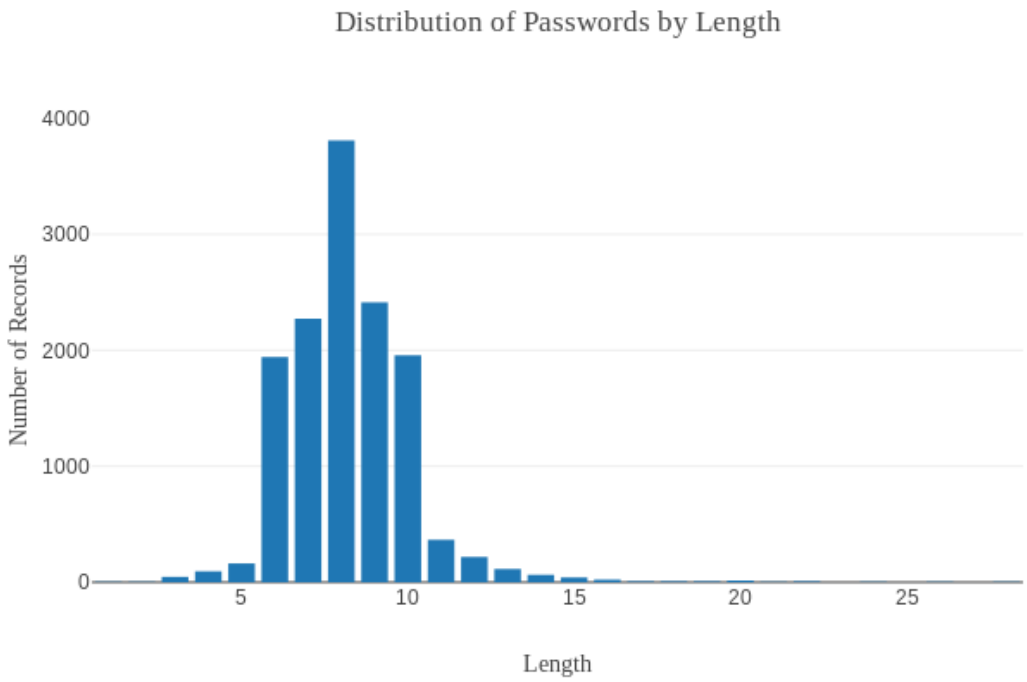

Figure 17. Bar Chart: EMU passwords by length.

PWDLength	#_of_Passwords
1	3
2	2
3	46
4	94
5	162
6	1941
7	2273
8	3809
9	2414
10	1958
11	366
12	218
13	114
14	64
15	41
16	22
17	8
18	8
19	6
20	15
21	3
22	6
23	1
24	4
25	1
26	2
27	1
28	2

Figure 16. EMU passwords by length.

Password recurrence, reuse by user. 2,787 passwords—approximately 21% of passwords observed— appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 18. For users with more than one breached password, password reuse was measured. Of the 1,794 users with multiple breached passwords, 358 (20%) were found to have reused the same password twice. Seven users reused the same password three times.

PWD	Count
password1	24
password	18
careers	13
myspace1	13
abc123	10
eastern	9
football	8
123456	7
fuckyou1	7
soccer	6

Figure 18. Most common passwords within the EMU dataset.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user with more than one record was calculated. 631 users (35%) averaged a distance of 3 or less (See Figures 19 & 21).

Average_Lev_Dist	Count
0.0	288
1.0	176
2.0	108
3.0	59
4.0	51
5.0	85
6.0	139
7.0	213
8.0	258
9.0	193
10.0	113
11.0	50
12.0	24
13.0	12
14.0	8
15.0	4
16.0	7
19.0	6

Figure 19. Distribution of Levenshtein Distance averages for EMU.

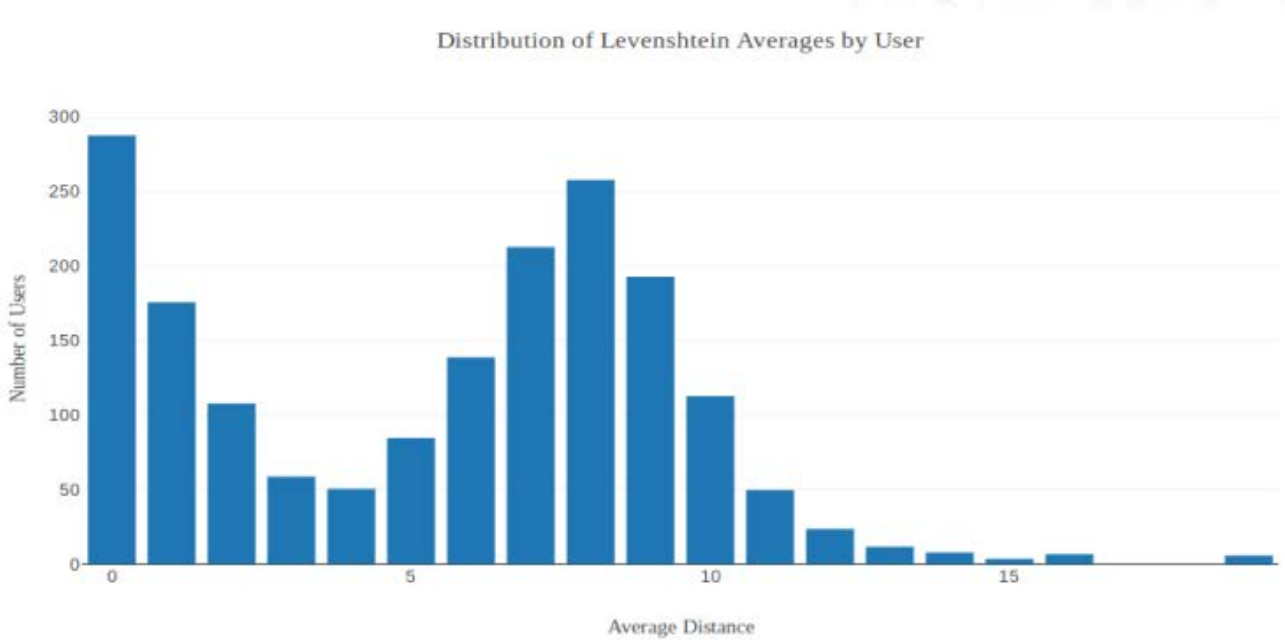

Figure 20. Bar Graph: Distribution of Levenshtein Distance averages for EMU.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 20 shows the most common root strings within the EMU dataset.

rootPWD	Appearances
password	58
eastern	51
love	37
hockey	28
july	28
nicole	24
myspace	24
detroit	23
football	22
chris	21

Figure 21. Most frequent password strings within the EMU dataset.

Ferris State University

Number of records. After data cleaning, 1,906 email/password records were extracted the source dataset. These records corresponded to 1,694 distinct “ferris.edu” email addresses. 212 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users

Breach exposure. The number of passwords recovered for each user ranged from one to sixteen. In other words, within the FSU dataset, a single email could be associated with as few as one, or as many as fourteen breach records. See Figures 22 & 23 for detailed information about user breach frequency.

TimesBreached	Count
1	1528
2	144
3	18
6	2
10	1
14	1

Figure 22. FSU users by number of times breached.

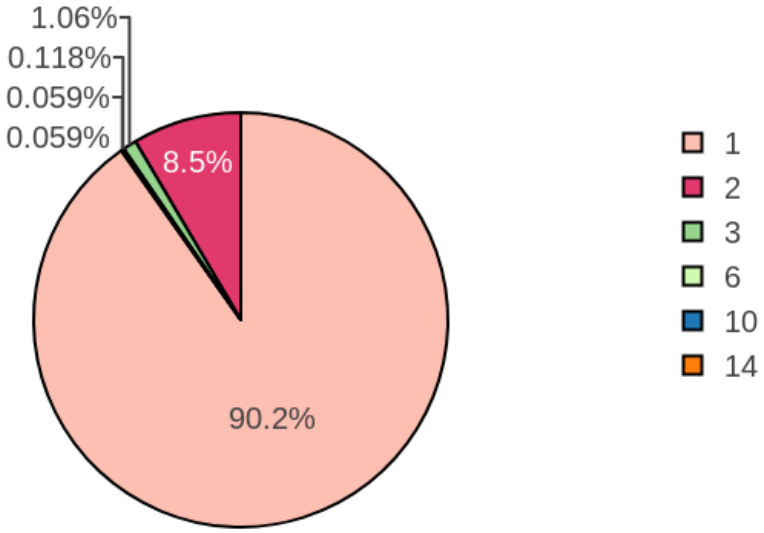

Figure 23. Pie Chart: FSU users by times breached.

Password recurrence, reuse by user. 100 passwords appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 24. For users with more than one breached password, password reuse was measured. Of the 212 users with multiple breached passwords, 35 (17%) were found to have reused the same password twice. Two users reused the same password three times.

PWD	Count
123456	6
bear114a	4
password	4
12345	4
fg767xrc	4
kathleen1	3
midnight1	3
bulldog	3
fbaker5270	3
jordan05	3

Figure 24. Most common passwords within the FSU dataset.

Length, min & max. Passwords from the Ferris dataset ranged from one character to fourteen characters in length; the average length of passwords within the dataset was found to be 8.25 characters. A majority of passwords (95 percent) ranged from six to ten characters (See Figures 25 & 26).

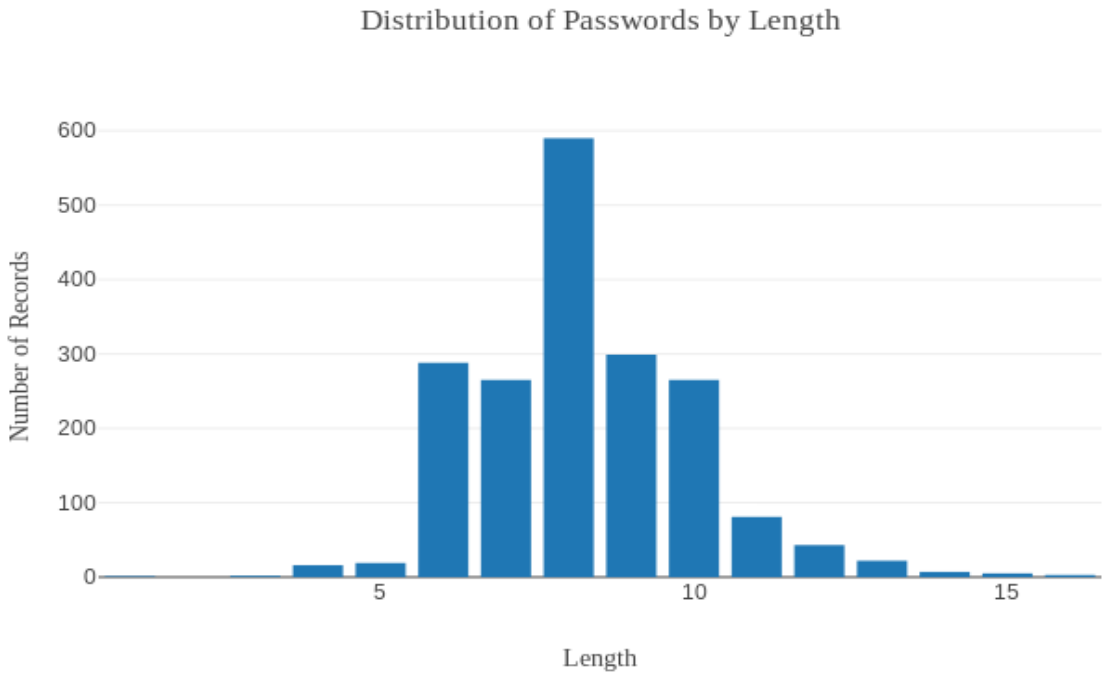

Figure 25. Bar Chart: FSU passwords by length.

PWDLength	#_of_Passwords
1	1
3	2
4	16
5	19
6	288
7	265
8	590
9	299
10	265
11	81
12	43
13	22
14	7
15	5
16	3

Figure 26. FSU passwords by length.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user with more than one password was calculated. 68 users (41%) averaged a distance of 3 or less, indicating passwords that vary within three characters (Figures 29 & 30).

Average_Lev_Dist	Count
0.0	31
1.0	19
2.0	11
3.0	7
4.0	6
5.0	6
6.0	11
7.0	17
8.0	21
9.0	13
10.0	14
11.0	6
12.0	4

Figure 27. Distribution of Levenshtein Distance averages for FSU.

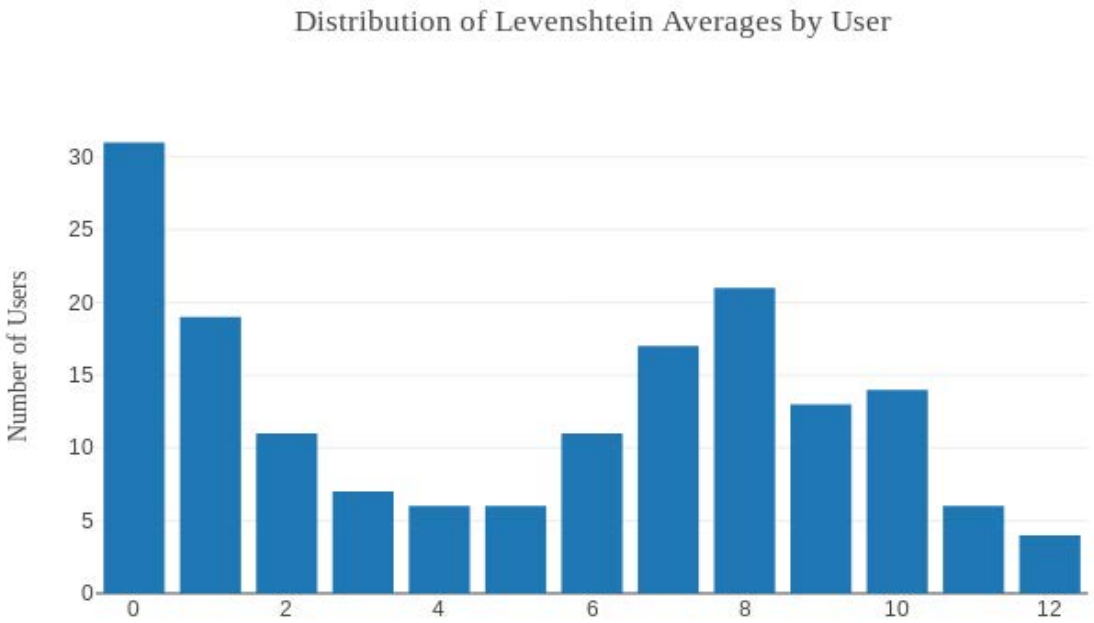

Figure 28. Bar Graph: Distribution of Levenshtein Distance averages for FSU.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 29 shows the most common root strings within the FSU dataset.

rootPWD	Appearances
ferris	28
bulldog	17
football	10
password	6
hockey	6
jordan	5
michigan	5
fbobh_	5
summer	4
sunshine	4

Figure 29. Most frequent password strings within the FSU dataset.

Michigan State University

Number of records. After data cleaning, 43,678 email/password records were extracted from the source dataset. These records corresponded to 35,527 distinct “msu.edu” email addresses. 6,581 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users.

Breach exposure. The number of passwords recovered for each user ranged from one to eighteen. In other words, within the MSU dataset, a single email could be associated with as few as one, or as many as eighteen breach records. Approximately 81.5% of users discovered within the MSU dataset had been breached a single time, 15.2% had been breached twice, and 2.7% had been breached three times (See Figures 30 & 31).

TimesBreached	Count
1	28946
2	5387
3	953
4	177
5	35
6	14
7	6
8	4
9	3
13	1
18	1

Figure 30. MSU users by number of times breached.

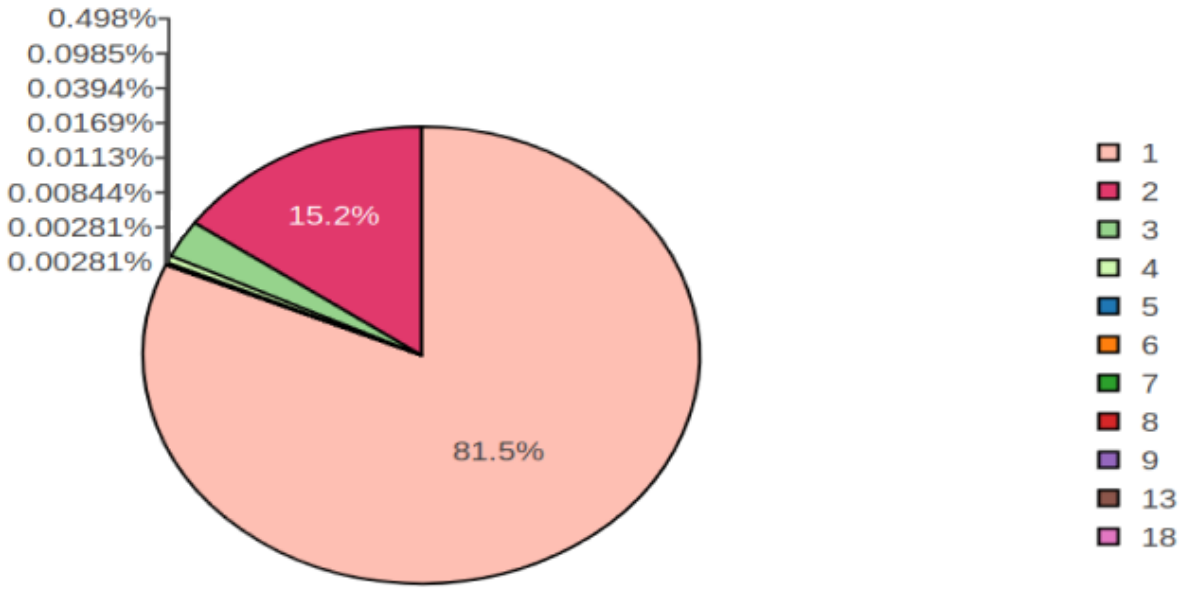

Figure 31. Pie Chart: MSU users by times breached.

Length, min & max. Passwords from the Michigan State dataset ranged from one character to twenty-nine characters in length; the average length of passwords within the dataset was 8.3 characters. Ninety percent of passwords within the MSU dataset ranged from six to ten characters (Figures 32 & 33).

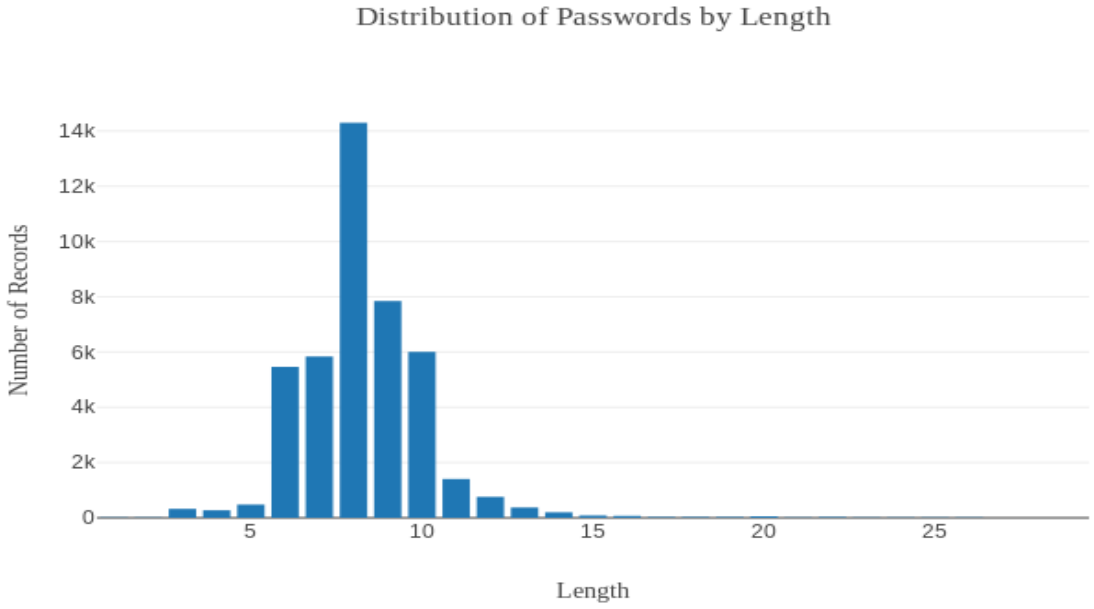

Figure 32. Bar Chart: MSU passwords by length.

PWDLength	#_of_Passwords
1	9
2	9
3	323
4	273
5	479
6	5462
7	5843
8	14297
9	7845
10	6011
11	1405
12	758
13	373
14	199
15	90
16	68
17	31
18	26
19	24
20	50
21	20
22	32
23	17
24	9
25	7
26	7
27	4
28	6
29	1

Figure 33. MSU passwords by length.

Password recurrence, reuse by user. 3,870 passwords appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 34. For users with more than one breached password, password reuse was measured. Of the 6,581 users with multiple breached passwords, 342 (21%) were found to have reused the same password twice. Three MSU users reused the same password three times.

PWD	Count
password	67
cme2012	65
password1	56
spartans	38
123456	34
spartan1	31
spartan	31
spartans1	29
soccer	24
michigan	23

Figure 34. Most common passwords within the MSU dataset.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user with more than one password was calculated. 2,145 users (36%) averaged a distance of 3 or less, indicating passwords that vary within three characters (See Figures 35 & 36).

Average_Lev_Dist	Count
0.0	232
1.0	1145
2.0	473
3.0	295
4.0	270
5.0	386
6.0	592
7.0	815
8.0	972
9.0	679
10.0	387
11.0	131
12.0	85
13.0	39
14.0	28
15.0	13
16.0	13
17.0	9
18.0	6
19.0	5
20.0	3
23.0	1
24.0	1
25.0	1

Figure 35. Distribution of Levenshtein Distance averages for MSU.

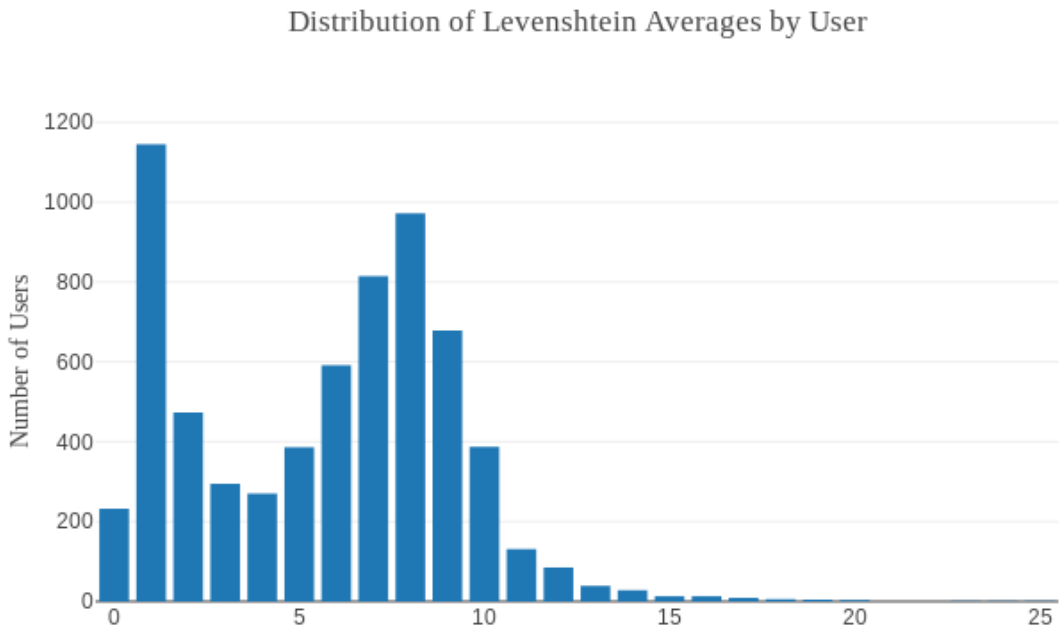

Figure 36. Bar Graph: Distribution of Levenshtein Distance averages for MSU.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 37 shows the most common root strings within the MSU dataset.

rootPWD	Appearances
spartan	213
password	178
soccer	143
spartans	130
sparty	121
hockey	91
michigan	81
state	63
football	62
july	60

Figure 37. Most frequent password strings within the MSU dataset

Northern Michigan University

Number of records. After data cleaning, 6,688 email/password records were extracted from the source dataset. These records corresponded to 5,689 distinct “nmu.edu” email addresses. 838 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users.

Breach exposure. The number of passwords recovered for each user ranged from one to eight. In other words, within the NMU dataset, a single email could be associated with as few as one, or as many as eight breach records. Approximately 85.3% of users discovered within the NMU dataset had been breached a single time, 12.6% had been breached twice (Figures 38 & 39).

TimesBreached	Count
1	4851
2	714
3	94
4	26
5	3
8	1

Figure 38. NMU users by number of times breached.

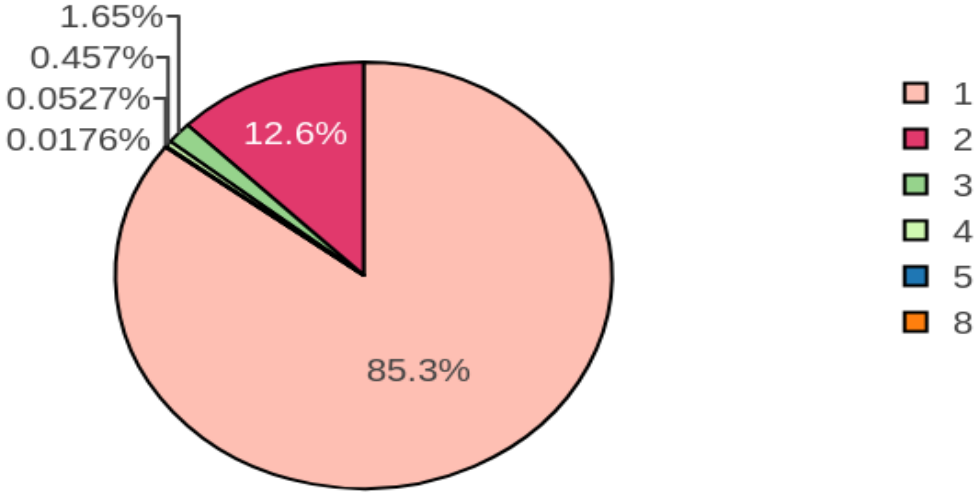

Figure 39. Pie Chart: NMU users by times breached.

Length, min & max. Passwords from the NMU dataset ranged from one character to twenty-two characters in length; the average length of passwords within the dataset was 8.1 characters. Ninety-three percent of passwords within the NMU dataset ranged from six to ten characters (Figures 40 & 41).

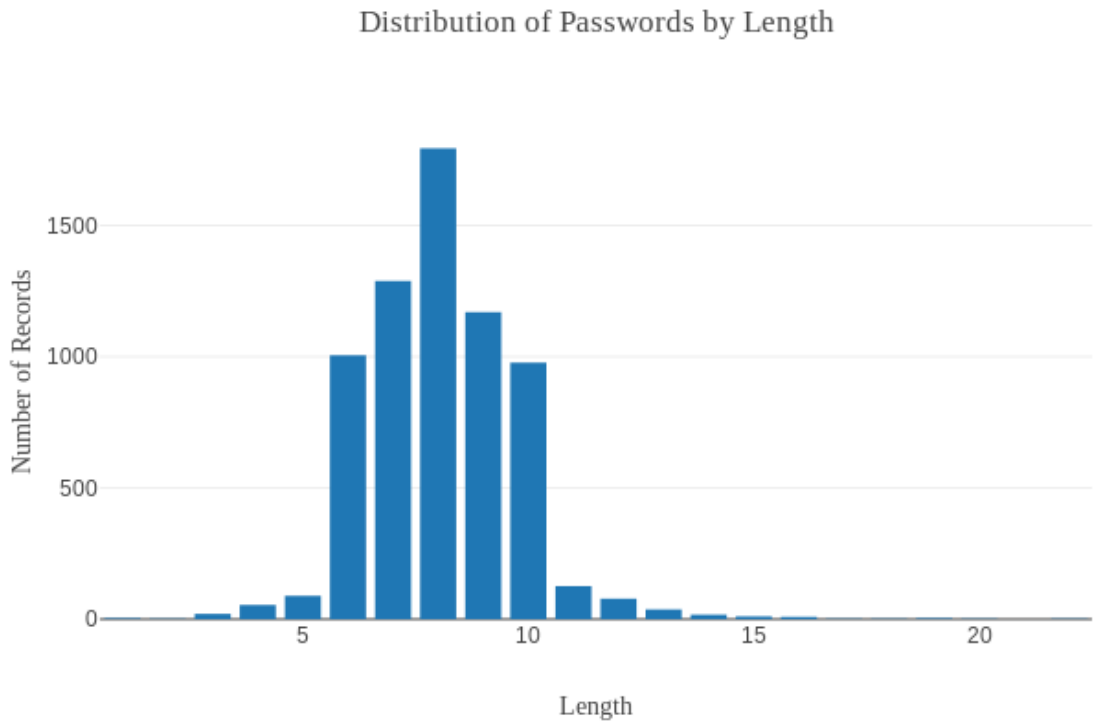

Figure 40. Bar Chart: NMU passwords by length.

PWDLength	#_of_Passwords
1	3
2	2
3	20
4	54
5	89
6	1006
7	1289
8	1794
9	1169
10	977
11	126
12	78
13	37
14	17
15	10
16	8
17	2
18	2
19	3
20	1
22	1

Figure 41. NMU passwords by length.

Password recurrence, reuse by user. 335 passwords appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 42. For users with more than one breached password, password reuse was measured. Of the 838 users with multiple breached passwords, 104 (12.4%) were found to have reused the same password twice. Within this sample, no instances of password reuse greater than two times was found.

Count	PWD
15	password1
8	password
7	myspace1
5	michigan1
5	123456
4	hereford
4	football1
4	qwerty
4	football
3	northern

Figure 42. Most common passwords within the NMU dataset.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user with more than one password was calculated. 277 users (17.5%) averaged a distance of 3 or less, indicating passwords that vary within three characters (Figures 43 & 44).

Average_Lev_Dist	Count
0.0	88
1.0	102
2.0	55
3.0	32
4.0	22
5.0	42
6.0	82
7.0	96
8.0	125
9.0	112
10.0	48
11.0	16
12.0	7
13.0	8
14.0	1
15.0	1
16.0	1

Figure 43. Distribution of Levenshtein Distance Averages for NMU.

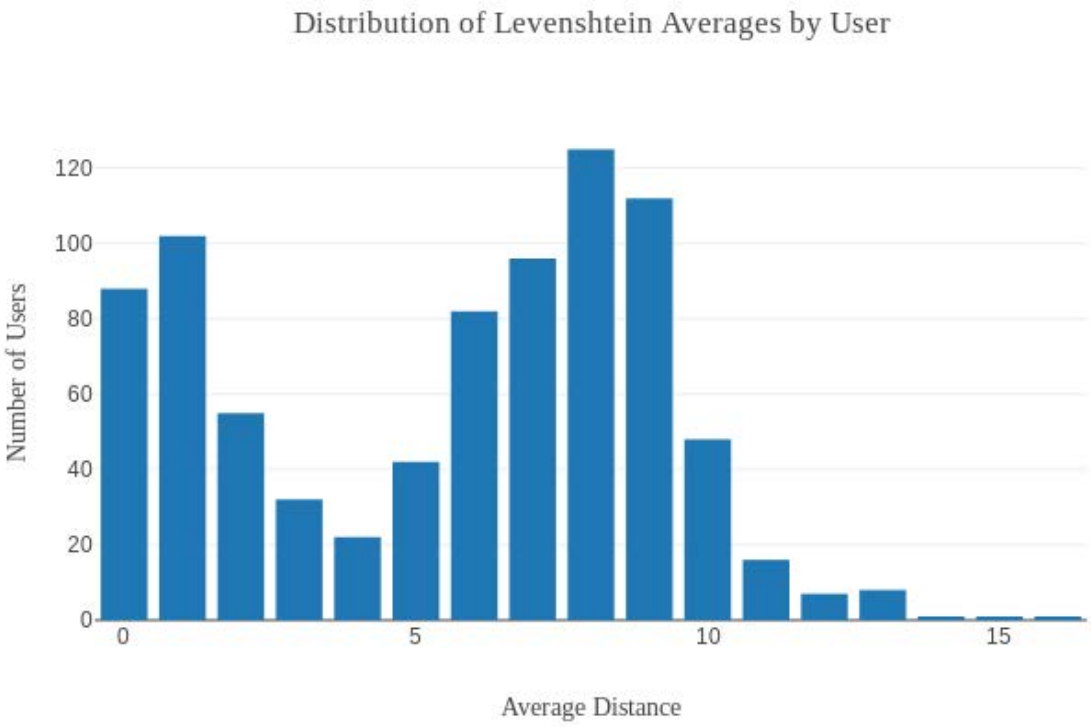

Figure 44. Bar Graph: Distribution of Levenshtein Distance averages for NMU.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 45 shows the most common root strings within the NMU dataset.

rootPWD	Appearances
password	36
hockey	30
football	22
soccer	18
northern	16
table	13
wildcat	12
hunter	12
march	12
glass	11

Figure 45. Most frequent password strings within the NMU dataset.

Western Michigan University

Number of records. After data cleaning, 11,511 email/password records were extracted from the source dataset. These records corresponded to 10,079 distinct “wmich.edu” email addresses. 1,199 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users.

Breach exposure. The number of passwords recovered for each user ranged from one to fifteen. In other words, within the WMU dataset, a single email could be associated with as few as one, or as many as fifteen breach records. Approximately 88% of users discovered within the WMU dataset had been breached a single time, whereas 10% had been breached twice (Figures 47 & 47).

TimesBreached	Count
1	8880
2	1025
3	140
4	28
5	3
6	1
13	1
15	1

Figure 46. WMU users by number of times breached.

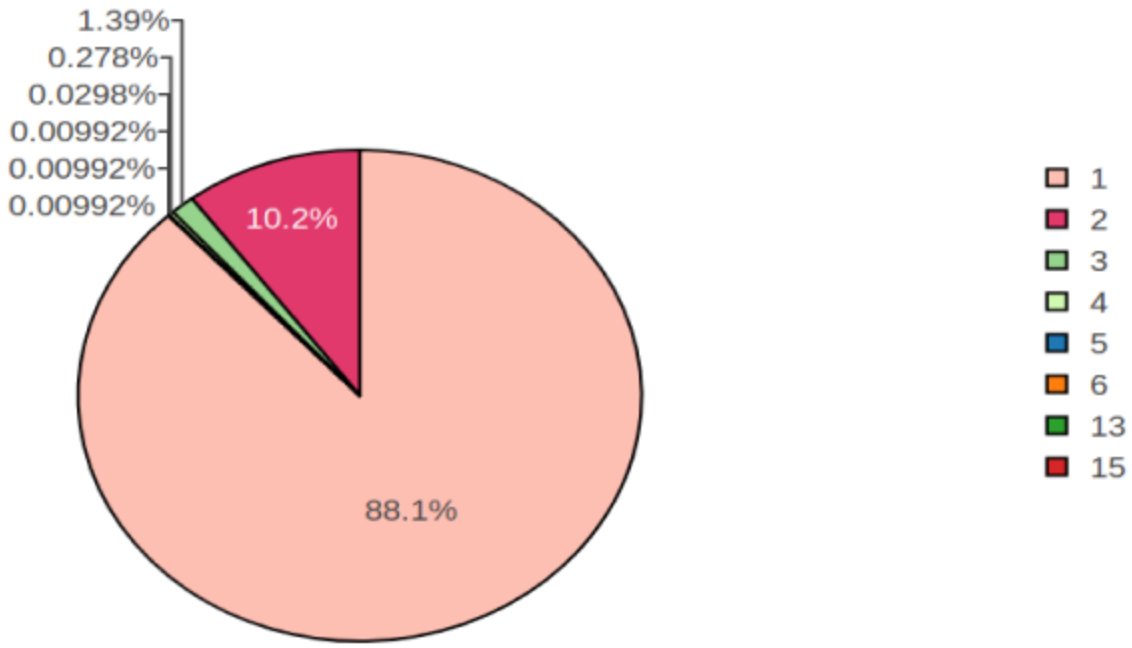

Figure 47. Pie Chart: WMU users by times breached.

Length, min & max. Passwords from the WMU dataset ranged from one character to twenty-nine characters in length; the average length of passwords within the dataset was 8.1 characters. Ninety-two percent of passwords within the WMU dataset ranged from six to ten characters (Figures 48 & 49) .

PWDLength	#_of_Passwords
1	6
2	5
3	29
4	63
5	108
6	1616
7	2156
8	3708
9	1752
10	1415
11	270
12	152
13	82
14	55
15	23
16	24
17	12
18	7
19	4
20	8
21	3
22	2
23	2
24	4
25	1
26	1
29	3

Figure 48. WMU passwords by length.

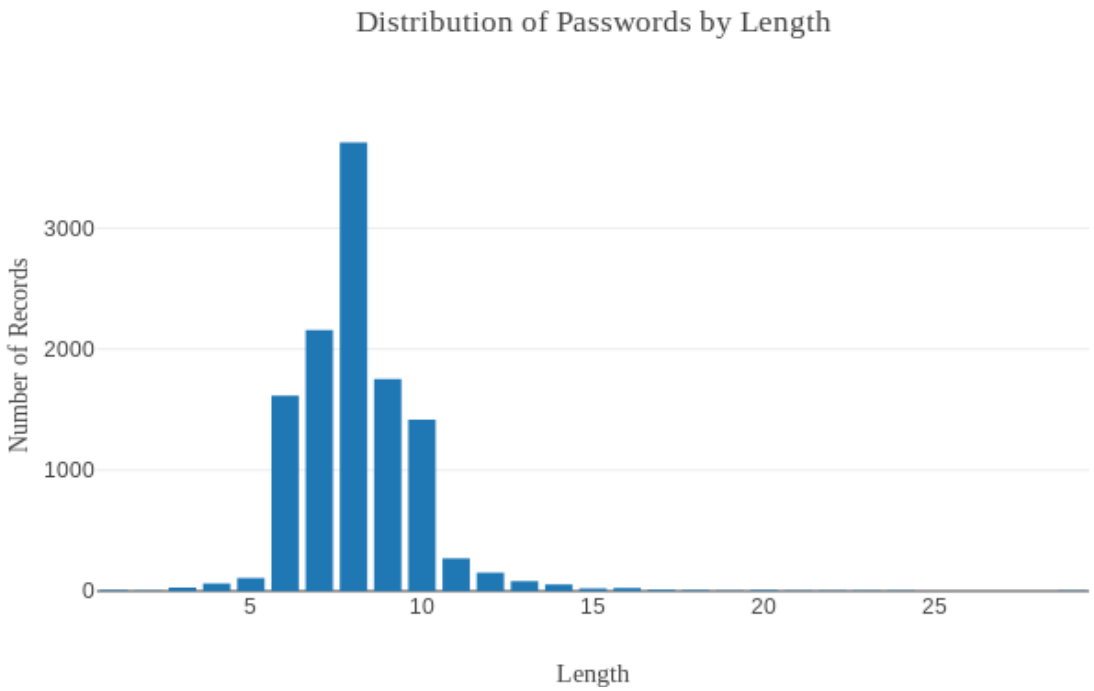

Figure 49. Bar Chart: WMU passwords by length.

Password recurrence, reuse by user. 872 passwords appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 50. For users with more than one breached password, password reuse was measured. Of the 1,199 users with multiple breached passwords, 249 (20.7%) were found to have reused the same password twice, while five users reused the same password three times.

Count	PWD
24	password1
19	western1
15	wmunrhh
14	password
12	western
11	broncos
8	123456
8	myspace1
7	abc123
7	hockey

Figure 50. WMU passwords by length.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user with more than one password was calculated. 455 users (37%) averaged a distance of 3 or less, indicating passwords that vary within three characters (Figures 51 & 52).

Average_Lev_Dist	Count
0.0	214
1.0	133
2.0	67
3.0	31
4.0	29
5.0	69
6.0	76
7.0	139
8.0	179
9.0	115
10.0	68
11.0	27
12.0	19
13.0	12
14.0	6
15.0	3
16.0	3
17.0	3
18.0	1
19.0	1
21.0	2
23.0	1
24.0	1

Figure 51. Distribution of Levenshtein Distance averages for WMU.

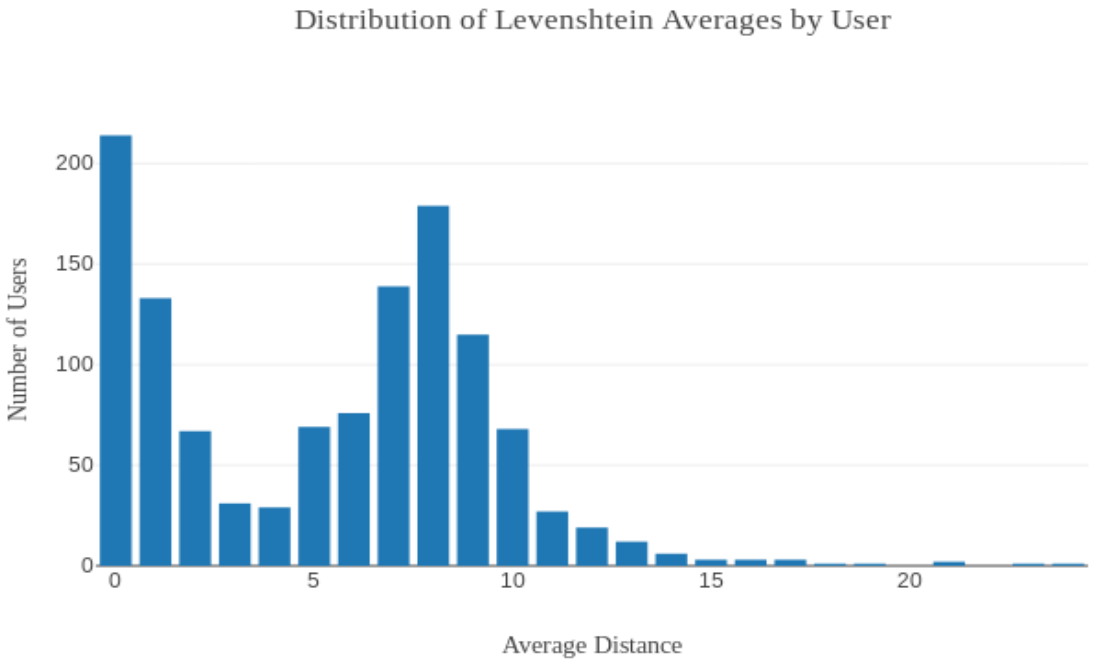

Figure 52. Bar Graph: Distribution of Levenshtein Distance averages for WMU.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 53 shows the most common root strings within the WMU dataset.

rootPWD	Appearances
western	76
password	53
broncos	32
hockey	32
soccer	27
bronco	26
baseball	21
myspace	21
july	21
football	19

Figure 53. Most frequent password strings within the WMU dataset.

Wayne State University

Number of records. After data cleaning, 13,932 email/password records were extracted from the source dataset. These records corresponded to 11,866 distinct "wayne.edu" email addresses. 1,667 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users.

Breach exposure. The number of passwords recovered for each user ranged from one to nineteen. In other words, within the Wayne State dataset, a single email could be associated with as few as one, or as many as nineteen breach records. Approximately 86% of users discovered within the Wayne State dataset had been breached a single time, whereas 12% had been breached twice (Figures 54 & 55).

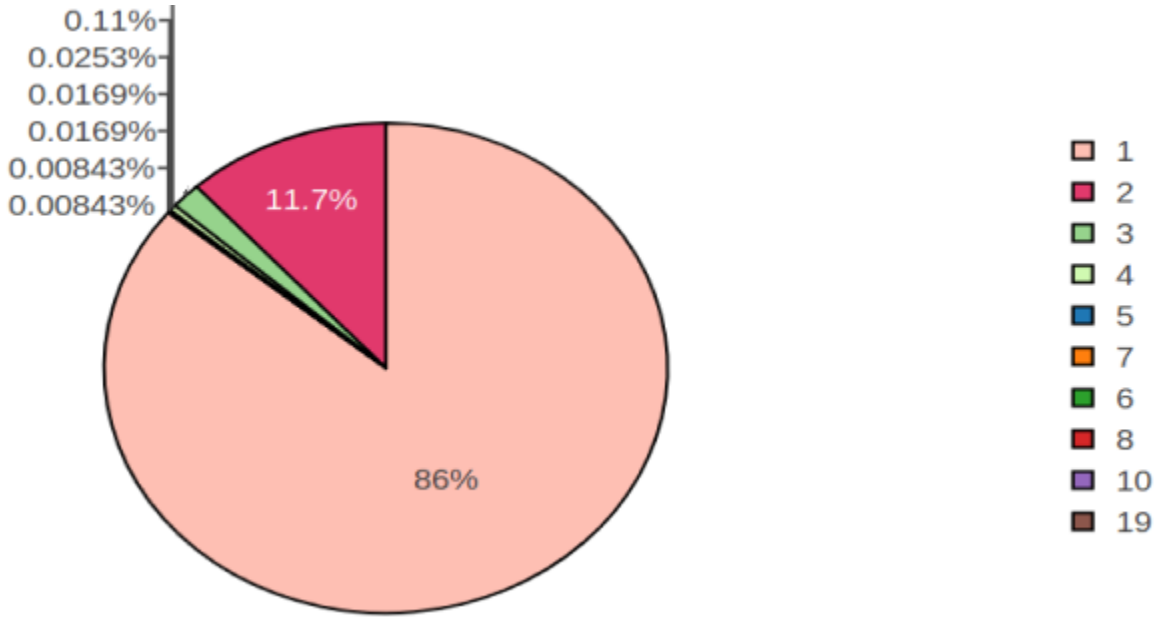

Figure 54. Pie Chart: Wayne State users by times breached.

TimesBreached	Count
1	10199
2	1393
3	204
4	48
5	13
6	2
7	3
8	2
10	1
19	1

Figure 55. Wayne State users by number of times breached.

Length, min & max. Passwords from the Wayne State dataset ranged from one character to twenty-eight characters in length; the average length of passwords within the dataset was 7.99 characters. Ninety-two percent of passwords within the Wayne State dataset ranged from six to ten characters (Figures 56 & 57).

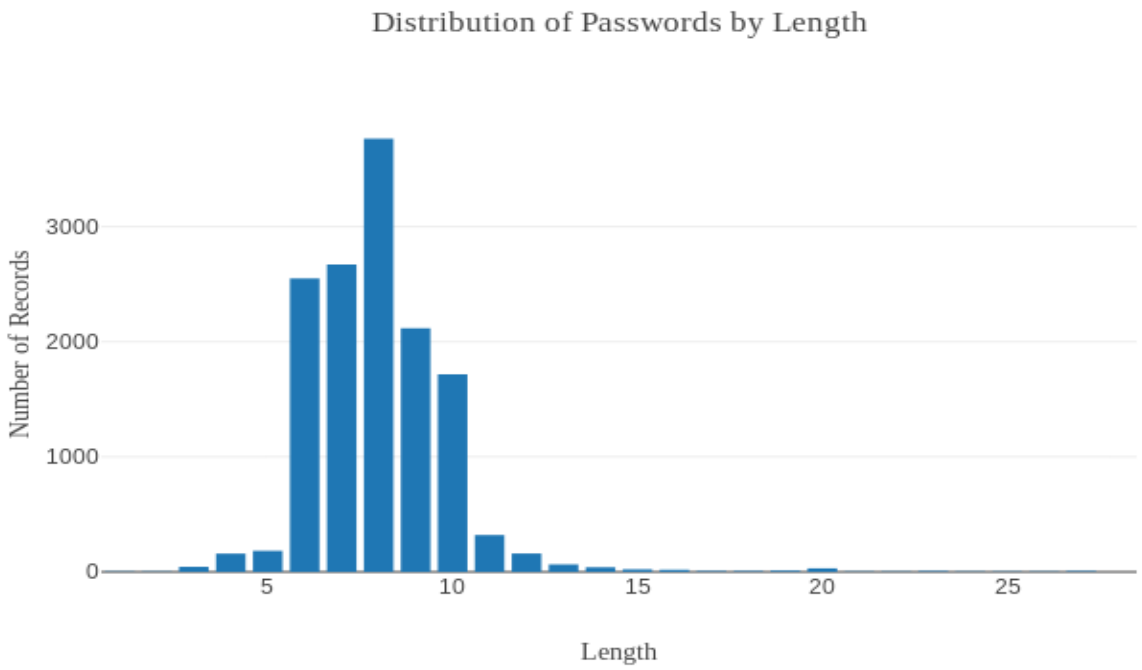

Figure 56. Bar Chart: Wayne State passwords by length.

PWDLength	#_of_Passwords
1	5
2	5
3	44
4	159
5	183
6	2552
7	2671
8	3765
9	2119
10	1716
11	320
12	160
13	65
14	40
15	21
16	19
17	7
18	9
19	10
20	31
21	4
22	4
23	7
24	4
25	2
26	3
27	6
28	1

Figure 57. Wayne State passwords by length.

Password recurrence, reuse by user. 1,156 passwords, or eight percent of passwords, appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 58. For users with more than one breached password, password reuse was measured. Of the 1,167 users with multiple breached passwords, 240 (20.6%) were found to have reused the same password twice, while 4 users were found to have reused the same password three times.

PWD	Count
123456	18
myspace1	17
password	14
anthony	8
waynestate	7
a123456	7
password1	7
jordan23	6
blink182	6
love	6

Figure 58. Most common passwords within the Wayne State dataset.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user with more than one password was calculated. 559 users (48%) averaged a distance of 3 or less, indicating passwords that vary within three characters (Figures 59 & 60).

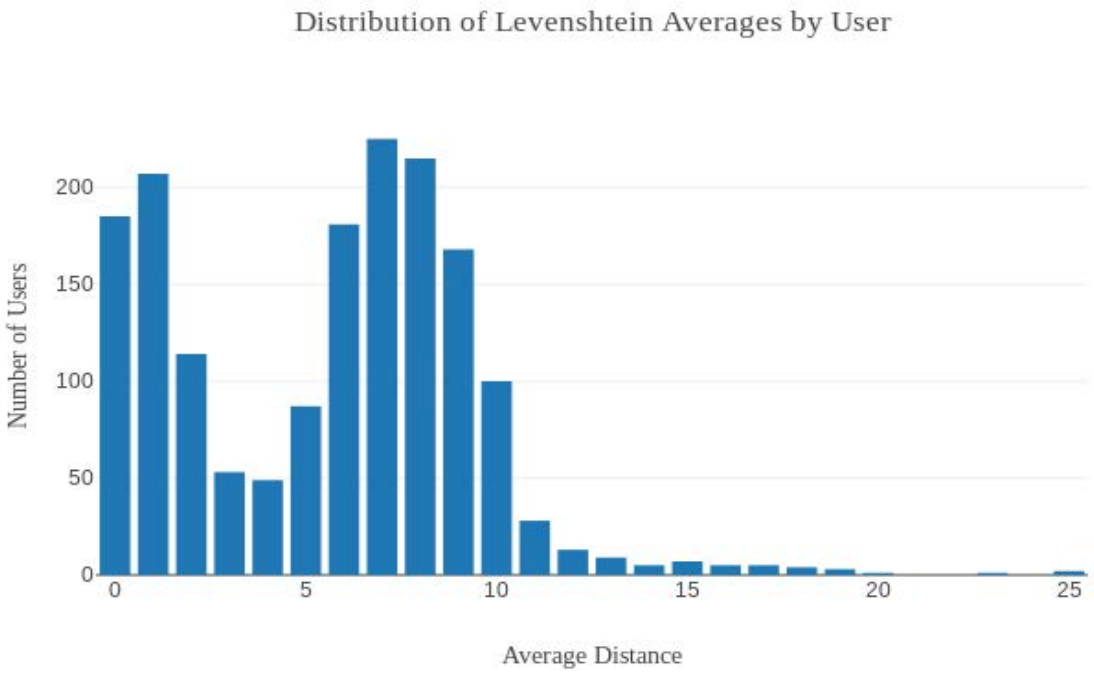

Figure 59. Bar Graph: Distribution of Levenshtein Distance averages for Wayne State.

Average_Lev_Dist	Count
0.0	185
1.0	207
2.0	114
3.0	53
4.0	49
5.0	87
6.0	181
7.0	225
8.0	215
9.0	168
10.0	100
11.0	28
12.0	13
13.0	9
14.0	5
15.0	7
16.0	5
17.0	5
18.0	4
19.0	3
20.0	1
23.0	1
25.0	2

Figure 60. Distribution of Levenshtein Distance averages for Wayne State.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 29 shows the most common root strings within the Wayne State dataset.

rootPWD	Appearances
detroit	36
wayne	35
love	34
password	34
myspace	26
july	26
anthony	23
hockey	23
nicole	22
soccer	18

Figure 61. Most frequent password strings within the Wayne State dataset.

University of Michigan

Number of records. After data cleaning, 40,029 email/password records were extracted from the source dataset. These records corresponded to 34,129 distinct “umich.edu” email addresses. 4,719 email addresses were observed within the dataset more than once, I.E. multiple passwords were recovered for these users.

Breach exposure. The number of passwords recovered for each user ranged from one to nineteen. In other words, within the U-of-M dataset, a single email could be associated with as few as one, or as many as nineteen breach records. Approximately 86.2% of users discovered within the U-of-M dataset had been breached a single time, whereas 11.6% had been breached twice (Figures 62 & 63).

TimesBreached	Count
1	29410
2	3965
3	585
4	102
5	34
6	8
7	2
8	1
9	7
10	2
13	1
14	2
15	4
16	3
17	1
18	1
19	1

Figure 62. U of M users by number of times breached.

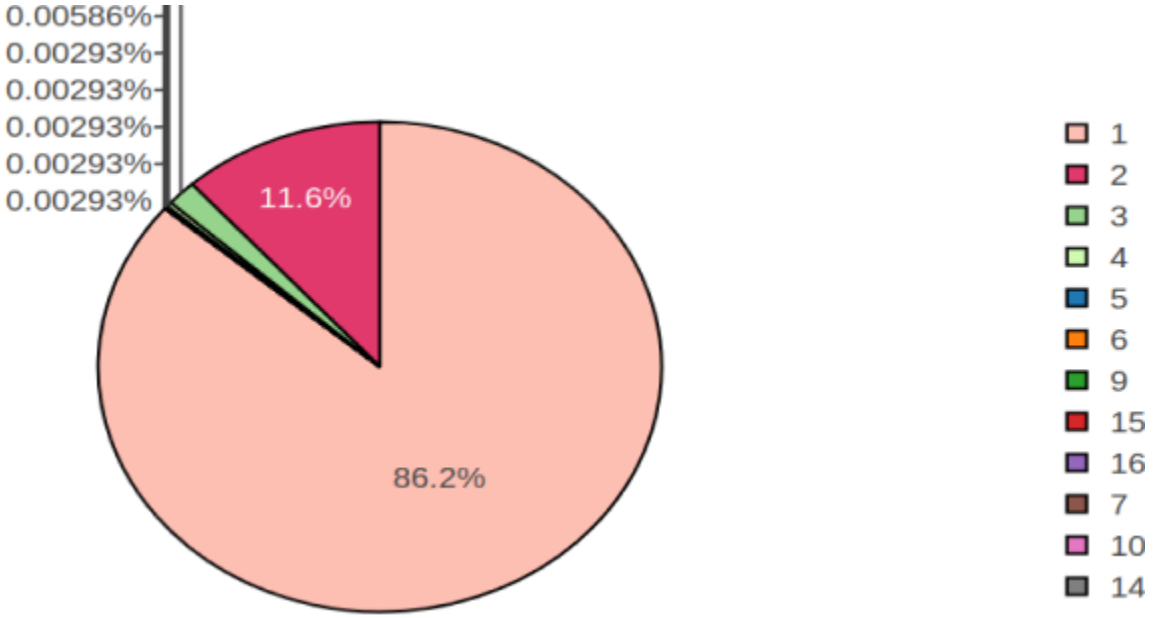

Figure 63. Pie Chart: U of M users by times breached.

Length, min & max. Passwords from the University of Michigan dataset ranged from one character to twenty-nine characters in length; the average length of passwords within the dataset was 8.16 characters. 89% of passwords within the U-of-M dataset ranged from six to ten characters (Figures 64 & 65).

PWDLength	#_of_Passwords
1	34
2	14
3	182
4	289
5	560
6	6675
7	7006
8	10980
9	5931
10	4953
11	1466
12	904
13	410
14	226
15	122
16	106
17	30
18	20
19	18
20	43
21	15
22	16
23	9
24	9
25	4
26	4
27	1
29	2

Figure 64. U of M passwords by length.

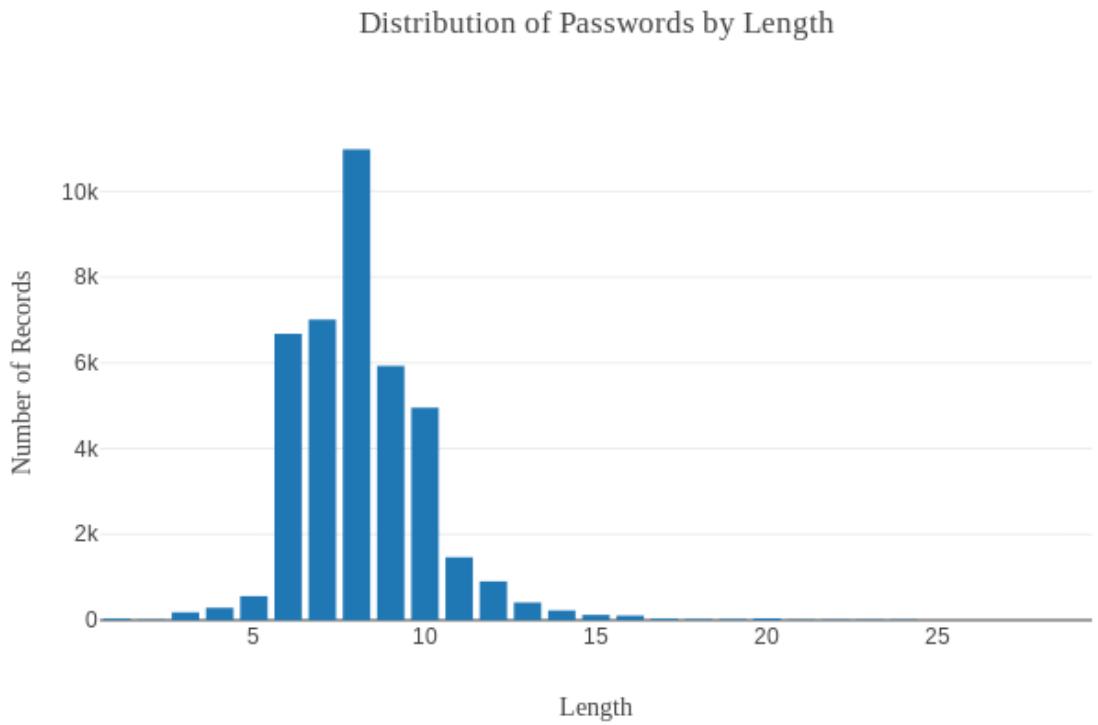

Figure 65. Bar Chart: U of M passwords by length.

Password recurrence, reuse by user. 2,787 passwords, or seven percent of passwords, appeared more than once within the university's dataset. The most common passwords were calculated for the dataset and may be found in Figure 66. For users with more than one breached password, password reuse was measured. Of the 4,719 users with multiple breached passwords, 212 (4%) were found to have reused the same password twice, while eleven users were found to have reused the same password three times.

Count	PWD
73	goblue
65	password
65	michigan
33	123456
32	password1
30	michigan1
29	linkedin
25	opensesame
23	wolverine
20	abc123

Figure 66. U of M passwords by length.

Calculate Levenshtein Distance per user. The average Levenshtein Distance for each user with more than one password was calculated. 1,401 users (30%) averaged a distance of 3 or less, indicating passwords that vary within three characters (Figures 67 & 68).

Average_Lev_Dist	Count
0.0	177
1.0	668
2.0	357
3.0	199
4.0	189
5.0	245
6.0	452
7.0	609
8.0	713
9.0	527
10.0	311
11.0	114
12.0	64
13.0	33
14.0	22
15.0	22
16.0	9
17.0	1
18.0	3
19.0	3
20.0	1

Figure 67. Distribution of Levenshtein Distance averages for U of M.

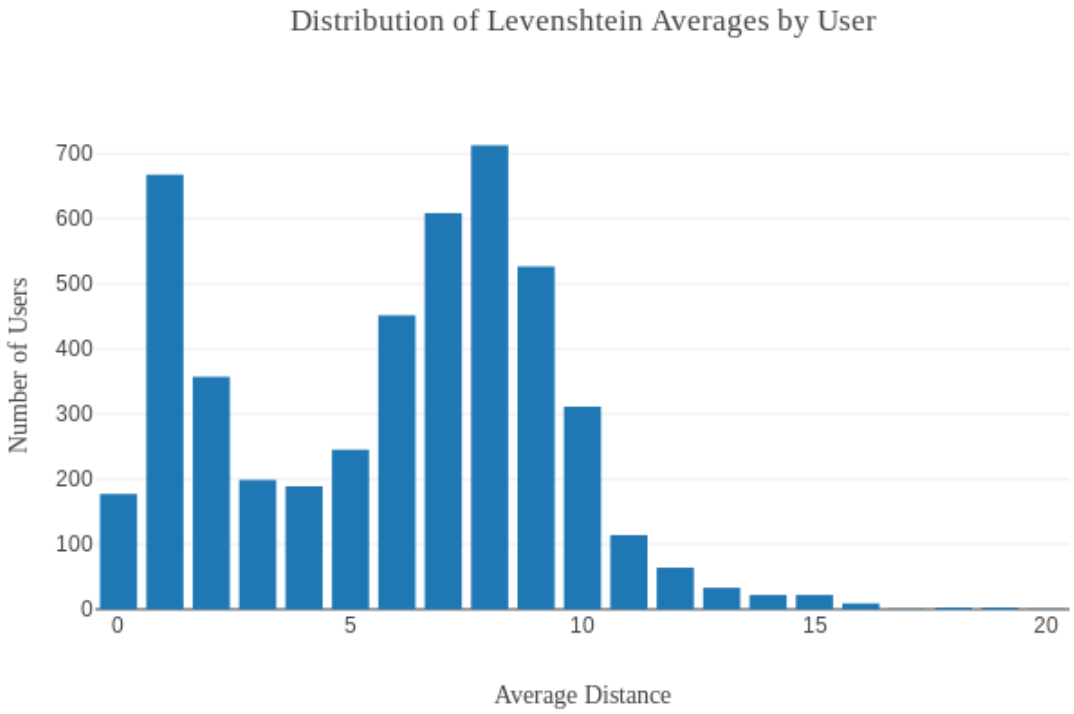

Figure 68. Bar Graph: Distribution of Levenshtein Distance averages for U of M.

Regex Transformations to identify common root words. Regex transformations were applied to the dataset to isolate common root words and substrings. Figure 69 shows the most common root strings within the U of M dataset.

rootPWD	Appearances
goblue	216
michigan	188
password	139
linkedin	98
wolverine	56
soccer	51
blue	51
umich	49
myspace	44
love	34

Figure 69. Most frequent password strings within the U of M dataset.

Chapter 5

This research study aimed to answer two research questions related to password and account security for students of eight Michigan universities. This chapter will summarize and provide additional analysis of the results presented in chapter four. The two research questions of interest to this study were as follows:

1. How many student records can be identified within the breach dataset(s) examined?
2. What password creation & reuse trends exist among the breached student passwords?

Summary of Findings

Number of Records. Of the approximate 1.4 billion records within the source dataset, a total of 144,083 email/password records were attributed to one of the eight institutions of interest to the study. These records represented 121,215 distinct email addresses. Michigan State University featured the most breach records with 43,678, while Ferris State University had the fewest with 1,906 records.

User Breach Frequency. Wayne State University and University of Michigan had the highest range of breaches per user: as nineteen sets of breached credentials located for a given email address within these two datasets. On average among all datasets, 85.9% of email addresses had been breached a single time, within a range of 84.3% – 90.2% for each university dataset. An average of 11.9% had been breached twice, within a range of 8.5% – 15.2%. Only a small percentage of users within each dataset were observed to have been breached more than twice within the dataset.

Password Length. The average password length for all datasets was eight characters. Overall, 90.6 % of passwords ranged from six to ten characters, with a range of 89%-95% for each university dataset.

Password Reuse & Modification. Of the 18,551 users identified with multiple passwords within the dataset, only 10% were found to have reused the same password twice, while less than one percent had reused the same password three times. No instance of password reuse greater than three times per user was observed for any of the universities studied. Levenshtein Distance was calculated for each user with more than one password within the dataset. 2,643 of the 18,551 multi-record accounts (14%) averaged a Levenshtein Distance of 1, indicating passwords that differ by only one character. 6,113 (33%) averaged a distance of three or less, indicating the users employed passwords that differed from one another by 3 characters or less.

Common Passwords & Strings. Common passwords and password sub-strings tended to consist of generic common passwords (such as “password” or “123456”), the names of online services (such as “linkedin”, or “myspace”), various sports (such as “hockey” or “football”) and university specific identifiers such as the school colors, school mascots, school cities, etc.

Conclusions

University users more frequently employed password mutation techniques as opposed reuse of identical passwords. The rate of direct reuse (10%) is interesting when compared with that of Das et al. (2014), which found a much higher (43%) reuse rate of identical passwords among breach users. The results, however, more closely align to that of Bailey, Durmuth, and Parr (2014), which found a rate 14% exact reuse across all breach accounts studied. More data is needed to determine if a causative link exists between university attendance and password reuse rates, or some other related factor. Both Das et al. (2014) and Bailey, Durmuth, and Parr (2014) measured rates of password modification in their studies; those results cannot be directly compared here because a different methodologies and tolerances where used for each calculation. Since very few users where found to have more than three breached passwords

within the dataset, a correlation between the number of breached passwords per user and password modification rates, as existed in Wang et al.'s (2017) study, could not be determined.

Common passwords and password sub-strings for each university followed similar, predicable trends. In addition to the inclusion generically common passwords such as "password", each dataset commonly featured the name of the school, school mascot, school city, or some other unique-but-guessable attribute among the top passwords. If the same pattern holds true for official university passwords, attackers could leverage this knowledge to increase the effectiveness of dictionary attacks against university networks.

Despite the predictability of common passwords and numerical-substrings within the university datasets, their overall rate of appearance was surprising low, indicating that the university passwords were highly heterogenous. Less than one percent of users overall were found to have used one of the top ten most common passwords, while two percent overall were found to have used one of the top ten most-common password substrings. This finding is further supported by the fact the average Levenshtein Distance for all unique combinations, matched the average length of passwords: eight. Had the passwords with-in the dataset been substantially similar, one would have expected a lower average Levenshtein Distance when compared to the average length of passwords.

As eluded to, many individual university metrics, such as average password length, common password ranges, user breach frequency, and the makeup of common passwords were highly similar when compared. This indicates that some aspects of password creation behavior are likely common among the universities studied. Further studies might attempt to pinpoint the reason for the commonalities observed.

Recommendations

This study has shown that password reuse and mutation occur on at least a statistically

significant basis among university students/users. It stands to reason that users may set the same password for both their official university account, as well as auxiliary accounts (such as those that have been breached, and subsequently examined in this study). For this reason, it would be prudent for universities to proactively search for and aggregate compromised account credentials which relate to their domains, then blacklist those passwords from future use. Breached passwords may also be hashed by the university using the same technique used to store account credentials, then compared to the password database to ascertain if a user has a compromised password. Accounts with known compromised passwords may be locked out or reset. These techniques would serve to protect university assets from attacks that leverage breached credentials. Similarly, universities could calculate common root strings that are used by the user population, apply password mangling rules to them, then blacklist those possible passwords. This could reduce the guessability of user passwords, thus further reducing the ability of attackers to leverage password breach analysis for attacks.

Recommendations for Further Study

A principle weakness of this study was the inability to attribute password data to any specific data breach; it was not possible to examine that data within the context of its source, as Notoatmodjo et al. could (2009). Additional studies may use a similar modus operandi, but manually compile breach data, tagging the records with their original source. By doing this, it would be possible to sort results by breach origin, which could allow for more granular password analysis, the possible discovery of trends missed here.

This study only examined a handful of Michigan universities. Future studies might also consider using a broader context by including more schools, additional geographic regions, or different types of institutions. This may allow for a more robust and interesting results. Results of additional studies could be compared to the results of this study.

References

American Psychological Association. (2017). Ethical principles of psychologists and code of conduct. Retrieved December 7, 2018, from https://www.apa.org/ethics/code/index.aspx

American Psychological Association. (2010). *Publication manual of the American Psychological Association*. Washington, DC: American Psychological Association.

Bailey, D. V., Dürmuth, M., & Paar, C. (2014). Statistics on password re-use and adaptive strength for financial accounts. *Proceedings of the International Conference on Security and Cryptography for Networks*, 218–235. https://doi.org/10.1007/978-3-319-10879-7_13

Bonneau, J. (2012). The science of guessing: analyzing an anonymized corpus of 70 million passwords. *Proceedings of the 2012 IEEE Symposium on Security and Privacy,* 538-552. https://doi.org/10.1109/SP.2012.49

Burr, W.E., Dodson, D. F., Polk, W.T., Newton E. M., Perlner, R. A., Gupta, S., Nabbus, E.A. (2006). Electronic authentication guideline. NIST Special Publication 800-63 Ver. 1.0.2. National Institute of Standards and Technology, Gaithersburg MD, Retrieved from https://csrc.nist.gov/publications/detail/sp/800-63/ver-102/archive/2006-04-30

Das, A., Bonneau, J., Caesar, M., Borisov, N., & Wang, X. (2014). The tangled web of password reuse. *Proceedings 2014 Network and Distributed System Security Symposium*, (February), 23–26. https://doi.org/10.14722/ndss.2014.23357

Dell'Amico, M., Michiardi, P., & Roudier, Y. (2009). Measuring password strength: An empirical analysis. Retrieved from http://arxiv.org/abs/0907.3402

Florencio, D., & Herley, C. (2007). A large-scale study of web password habits. *Proceedings of the 16th International Conference on World Wide Web - WWW '07*, 657.

https://doi.org/10.1145/1242572.1242661

Gilleland, M.(n.d.). Levenshtein Distance, in three flavors. Retrieved December 3, 2018, from https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm

Haque, S. M. T., Wright, M., & Scielzo, S. (2013). A study of user password strategy for multiple accounts. *Proceedings of the Third ACM Conference on Data and Application Security and Privacy - CODASPY '13*, 173. https://doi.org/10.1145/2435349.2435373

Klein, B. (n.d.). Levenshtein Distance. Retrieved December 3, 2018, from https://python-course.eu/levenshtein_distance.php

Klein, D. V. (1991). Foiling the cracker; A survey of, and improvements to Unix password security. *Proceedings of the 14th DoE Computer Security Group.* Retrieved from https://http://www.danklein.com/dvk/publications/passwd.pdf

Liu, Z., Hong, Y., & Dechang P. (2014). A large-scale study of web password habits of Chinese network users. *Journal of Software, 9*(2). Retrieved from http://www.jsoftware.us/vol9/jsw0902-05.pdf

Mazurek, M. L., Komanduri, S., Vidas, T., Bauer, L., Christin, N., Cranor, L. F., … Ur, B. (2013). Measuring password guessability for an entire university. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13*, 173–186. https://doi.org/10.1145/2508859.2516726

Morris, R., & Thompson, K. (1979). Password security: a case history. *Communications of the ACM*, *22*(11), 594–597. https://doi.org/10.1145/359168.359172

Narayanan, A., & Shmatikov, V. (2005). Fast dictionary attacks on passwords using time-space tradeoff. *Computer and Communications Security*, 364. https://doi.org/10.1145/1102120.1102168

Notoatmodjo, G., & Thomborson, C. (2009). Passwords and perceptions. *Conferences in Research and Practice in Information Technology Series*, *98*(Aisc), 71–78. Retrieved from https://www.cs.auckland.ac.nz/~cthombor/Pubs/IdMgmt/gno09.pdf

Robinson, K. (2018, September). Analyzing Pwned Passwords with Apache Sparks. Presented at the GrrCon Hacker Conference 2018, Grand Rapids, MI.

Schneier, B. (2006, December 14). Real-world passwords [Web blog post]. Retrieved December 3, 2018, from https://www.schneier.com/blog/archives/2006/12/realworld_passw.html

Spafford, E. H. (1992). Observing reusable password choices. *System*, 1–14. Retreived from https:// docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1969&context=cstech

Wang, C., Jan, S. T. K., Hu, H., & Wang, G. (2017). Empirical analysis of password reuse and modification across online service. Retrieved from http://arxiv.org/abs/1706.01939

Wu, T. (1999). A real-world analysis of kerberos password security. *Proceedings of the 1999 ISOC Symposium on Network and Distributed System Security*, 1–10. Retrieved from http://www.gnu.org/s/shishi/wu99realworld.pdf

Appendix A

Data Cleaning

```
In [1]: #Import and start spark session

        #Note: " \" denotes a line return in Pyspark
        #and is not strictly part of the code.

        #Line returns were required in order to format
        #the notebook for printing

        import pyspark as spark
        from pyspark.sql import SparkSession
        spark = SparkSession \
            .builder \
            .appName("Breach Analysis") \
            .config("spark.some.config.option") \
            .getOrCreate()

In [2]: from pyspark.sql import *

In [3]: from pyspark.sql.functions import *

In [ ]:

In [4]: df = spark.read.csv('UniversityLists/Raw/Ferris.csv') \
        .toDF('Email', 'PWD')

In [ ]:

In [5]: df.show(10, False)

+---------------------+----------+
|Email                |PWD       |
+---------------------+----------+
|AaronWaltz@ferris.edu|noveber10 |
|abatak@ferris.edu    |ktloo22   |
|abbasea@ferris.edu   |economic  |
|abrahaa2@ferris.edu  |Callie25  |
|AdsTorch@ferris.edu  |callicat  |
|aeblih@ferris.edu    |fsu11hc07 |
```

```
|afzala@ferris.edu    |B7umnen2  |
|ahearnt1@ferris.edu  |6stringer |
|aholak@ferris.edu    |walker8521|
|ahopcroa@ferris.edu  |bakedpotat|
+---------------------+----------+
only showing top 10 rows
```

```
In [ ]: df.count()
```

```
In [ ]:
```

Normalize Email Column

```
In [6]: #Show strange subdomains
        df.select("*").filter(df.Email.contains(".edu.")) \
        .show(5000, False)
```

```
+-------------------------------+-------+
|Email                          |PWD    |
+-------------------------------+-------+
|chrisim@ferris.edu.fsuimail.com|krink41|
+-------------------------------+-------+
```

```
In [ ]: #Remove strange subdomains
        df = df.select("*").filter(~(df.Email.contains(".edu.")))
```

```
In [7]: #Show www. prefixes on email address
        df.select("*").filter(df.Email.contains("www.")) \
        .show(5000, False)
```

```
+--------------------------+---------+
|Email                     |PWD      |
+--------------------------+---------+
|www.brewerc2@ferris.edu   |password |
|www.burrk@ferris.edu      |dream    |
|www.jackso78@ferris.edu   |peaches11|
|www.thomassmith@ferris.edu|hunter123|
+--------------------------+---------+
```

```
In [ ]: #Remove records with www.prefix on email address
        df = df.select("*").filter(~df.Email.contains("www."))
```

```
In [ ]:
```

```
In [11]: #Show instances of Yahoo, Gmail, or Hotmail in Email
         df.select("*").filter(df.Email.contains("yahoo") | \
                               df.Email.contains("gmail") | \
                               df.Email.contains("hotmail")) \
         .show(500,False)
```

```
+--------------------+----------------------------------+
|Email               |PWD                               |
+--------------------+----------------------------------+
|johar.d@hotmail.co.uk|yelkylethornton@ferris.edu        |
|smart_1958@yahoo.com |smartonlotoczkj@ferris.edu:jackson9|
+--------------------+----------------------------------+
```

```
In [ ]: # Remove instances of Yahoo, Gmail, or Hotmail in Email
        df = df.select("*").filter(~(df.Email.contains("yahoo") | \
                                     df.Email.contains("gmail") | \
                                     df.Email.contains("hotmail")))
```

```
In [ ]:
```

```
In [ ]: #Show & Count # of Emails Greater than 30 Char or
        #that contain ";" other web domains
        df.select("*").filter((length("Email") > 30) | \
                              df.Email.contains(";") |  \
                              df.Email.contains('.co') | \
                              df.Email.contains('.net') | \
                              df.Email.contains('.mil') | \
                              df.Email.contains('.gov') |
                              df.Email.contains('.de') |
                              df.Email.contains('.ru'))
        .show(500, False),

        df.select("*").filter((length("Email") > 30) | \
                              df.Email.contains(";") | \
                              df.Email.contains('.co') | \
                              df.Email.contains('.net') | \
                              df.Email.contains('.mil') | \
                              df.Email.contains('.gov') \
                              | df.Email.contains('.de') | \
                              df.Email.contains('.ru')).count()
```

```
In [ ]: # Isolate any good records returned in search
        df.select("*").filter((df.Email.contains("Aimee.r")) \
                              .show(50, False)
```

```
In [ ]:
```

```
In [ ]: #Save isolated records in a new dataframe
        df2 = df.select("*").filter((df.Email.contains("david.cole@") | \
                                     df.Email.contains("david.rueda") | \
                                     df.Email.contains("christina.depugh@") \
                                     | df.Email.contains("Kara.cox") | \
                                     df.Email.contains("marc.cogan") | \
                                     df.Email.contains("l.dezur") | \
                                     df.Email.contains("lauren.miller") | \
                                     df.Email.contains("j.corvino")| \
                                     df.Email.contains("joseph.rubino") | \
                                     df.Email.contains("a.delphy")))

In [ ]: df2.count()

In [ ]:

In [ ]:

In [ ]: #Remove Bad Email records > 30 char or that contain multiple email domains

        df = df.select("*").filter(~((length("Email") > 30) | \
                                     df.Email.contains(";") | \
                                     df.Email.contains('.co') \
                                     | df.Email.contains('.net') \
                                     | df.Email.contains('.mil') |  \
                                     df.Email.contains('.gov') | \
                                     df.Email.contains('.de') |
                                     df.Email.contains('.ru')))

In [ ]:
```

Combine filtered dataframes using SQL

```
In [ ]: df.createOrReplaceTempView("PWData")
        df2.createOrReplaceTempView("SavedPWDs")

In [ ]: #Show and Count Results
        spark.sql("Select Email, PWD from PWData UNION \
        Select Email, PWD from SavedPWDs").show()
        spark.sql("Select Email, PWD from PWData UNION \
        Select Email, PWD from SavedPWDs").count()

In [ ]: #Save combined data frame
        df = spark.sql("Select Email, PWD from PWData \
        UNION Select Email, PWD from SavedPWDs")

In [ ]:
```

Normalize Password Field

```
In [8]: #Isolate Null Passwords
        df.select("*").filter(df.PWD.contains("null")) \
        .show(50, False)

+-----+---+
|Email|PWD|
+-----+---+
+-----+---+
```

```
In [ ]: #Remove Null Passwords
        df = df.select("*").filter(~df.PWD.contains("null"))
```

```
In [10]: #Isolate & Count 15 char numeric non-passwords
         df.select("*").where((length("PWD") == 15) & \
                              ~(df.PWD.contains("a") | df.PWD.contains("b") |
                                df.PWD.contains("c") | df.PWD.contains("d") |
                                df.PWD.contains("e") | df.PWD.contains("f") |
                                 df.PWD.contains("g") | df.PWD.contains("h")|
                                df.PWD.contains("i") | df.PWD.contains("j") |
                                df.PWD.contains("k") | df.PWD.contains("l") |
                                df.PWD.contains("m") | df.PWD.contains("n") |
                                df.PWD.contains("o") | df.PWD.contains("p") |
                                df.PWD.contains("q") | df.PWD.contains("r") |
                                df.PWD.contains("s") | df.PWD.contains("t") |
                                df.PWD.contains("u") | df.PWD.contains("v") |
                                df.PWD.contains("w") | df.PWD.contains("x") |
                                df.PWD.contains("y") | df.PWD.contains("z") |
                                df.PWD.contains("A") | df.PWD.contains("B") |
                                df.PWD.contains("C") | df.PWD.contains("D") |
                                df.PWD.contains("E") | df.PWD.contains("F") |
                                 df.PWD.contains("G") | df.PWD.contains("H")|
                                df.PWD.contains("I") | df.PWD.contains("J") |
                                df.PWD.contains("K") | df.PWD.contains("L") |
                                df.PWD.contains("M") | df.PWD.contains("N") |
                                 f.PWD.contains("O") | df.PWD.contains("p") |
                                df.PWD.contains("P") | df.PWD.contains("Q") |
                                df.PWD.contains("R") | df.PWD.contains("S") |
                                df.PWD.contains("T") | df.PWD.contains("U") |
                                df.PWD.contains("V") | df.PWD.contains("W") |
                                df.PWD.contains("X") | df.PWD.contains("Y") |
                                df.PWD.contains("Z"))).show(20,False) #


         df.select("*").where((length("PWD") == 15) & \
                              ~(df.PWD.contains("a") | df.PWD.contains("b") |
                                df.PWD.contains("c") | df.PWD.contains("d") |
```

```
                                df.PWD.contains("e") | df.PWD.contains("f") |
                               df.PWD.contains("g") | df.PWD.contains("h")|
                                df.PWD.contains("i") | df.PWD.contains("j") |
                                 df.PWD.contains("k") | df.PWD.contains("l") |
                                 df.PWD.contains("m") | df.PWD.contains("n") |
                                  df.PWD.contains("o") | df.PWD.contains("p") |
                                 df.PWD.contains("q") | df.PWD.contains("r") |
                                   df.PWD.contains("s") | df.PWD.contains("t") |
                                   df.PWD.contains("u") | df.PWD.contains("v") |
                                df.PWD.contains("w") | df.PWD.contains("x") |
                                 df.PWD.contains("y") | df.PWD.contains("z") |
                                 df.PWD.contains("A") | df.PWD.contains("B") |
                                df.PWD.contains("C") | df.PWD.contains("D") |
                                 df.PWD.contains("E") | df.PWD.contains("F") |
                                 df.PWD.contains("G") | df.PWD.contains("H")|
                                  df.PWD.contains("I") | df.PWD.contains("J") |
                                 df.PWD.contains("K") | df.PWD.contains("L") |
                                  df.PWD.contains("M") | df.PWD.contains("N") |
                                  df.PWD.contains("O") | df.PWD.contains("p") |
                                  df.PWD.contains("P") | df.PWD.contains("Q") |
                                  df.PWD.contains("R") | df.PWD.contains("S") |
                                 df.PWD.contains("T") | df.PWD.contains("U") |
                                 df.PWD.contains("V") | df.PWD.contains("W") |
                                  df.PWD.contains("X") | df.PWD.contains("Y") |
                                   df.PWD.contains("Z"))).count()
```

```
+--------------------------+---------------+
|Email                     |PWD            |
+--------------------------+---------------+
|ArleneKrellwitz@ferris.edu|630838395067361|
|Beckc5@ferris.edu         |696686794908995|
|boedekr@ferris.edu        |134210164426752|
|brandtc1@ferris.edu       |369133314143433|
|caina@ferris.edu          |642229611693789|
|CarlaHatfield@ferris.edu  |723914901225702|
|chanm@ferris.edu          |829969304731432|
|chencib@ferris.edu        |549739197648814|
|ConnieBookshaw@ferris.edu |568674226988775|
|creedk1@ferris.edu        |713625538184689|
|curt19@ferris.edu         |464907498438612|
|daunaid@ferris.edu        |577165727499492|
|Deanj14@ferris.edu        |390019274934515|
|demaagb@ferris.edu        |369637401930931|
|ecklesa@ferris.edu        |955281986681895|
|farisj@ferris.edu         |688626444959544|
|Fraserc@ferris.edu        |623707832522158|
|fuhrmal1@ferris.edu       |417895170137051|
|gilberl4@ferris.edu       |399196772750335|
```

```
|gilleya@ferris.edu        |622523955129616|
+-------------------------+---------------+
only showing top 20 rows
```

Out[10]: 70

```
In [ ]: #Remove 15 char numerics
        df = df.select("*").where(~((length("PWD") == 15) & \
                        ~(df.PWD.contains("a") | df.PWD.contains("b") |
                          df.PWD.contains("c") | df.PWD.contains("d") |
                         df.PWD.contains("e") | df.PWD.contains("f") |
                          df.PWD.contains("g") | df.PWD.contains("h")|
                         df.PWD.contains("i") | df.PWD.contains("j") |
                         df.PWD.contains("k") | df.PWD.contains("l") |
                         df.PWD.contains("m") | df.PWD.contains("n") |
                         df.PWD.contains("o") | df.PWD.contains("p") |
                         df.PWD.contains("q") | df.PWD.contains("r") |
                         df.PWD.contains("s") | df.PWD.contains("t") |
                         df.PWD.contains("u") | df.PWD.contains("v") |
                         df.PWD.contains("w") | df.PWD.contains("x") |
                         df.PWD.contains("y") | df.PWD.contains("z") |
                         df.PWD.contains("A") | df.PWD.contains("B") |
                         df.PWD.contains("C") | df.PWD.contains("D") |
                         df.PWD.contains("E") | df.PWD.contains("F") |
                          df.PWD.contains("G") | df.PWD.contains("H")|
                         df.PWD.contains("I") | df.PWD.contains("J") |
                         df.PWD.contains("K") | df.PWD.contains("L") |
                         df.PWD.contains("M") | df.PWD.contains("N") |
                         df.PWD.contains("O") |
                          df.PWD.contains("P") | df.PWD.contains("Q") |
                          df.PWD.contains("R") | df.PWD.contains("S") |
                          df.PWD.contains("T") | df.PWD.contains("U") |
                          df.PWD.contains("V") | df.PWD.contains("W") |
                          df.PWD.contains("X") | df.PWD.contains("Y") |
                          df.PWD.contains("Z"))))

In [ ]: #Select Hex Records
        df.select("*").filter(df.PWD.contains("$HEX[")).show(500, False)

In [ ]: #Remove Hex Records
        df = df.select("*").filter(~df.PWD.contains("$HEX["))

In [ ]: #Show Records with Password over 30 Char
        df.select("*").filter((length("PWD") > 30)).show(500, False),
        df.select("*").filter((length("PWD") > 30)).count()

In [ ]: #Remove Password Hashes and other bad records > 30 char
        df = df.select("*").filter((length("PWD") < 30))
```

```
In [ ]: df.count()

In [ ]:

In [ ]: #Show Third Party Domains in the Password column due to bad data
        df.select("*").filter((length(df.PWD) > 14) & \
                              (df.PWD.contains("@"))).show(500, False)

In [ ]: #Remove Third Party Domains
        df = df.select("*").filter(~((length(df.PWD) > 14) & \
        (df.PWD.contains("@"))) | df.Email.contains("____@umich.edu"))
        #.show(500, False)

In [ ]: #Filter .Edu addresses in password column
        df.select("*").filter((df.PWD.contains("@nmu.edu"))) \
        .show(500, False)

In [ ]: #Remove .Edu addresses in password column
        df = df.select("*").filter(~(df.PWD.contains("@wmich.edu")))

In [ ]:

In [ ]: #Other Checks
        df.select("*").filter((length(df.PWD) > 14) & \
                (df.PWD.contains("."))).show(500, False)

In [ ]:

In [ ]: df.select("*").filter(df.PWD.contains(";")).show(500, False)

In [ ]: df.select("*").filter(df.PWD.contains("@")).show(500, False)

In [ ]: df.select("*").filter(df.PWD.contains(".")).show(500, False)

In [ ]:

In [ ]: #Recheck long email and passwords for bad data

In [ ]: df.select("*").filter(length(df.PWD) > 14).show(500, False)

In [ ]: df.select("*").filter(length(df.Email)> 30).show(500, False)

In [ ]: df.select("*").filter(length(df.Email) < 15).show(5000, False)

In [ ]: df = df.select("*").filter(~df.Email.contains("Email"))

In [ ]: df.count()

In [ ]: spark.sql("SELECT COUNT(*) From PWdata \
        WHERE PWD LIKE 'Password'").show()
```

```
In [ ]: #Number of email accounts with more than one record
        df.count() - df.select("Email") \
        .distinct().count()

In [ ]: df.select("Email").distinct().count()

In [ ]: df.select(avg(length(df.Email))).show()

In [ ]: df.select(avg(length(df.PWD))).show()

In [ ]: df.select(min(length(df.PWD))).show()

In [ ]: df.select(max(length(df.PWD))).show(), \
        df.select("*").filter((length(df.PWD) == 19)) \
        .show(30, False)

In [ ]: df.count()

In [ ]:
```

Save Dataframe to CSV file

```
In [ ]: df.coalesce(1).write.option('header', 'true') \
        .csv("/home/major/PWDAnalysis/WMichClean")
```

Appendix B

Password Analysis

In [ ]:
```
#import and start spark session

#Note: " \" denotes a line return in Pyspark
#and is not strictly part of the code.

#Line returns were required in order to format
#the notebook for printing

import pyspark as spark
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Semantic Analysis") \
    .config("spark.some.config.option") \
    .getOrCreate()
```

In [ ]:
```
from pyspark.sql.types import *
```

In [ ]:
```
#functions specified explictly, pyspark min and max defined as min_, max_ to
#avoid compiler confusion between pyspark and python methods
from pyspark.sql.functions import min as min_, avg, max as max_, lower, \
length, udf, round, count, desc, asc, col, sum, regexp_extract, lit, \
levenshtein, first
```

In [ ]:
```
#Plotly for making graphs
import plotly
```

In [ ]:
```
import plotly.offline as py
import plotly.graph_objs as go
plotly.offline.init_notebook_mode()
```

Pre-processing

In [ ]:
```
#Import Clean Dataset
df = spark.read.csv('UniversityLists/Clean/FSU.csv').toDF('Email', 'PWD')
```

In [ ]:

```
In [ ]: #Convert all emails to lowercase so that the number of recurring
        #addresses can be counted properly.

        df = df.select(lower(df.Email).alias('Email'), df.PWD)

In [ ]: #Create SQL View so that SQL queries can run
        df.createOrReplaceTempView("PWData")

In [ ]:

In [ ]: #preview data to verify correct import
        df.show(10, False)
```

Dataset Analysis
Length, Min, & Max

```
In [ ]: #Number of records in dataset
        df.count()

In [ ]:

In [ ]: #Number of unique email addresses
        df.select("Email").distinct().count()

In [ ]: #Number of email accounts with more than one record
        df.count() - df.select("Email").distinct().count()

In [ ]:

In [ ]: #Users with > 1 Breached Record
        df3 = spark.sql("SELECT Email, Count(Email) as TimesBreached \
        From PWData GROUP BY Email ORDER BY TimesBreached DESC")

        df3.select("*").filter(df3.TimesBreached > 1).agg(count(df3.TimesBreached) \
        .alias("Users > 1 Breached Record")).show()

In [ ]:

In [ ]: #Average password length
        df.select(avg(length(df.PWD))).show()

In [ ]: #Lowest password length in dataset
        df.select(min_(length(df.PWD))).show()

In [ ]: df.select(max_(length(df.PWD))).show()

In [ ]:
```

Password Recurrence, Reuse by user

```
In [ ]: #Number of passwords in dataset seen more than once
        df.count() - df.select("PWD").distinct().count()

In [ ]: #Show list of most common passwords
        spark.sql("SELECT Count(PWD) as Count, PWD From PWData \
        GROUP BY PWD ORDER BY COUNT(PWD) DESC").show(50, False)

In [ ]:

In [ ]: #Create second dataframe for password counts
        df2 = spark.sql("SELECT PWD, Count(PWD) as Count From PWData \
        GROUP BY PWD ORDER BY COUNT(PWD) DESC")

In [ ]: #Most common complete passwords
        df2.filter(df2.Count > 2).show(50, False)

In [ ]:

In [ ]: #Count by PWD length
        df2.withColumn("PWDLength", length("PWD")).select("PWDLength", "Count") \
        .groupBy("PWDLength").agg(sum("Count").alias("#_of_Passwords")) \
        .orderBy(asc("PWDLength")).show(50)

In [ ]: #Distribution of Passwords by Length Graph

        dfGraph = df2.withColumn("PWDLength", length("PWD")).select("PWDLength", "Count") \
        .groupBy("PWDLength").agg(sum("Count").alias("Passwords")) \
        .orderBy(asc("PWDLength")).toPandas()

        data = [go.Bar(x=dfGraph.PWDLength,
                       y=dfGraph.Passwords)]

        layout = go.Layout(
                       title='Distribution of Passwords by Length', \
                       titlefont=dict(family='Times New Roman, monospace'), \
                       xaxis=dict(title='Length', \
                       titlefont=dict(family='Times New Roman, monospace',size=14)),
                       yaxis=dict(title='Number of Records', \
                       titlefont=dict(family='Times New Roman, monospace',size=14)))

        fig = go.Figure(data=data, layout=layout)

        plotly.offline.iplot(fig, filename='jupyter-PWDLen')

In [ ]: #Plotly Sample Layout
        '''
        layout = go.Layout(
            title='Plot Title',
            xaxis=dict(
```

```
            title='x Axis',
            titlefont=dict(
                family='Courier New, monospace',
                size=18,
                color='#7f7f7f'
            )
        ),
        yaxis=dict(
            title='y Axis',
            titlefont=dict(
                family='Courier New, monospace',
                size=18,
                color='#7f7f7f'
            )
        )
    )'''
```

In [ ]:

In [ ]:
```
#Create SQL view for count dataframe
df2.createOrReplaceTempView("PWCount")
```

In [ ]:
```
#Most common passwords and the users with those passwords
spark.sql("SELECT Email, PWData.PWD, Count From PWData \
INNER JOIN PWCount ON PWCount.PWD = PWData.PWD \
ORDER BY Count DESC, PWCount.PWD DESC").show(500, False)
```

In [ ]:
```
#Accounts with the most reused passwords
spark.sql("SELECT Email, PWD, Count(PWD) as TimesReused From PWData \
GROUP BY PWD, Email ORDER BY TimesReused DESC").show(40, False)
```

In [ ]:

In [ ]:
```
#Count of Times Password Reused
df3 = spark.sql("SELECT Email, PWD, Count(PWD) as TimesReused From PWData \
GROUP BY PWD, Email ORDER BY TimesReused DESC")

df3.filter(df3.TimesReused > 1).groupBy(df3.TimesReused) \
.agg((count("TimesReused")) \
.alias("Count")).orderBy(asc("TimesReused")).show()
```

In [ ]:

In [ ]:
```
#Emails with the greatest number of breached accounts
spark.sql("SELECT Email, Count(Email) as TimesBreached From PWData \
GROUP BY Email ORDER BY TimesBreached DESC").show(500, False)
```

In [ ]:
```
spark.sql("Select Email, PWD FROM PWData WHERE Email = 'barfied@ferris.edu'").show(15)
```

In [ ]:
```
#Users with > 1 Breached Record
df3 = spark.sql("SELECT Email, Count(Email) as TimesBreached From PWData \
GROUP BY Email ORDER BY TimesBreached DESC")

df3.select("*").filter(df3.TimesBreached > 1).agg(count(df3.TimesBreached) \
.alias("Users > 1 Breached Record")).show()
```

In [ ]:

In [ ]:
```
#Number of records that were breached x times
spark.sql("SELECT Email, Count(Email) as TimesBreached From PWData \
GROUP BY Email ORDER BY TimesBreached DESC").select("TimesBreached") \
.groupBy("TimesBreached").agg((count("TimesBreached")).alias("Count")) \
.orderBy(asc("TimesBreached")).show()
```

In [ ]:
```
dfGraph = spark.sql("SELECT Email, Count(Email) as TimesBreached From PWData \
GROUP BY Email ORDER BY TimesBreached DESC").select("TimesBreached") \
.groupBy("TimesBreached").agg((count("TimesBreached")).alias("Count")) \
.orderBy(asc("TimesBreached"))
```

In [ ]:

In [ ]:
```
#Create Pie Chart of User Breach
dfGraph = spark.sql("SELECT Email, Count(Email) as TimesBreached From PWData \
GROUP BY Email ORDER BY TimesBreached DESC").select("TimesBreached") \
.groupBy("TimesBreached").agg((count("TimesBreached")).alias("Count")) \
.orderBy(asc("TimesBreached"))

#Column objects must be converted into Python Lists for the
#Plotly pie chart function to work.
pieBreach = dfGraph.select(dfGraph.TimesBreached).collect()
pieCount = dfGraph.select(dfGraph.Count).collect()
pieBreach = [int(i.TimesBreached) for i in pieBreach]
pieCount = [int(i.Count) for i in pieCount]

colors = ['#FEBFB3', '#E1396C', '#96D38C', '#D0F9B1']
trace = go.Pie(labels=pieBreach, values=pieCount, textfont=dict(size=20), \
marker=dict(colors=colors,line=dict(color='#000000', width=2)))

layout = go.Layout(legend=dict(font = dict(size=20)), title='', \
titlefont=dict(family='Times New Roman, monospace', size= 20))

fig = go.Figure([trace], layout)

py.iplot(fig, filename='pie_chart')
```

In [ ]:

In [ ]:

Calculate Levenshtein Distance per User

```
In [ ]: #Create new dataframe consisting only of users with multiple records
        df3 = df.select(df.Email).groupBy(df.Email).agg(count("Email") \
        .alias("Count")).orderBy(desc ("Count")).filter("Count > 1")

In [ ]: df3.count()

In [ ]: df3.show()

In [ ]: #Alias email so that it can be referenced uniquely in later joins
        df3 = df3.select((df3.Email).alias("df3mail"), df3.Count)

In [ ]:

In [ ]: #Perform inner join with orginal dataframe to create list
        #of muliple breach users sorted by Email, PWD ASC
        df.select(df.Email, df.PWD).join(df3.select(df3.df3mail), \
        df.Email == df3.df3mail, 'inner')

In [ ]: #Filter out extraneous df3mail column and safe as dataframe
        df3 = df.select(df.Email, df.PWD).join(df3.select(df3.df3mail), \
        df.Email == df3.df3mail, 'inner').select("Email", "PWD")

In [ ]: df3.show(20)

In [ ]: #import functions as character to perform collect_list data
        #operation in aggregate
        from pyspark.sql import functions as F

In [ ]: #Group email with all user passwords in list format
        df3.groupBy(df3.Email).agg(F.collect_list(df3.PWD).alias("PWDList")) \
        .show(20, False)

In [ ]: #Save as new dataframe for continued operations
        df4 = df3.groupBy(df3.Email).agg(F.collect_list(df3.PWD) .alias("PWDList"))

In [ ]: #The built-in Spark Levenshtein distance function was not
        #suitable for this use case,
        #so a python version was modified and included.

        #Adapted from original source at:
        #https://www.python-course.eu/levenshtein_distance.php

        def iterative_levenshtein(s, t):
            """
            Creates comparision matrix to calculate Levenshtein distance
            for inputs s, t
            """
            rows = len(s)+1
```

```
    cols = len(t)+1
    dist = [[0 for x in range(cols)] for x in range(rows)]
    # source prefixes can be transformed into empty strings
    # by deletions:
    for i in range(1, rows):
        dist[i][0] = i
    # target prefixes can be created from an empty source string
    # by inserting the characters
    for i in range(1, cols):
        dist[0][i] = i

    for col in range(1, cols):
        for row in range(1, rows):
            if s[row-1] == t[col-1]:
                cost = 0
            else:
                cost = 1
            dist[row][col] = min(dist[row-1][col] + 1, dist[row][col-1] + 1,
                                 dist[row-1][col-1] + cost)

    output = dist[row][col]

    return output
```

```
In [ ]: #Accept passwords for each user as list, enumerate all unique combinations,
        #calculate average Lev. Distance for each user and return.
        def userLev(testList):

            average = 0
            levOutput = list()

            #Create all unique password pairs and send to Lev distance function
            for num1 in range(len(testList)):
                for num2 in range (num1 + 1, len(testList)):
                    levOutput.append(iterative_levenshtein(testList[num1], testList[num2]))
                    #print(testList[num1] + " / " + testList[num2] + ": " + str(levOutput))

            # iterate and combine results for avg calc

            for item in levOutput:
                average = average + item

            # Number of unique possibilites can be expressed as (n * (n-1)) / 2)
            combos = (len(testList) * (len(testList) - 1) / 2)

            #Calculate average levenshtein distance for all user records
            average = (average / combos)
```

```
        return average

    #Defined as Pyspark User Defined Function for inclusion in pyspark queries
    userLevUDF = udf(userLev, FloatType())
```

In [ ]:

In [ ]:
```
#For each email, show average Lev. Dist. for that user's passwords
df4.select(df4.Email, round(userLevUDF(df4.PWDList), 0) \
.alias("Average_Lev_Dist")).show(50, False)
```

In [ ]:
```
#Save results as new dataframe
df5 = df4.select(df4.Email, round(userLevUDF(df4.PWDList), 0) \
.alias("Average_Lev_Dist"))
```

In [ ]:
```
#Average Lev. Dist. for all Multi-record Users
df5.select(avg(df5.Average_Lev_Dist). \
alias("Average_Lev_Dist_All_MultiRecord_Users")).show()
```

In [ ]:
```
#Distribution of Lev Averages per User
df4.select(df4.Email, round(userLevUDF(df4.PWDList), 0) \
.alias("Average_Lev_Dist")) \
.select("Average_Lev_Dist").groupBy("Average_Lev_Dist") \
.agg((count("Average_Lev_Dist")) \
.alias("Count")).orderBy(asc("Average_Lev_Dist")).show(50, False)
```

In [ ]:
```
#Distribution of Lev Averages per User Graph
dfGraph = df4.select(df4.Email, round(userLevUDF(df4.PWDList), 0) \
.alias("Average_Lev_Dist")) \
.select("Average_Lev_Dist").groupBy("Average_Lev_Dist") \
.agg((count("Average_Lev_Dist")) \
.alias("Count")).orderBy(asc("Average_Lev_Dist")).toPandas()

data = [go.Bar(x=dfGraph.Average_Lev_Dist,
            y=dfGraph.Count)]

layout = go.Layout(
            title='Distribution of Levenshtein Averages by User',
            titlefont=dict(family='Times New Roman, monospace'),
            xaxis=dict(title='Average Distance',
                titlefont=dict(family='Times New Roman, monospace',size=14)),
            yaxis=dict(title='Number of Users',
                titlefont=dict(family='Times New Roman, monospace',size=14)))

fig = go.Figure(data=data, layout=layout)

plotly.offline.iplot(fig, filename='jupyter-LevAvg')
```

In [ ]:
```
#Return df3 to earlier state (df3mail, Count > 1)
df3 = df.select(df.Email).groupBy(df.Email).agg(count("Email").alias("Count")) \
```

```
        .orderBy(desc ("Count")).filter("Count > 1")
        df3 = df3.select((df3.Email).alias("df3mail"), "Count")

In [ ]: #Display Email, Number of Passwords in dataset, and the
        #average Lev. Dist. for those passwords
        df5.join(df3, df5.Email == df3.df3mail, 'inner').select("Email", (df3.Count) \
        .alias("No_of_PWDS"), "Average_Lev_Dist").show(20, False)

In [ ]: #Create new dataframe
        df6 = df5.join(df3, df5.Email == df3.df3mail, 'inner') \
        .select("Email", (df3.Count) \
        .alias("No_of_PWDS"), "Average_Lev_Dist")

In [ ]:

In [ ]: df6.createOrReplaceTempView("PWLev")

In [ ]: #Distribution of Lev Averages,
        #Broken down by the number of passwords for each user

        spark.sql("SELECT No_of_PWDS, Average_Lev_Dist, COUNT(Average_Lev_Dist) \
        AS Count FROM PWLev GROUP BY No_of_PWDS, Average_Lev_Dist \
        ORDER BY No_of_PWDS ASC, Average_Lev_Dist ASC ").show(50)

In [ ]:

In [ ]: #Aggregate Lev Distance Averages for
        #X num passwords rounded to nearest whole number

        df6.select("No_of_PWDS", df6.Average_Lev_Dist,).groupBy("No_of_PWDS") \
        .agg(round(avg(df6.Average_Lev_Dist),2).alias("Aggregate_Lev_Average")) \
        .orderBy("No_of_PWDS").show()

In [ ]:

In [ ]: #Average Lev Dist by Number of Breached Passwords
        dfGraph= df6.select("No_of_PWDS", df6.Average_Lev_Dist,).groupBy("No_of_PWDS") \
        .agg(round(avg(df6.Average_Lev_Dist),2).alias("Aggregate_Lev_Average")) \
        .orderBy("No_of_PWDS").toPandas()

        # Create a trace
        trace = go.Scatter(
            x = dfGraph.No_of_PWDS,
            y = dfGraph.Aggregate_Lev_Average,
            mode = 'lines+markers'
        )

        data = [trace]
```

```
        layout = go.Layout(
                    title='', titlefont=dict(family='Times New Roman, monospace'),
                    xaxis=dict(title='# of Passwords per User in Dataset',
                        titlefont=dict(family='Times New Roman, monospace',size=14)),
                    yaxis=dict(title='Average Levenshtein Distance',
                        titlefont=dict(family='Times New Roman, monospace',size=14)))

        fig = go.Figure(data=data, layout=layout)
        # Plot and embed in ipython notebook!
        py.iplot(fig, filename='basic-scatter')
```

Calculate Levenshtein Distance Between All Passwords

```
In [ ]: '''
        Effectively a crossjoin of the password column against itself,
        except here only unique combinationsand produced instead of all
        possible combinations

        Accepts a list containing passwords, enumerates all unique combinations
        Writes to csv in working directory
        '''

        def uniqueCombos(testList):

            f = open("uniqueCombos.csv", "w+")

            #Create all unique password pairs, write to list iteratively
            for num1 in range(len(testList)):
                for num2 in range (num1 + 1, len(testList)):
                    f.write(testList[num1] + "," + testList[num2] + '\n')
            f.close()
            return "Success"

        uniqueCombosUDF = udf(uniqueCombos)

In [ ]: #Push all password records into an single list
        #Should work for up to ~500 Million Rows (list container max)
        dfReaper = df.agg(F.collect_list(df.PWD).alias("PWDList"))

In [ ]: #Run the uniqueCombos UDF
        #(Outputs success message if everything ran correctly)
        dfReaper.select(uniqueCombosUDF(dfReaper.PWDList)).show()

In [ ]: #Read the csv file back into spark, save as a dataframe
        df7 = spark.read.csv('uniqueCombos.csv').toDF('PWD0', 'PWD')

In [ ]: #Test if the number of unique records for the orginal dataset
        #is correct per formula (N * (N-1)) / 2
        if ((df.count()) * (df.count() - 1)) / 2 == df7.count():
```

```
            print("Output looks correct; " + str(df7.count()) + " Records")
        else:
            print("Something is wrong")
```

In [ ]:

In [ ]:
```
df7.show(10)
```

In [ ]:

In [ ]:
```
#Create a sql view containing only matching passwords
df7.filter(df7.PWD0 == df7.PWD).createOrReplaceTempView("PWDupl")
spark.sql("SELECT PWD, PWD0, COUNT(PWD) AS Count FROM PWDupl \
GROUP BY PWD, PWD0 ORDER BY Count DESC").show()
```

In [ ]:
```
spark.sql("SELECT PWD, PWD0, COUNT(PWD) AS Count FROM PWDupl \
GROUP BY PWD, PWD0 ORDER BY Count DESC").show()
```

In [ ]:
```
spark.sql("SELECT PWD, PWD0 FROM PWDupl").count()
```

In [ ]:

In [ ]:
```
spark.sql("SELECT PWD, PWD0, COUNT(PWD) AS Count FROM PWDupl \
GROUP BY PWD, PWD0 ORDER BY Count DESC").count()
```

In [ ]:

In [ ]:
```
df7.withColumn("Lev", levenshtein('PWD', 'PWD0')).show()
```

In [ ]:
```
#Use Spark built-in Levenshtein distance formula to
#calculate lev dist for all pairs.
df8 = df7.select("*").withColumn("Lev", levenshtein(df7.PWD0, df7.PWD))
```

In [ ]:
```
df8.select("*").filter(df8.Lev < 3).filter(df8.Lev > 0).show(10, False)
```

In [ ]:
```
#Average Lev. Dist. for all Users from X University
df8.select(avg(df8.Lev).alias("Average_Lev_Dist_University")).show()
```

In [ ]:

In [ ]:
```
#Select passwords that have a closeness of less than 3
#Identifies passwords that are most highly correlated
#to other passwords within the dataset
df8.select("*").filter(df8.Lev < 3).groupBy("PWD0").agg((count("Lev")) \
.alias("Lev Matches < 3")).orderBy(desc("Lev Matches < 3")).show(25,False)
```

In [ ]:
```
#Save as new dataset
df9 = df8.select("*").filter(df8.Lev < 3).groupBy("PWD0").agg((count("Lev")) \
.alias("Lev Matches < 3")).orderBy(desc("Lev Matches < 3"))
```

In [ ]:

Regex Transformations to Identify Common Root Words

```
In [ ]: #Use regex to remove all non alphabetic chars in each password string
        df9.select(lower(regexp_extract(df9.PWD0, '\D+', 0)) \
        .alias("rootPWD"), "Lev Matches < 3")\
        .show(5)

In [ ]: #Save as new dataset
        df10 = df9.select(lower(regexp_extract(df9.PWD0, '\D+', 0)) \
        .alias("rootPWD"), "Lev Matches < 3")

In [ ]:

In [ ]: #By removing numbers and special characters, highly correlated
        #strings can be aggregated.

        #This shows the root alphabetic strings for the most
        #highly corr. passwords in the dataset

        df10.select("*").filter(("rootPWD != ''")).groupBy("rootPWD") \
        .agg((sum("Lev Matches < 3")) \
        .alias("Sum of Lev Values")).orderBy(desc("Sum of Lev Values")).show(10)

In [ ]: #Apply same technique to the orginal dataset.
        #This shows the most common root alphabetic strings in the dataset
        df10 = df.select(lower(regexp_extract(df.PWD, '\D+', 0)) \
        .alias("rootPWD")) \
        .filter("rootPWD != ''").filter(length("rootPWD") > 3) \
        .withColumn('Count', lit(1))

In [ ]: df10.groupBy("rootPWD").agg(sum("Count").alias("Appearances")) \
        .orderBy(desc("Appearances")).show(10)

In [ ]: dfGraph = df10.groupBy("rootPWD").agg(sum("Count").alias("Appearances")) \
        .orderBy(desc("Appearances")).limit(10).toPandas()

        trace = go.Table(columnwidth = [60,40],
            header=dict(values=['Root Password', 'Count'],  \
            fill = dict(color='#C2D4FF'), align = ['center'], \
                 font = dict(color = 'black', size = 16)),
            cells=dict(values=[dfGraph.rootPWD, dfGraph.Appearances], \
                       fill = dict(color='#F5F8FF'),
                       align = ['center'], font = dict(color = 'black', size = 14)))

        data = [trace]
        py.iplot(data, filename = 'basic_table')

In [ ]:
```

Save Dataframe to CSV file

In [ ]:
```
#Export a dataset to csv format
#df.coalesce(1).write.option('header', 'true') \
#.csv("/home/major/PWDAnalysis/dataset")
```