Explorations in Information Security and Privacy

CPSC 329

Assignment 1

Kenneth Sharman

ID Number: 00300185

Tutorial: T04

Instructor: Joshua Horacsek

TA: Shang Li

February 6, 2019

Question 1

1.1 Hacktivism

The WANK Worm

On October 16th 1989, The Space Physics Analysis Network (SPAN) of the National Aeronautics and Space Administration (NASA) was attacked by the WANK worm (Worms Against Nuclear Killers) [1]. It was just two days before the plutonium powered Galileo Space probe was launched into space, when the agency's computer screens began to display the messages; "Your computer has been officially Wanked. You talk of times of peace for all, and then prepare for war" and "Remember, even if you win the rat race, you're still a rat" [3].

NASA was able to quickly develop a fix for the worm, dubbed an anti-WANK prodecure. These worm killing programs were very effective however the worm still had time to propagate through the network of connected computers. Within weeks it had circumvented the globe. The source was eventually traced back to Australia, where the Melbourne federal police attribute the worm to hackers who went by the name Electron and Phoenix. Despite this, the attack is considered to remain unresolved as no one was ever charged. This was the global introduction to Hacktivism, as it was the first major worm to include a political message [1], which was motivated Cold War nihilism. Reaction included increased importance towards password security, as the worm successfully guessed the passwords. That is to say; proper password selection became a prominent topic in the future discussions of information system security.

Link:

https://www.theage.com.au/national/hack-to-the-future-20030525-gdvriu.html

Evidence that link is credible:

The Age has produced daily new publications in Melbourne Australia since 1854. The fact that this publisher has withstood the test of time gives the article a certain amount of credibility, as a valid news outlet is not likely to remain a business if it has a long period of controversial reporting.

The information included in the article is consistent with other independent sources, in particular the article cited here [3]. This establishes a certain amount of accuracy and objectivity of the article.

1.2 Cyber Warfare

The Farewell Dossier

This example is also linked to the Cold War. In 1981, France was able to recruit a K.G.B officer, and obtain access to documents that showed how the Soviets were purchasing US made equipment and software. The French referred to this document as the Farewell Dossier, and shared the information with the States. Instead of focusing on preventing this from occurring, the US National Security Council decided to use it to their advantage. A trojan horse was included on a computer control system that would eventually make its way to a pipeline between Europe and Siberia. The software was designed specifically to pass any tests that the Soviets might think to run, but then malfunction when implemented and produce pressures that greatly exceeded the capacity of the pipeline. The result was the a 3 kiloton explosion- which Norad monitors initially mistook as a nuclear reaction [5].

This example illustrates one of the earliest examples of cyber warfare, as software was intentionally used by the United States to gain ground in the Cold war. The blast resulted in a severe setback for the Soviets, as it cast doubt on all information and goods that had been obtained from the US, which had major consequences on all scientific research and engineering. The economic cost of the explosion was equally as severe, since the pipeline was expected to generate approximately 8 billion dollars a year for the Soviets. The example shows how profound of an impact a single piece of software can have. It comes as no surprise that event highlighted the importance of integrity in computer security- if the Soviets had detected the unexpected code, then the explosion would have been avoided.

Link:

https://www.nytimes.com/2004/02/02/opinion/the-farewell-dossier.html

Evidence that link is credible:

The source article is published in the New York Times which has established itself as a world leader in journalism. The article itself was written by William Safire, who made quite the name for himself as an author and journalist [4]. This combination of author and publisher support the credibility of the article. Furthermore, the accuracy and objectivity of the article were cross-referenced with several other credible sources (see the reference link for a publication from the CIA) [2].

1.3 Cyber Crime

WannaCry

In May 2017 an encryption-based ransomware, referred to as WannaCry, infected approximately 200,000 computers worldwide. Most of the infected computers were running Microsoft 7 and had not installed the recent security update. The ransomware was delivered via email as an attachment. Once a user opened the attachment, their system files were copied and encrypted, the original files were deleted, and a ransom of $300 was requested. The ransom had a deadline and failure to transfer the require amount of Bitcoin within this time line would result in an increased cost to decrypt the files or the complete loss of data. This attack is considered Cyber Crime due to the illegal actions of the attacker by use of the Internet.

The attack was made possible through an exploit developed by the US National Security Agency. Microsoft had released patches to close this exploit the previous month, but computers without this fix were susceptible to the attack. A hardcoded kill switch was (accidentally) found by Marcus Hutchins and within 4 days the number of new infections has nearly subsided. The amount of Bitcoin transfers to the attackers was mitigated by reports that payment did result in file access, however the severity of the impact was seen in lost time and data- resulting in economic losses in the billions of dollars. The US Department of Justice attributes the attack to a team of experts for the North Korean Reconnaissance, however North Korea denies this accusation.

Link:

https://www.csoonline.com/article/3227906/ransomware/what-is-wannacry-ransomware-how-does-it-infect-and-who-was-responsible.html

Evidence that link is credible:

The first piece of evidence is that the article itself has a large number of source links. The links themselves offer relevant information pertaining to their use in the original article. Clearly, the author was quite thorough with his research of the topic. The second source of credibility comes from the content of the site. CSO is a specialty publication which focuses explicitly on security and risk management topics. Since this is the focus of their business, it is less likely that articles would be posted without getting their facts straight.

1.4 Cyber Espionage

2010 Chinese Shadow Network Attack on India

In 2010 a China based cyber espionage operation sole classified documents from the Indian government. The operation was able to exploit several Internet services such as; Twitter, Google groups and Yahoo mail, to penetrate high-level government networks. Sensitive information was stolen from India's National Security Council Secretariat, which included security assessments of several of their border states. The attack was discovered by an 8 month investigation by researchers from Canada's Information Warfare Monitor and the Unites State's Shadowserver foundation.

There was no evidence to support any involvement by the Chinese government, in fact the Chinese computer emergency response team was eager to understand what the researchers had learned about the attack in order to shut down the operation. The true attackers and their motive remains unclear. Despite this, we are considering this as *Cyber Espionage* and not simply *Cyber Crime*, due to the illicit acquisition of confidential government information.

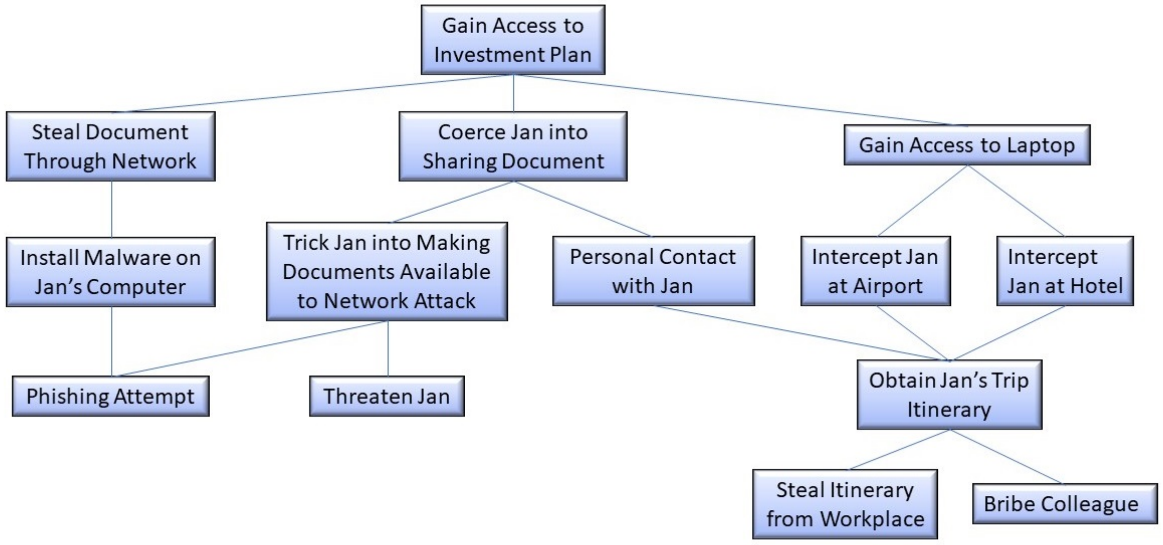
Link:

https://www.theguardian.com/technology/2010/apr/06/cyber-spies-china-target-india

Evidence that link is credible: The publisher of the article is *The Guardian*, a daily newspaper that originated in 1821. Due to its longevity and loyal following, it has received numerous awards over the years. This establishes a certain level of reliability as a news outlet and in my opinion gives a certain level of credibility to the articles it publishes. The *timeliness* of the web page provides a second piece of evidence for credibility. The article was published in 2010, however the web site is most certainly up to date, as the links contained in the page direct you to recent news publications. This establishes that the site is legitimate and is actively being maintained.

Question 2 - Attack Tree

Consider a scenario where Jane, the CEO of a company, has stored the company's highly confidential investment plan on her laptop. Jane has a busy travel schedule, passing through airports and staying at hotels in different cities. In addition to doing work, she uses her laptop for email as well as browsing internet for per personal purposes. Draw an attack tree assuming the goal of the attacker is to gain access to the investment plan document.



Gain Access to Investment Plan
Steal Document Through Network
Coerce Jan into Sharing Document
Gain Access to Laptop
Install Malware on Jan's Computer
Trick Jan into Making Documents Available to Network Attack
Personal Contact with Jan
Intercept Jan at Airport
Intercept Jan at Hotel
Phishing Attempt
Threaten Jan
Obtain Jan's Trip Itinerary
Steal Itinerary from Workplace
Bribe Colleague

Figure 1: Attack Tree with Goal of Obtaining Jan's Investment Plan

Three basic methods of obtaining the investment plan are illustrated in the attack tree; unauthorized physical access to the laptop, have Jan voluntarily (or non voluntarily) share the document, or steal the document through network access to Jan's computer. Clearly, this represents a very basic outline of how to accomplish the root objective- the point of the attack tree is to make an attempt to visualize all possible methods, starting at the leaf node, that *could* result in the root node (main objective).

The list is not comprehensive by any means, and this becomes clear when considering the "Steal Document Through Network" node. I have only included one child node to this, in the form of Malware installation, which again has only one child. From the WannaCry example above, we know that perhaps a way of installing malware could be through email. Further, the installation of malicious software could lead (for example) straight to the root node; Suppose the software forced Jan's computer to print the document to a public printer, when connected to WiFi in an airport or hotel. We see here that many of the child nodes can be connected to multiple parents nodes, and even to other nodes in the same "level" of the tree.

In light of this discussion, we realize *this* attack tree is only a very basic representation of the possible threats to the root objective, however it successfully conveys the point that there are multiple ways of obtaining the investment plan, which may not have been obvious when first considering the situation.

Question 3

Estimate the entropy of passwords of length 10 for the following scenarios:

A: Passwords consist of lowercase characters only

There are 26 lowercase characters in English. Using the product rule, we see that there are:

$$26 \times 26 \times 26 \cdots \times 26 = 26^{10} = 141167095653376$$

possible passwords of length 10. This gives:

$$\lceil \log_2 26^{10} \rceil = \lceil \log_2(141167095653376) \rceil = \lceil 47.00439718141092 \rceil = \boxed{\text{48 bits of entropy}}$$

B: Passwords consist of lowercase and uppercase characters

There are 26 lowercase characters, and 26 uppercase characters in English. This gives a total of 52 possible characters. Using the product rule, we see that there are:

$$52 \times 52 \times 52 \cdots \times 52 = 52^{10} = 144555105949057024$$

possible passwords of length 10. This gives:

$$\lceil \log_2 52^{10} \rceil = \lceil \log_2(144555105949057024) \rceil = \lceil 57.00439718141092 \rceil = \boxed{\text{58 bits of entropy}}$$

C: Passwords consist of lowercase and uppercase characters, and also digits 0-9

There are 52 possible uppercase and lowercase characters, along with 10 different digits. This gives a total of 62 possible characters. Using the product rule, we see that there are:

$$62 \times 62 \times 62 \cdots \times 62 = 62^{10} = 839299365868340224$$

possible passwords of length 10. This gives:

$$\lceil \log_2 62^{10} \rceil = \lceil \log_2(839299365868340224) \rceil = \lceil 59.54196310386875 \rceil = \boxed{\text{60 bits of entropy}}$$

D: Passwords consist of lowercase and uppercase characters, digits, and also 11 symbols

Using uppercase, lowercase, and 10 digits gives 62 possible characters. Adding the 11 symbols gives at total of 73 possible characters. Thus, there are:

$$73 \times 73 \times 73 \times \cdots \times 73 = 73^{10} = 4297625829703557649$$

possible passwords of length 10. This gives:

$$\lceil \log_2 73^{10} \rceil = \lceil \log_2(4297625829703557649) \rceil = \lceil 61.89824558880017 \rceil = \boxed{\text{62 bits of entropy}}$$

Question 4

Compare the entropy calculations, from the previous question, to two tools. I chose to use the links provided in the assignment:

Tool 1: http://rumkin.com/tools/password/passchk.php
Tool 2: https://apps.cygnius.net/passtest/

Password Entropy Results

Password	**My Estimate**	**Tool 1 Estimate**	**Tool 2 Estimate**
helloworld	48 bits	36.4 bits	14.874 bits
HelloWorld	58 bits	44.2 bits	16.874 bits
He110W0r1d	60 bits	42.3 bits	20.874 bits
He!!0W0r1d	62 bits	43.8 bits	31.158 bits

General Notes on Using the Tools

Right from the start, we can see that the entropy calculation used by both tools is different than ours, because the entropies are non-integers. It is easy to think that perhaps they just didn't use the ceiling function, however a quick glance at site tells us that there is more involved with their calculations.

Tool 1 - http://rumkin.com/tools/password/passchk.php
Introduction of digits actually decreased the entropy, despite the fact that the number of possible characters increased.
Adding the symbols increased entropy from the case of uppercase, lowercase, and digits, however the entropy was still smaller than just upper and lower case characters.
Site states that it checks if you picked a common password and takes into account the probability of letters landing close to each other, based on letter pairs in the English language.

Tool2 - https://apps.cygnius.net/passtest/
Unlike Tool 1, increasing the character set has increased the entropy for all the examples in the table. After playing around with it for a bit, I noticed that the character set is not the only factor used to determine the entropy. For example, 'password123' has entropy of 4.585 bits while 'badpassword' has 8.7 bits. Clearly, there are more factors involved.
Site does not tell you how the entropy calculation is done, however it displays more detail than Tool 1 (which may or may not be useful information to the user).
There appears to be other factors considered when determining the password strength, such as cardinality and pattern recognition, or considering substitutions used, e.g. replacing the 'o' with '0'.

A: Describe the effect of password set size on the entropy of passwords

As previously mentioned, Tool 1 gave a bit of a surprise when calculating the entropy of the third password. We increased the number of available characters, yet the entropy was smaller than the case with only upper and lower case characters. Without knowing the exact algorithm used, we can only speculate, however it seems reasonable to assume this decrease in entropy can be attributed to the relatively common and straightforward substitution where 'o' was replaced with '0' and 'l' with '1'. If we think about it in a practical setting; simply using a larger number of characters should not guarantee that the entropy of our password will be larger. There are definitely passwords that use upper/ lower case, symbols, and digits, yet are relatively easy to guess, so it would only make sense to devise an algorithm that would calculate a smaller entropy for such passwords.

That being said; introducing a larger set of available characters allows us for more possible passwords (of a given length). Thus, it is *one* of the key components that needs to be considered when determining the strength of the password - which we want to be quantified by the entropy value. To drive this point home; set size is not the only factor that should be used to determine the entropy (as we did in our calculation), however it is without a doubt an important factor.

B: Describe the effectiveness of tools for password estimation

Before trying out these tools, I would have guessed that there there would be relatively little discrepancy between the entropy values determined by two different tools. This assumption was not because I expected the algorithms to be similar, however I thought that the factors considered would be more or less universal. This does not seem to be the case with the tools used here. The authors of each tool seem to place different emphasis on the factors involved. In theory, this is what makes the quantification of password strength a somewhat subject topic.

This shouldn't come as a surprise. An attacker can use a guessing-scheme or a brute force method attack. In both cases, the attacker is attempting to minimize the number of required guesses. The factors he/ she considers to be of the greatest importance are the same factors that are use by these password entropy calculators. This is where we see both the benefits and drawbacks of these tools. If only one tool is used, then perhaps an attacker can get lucky and focus on a factor that the tool did not consider. If we used multiple entropy calculators then we are increasing the chance that the factors considered in common attacks will be analyzed, and any weakness in our password will be exposed.

To apply this knowledge to my common passwords, it seems that both tools rate my passwords relatively low. There is definitely something to be said when two different tools, that clearly have much different algorithms for determining the entropy, come up with the same (poor) result - this was a very informative exercise!

Question 5

Consider a password system that uses password hashing for password verification. Each password consists of a string of 4 digits: $(a_3a_2a_1a_0)$, that is each a_i can be a digit $\{0, 1, 2, \ldots, 9\}$. The hash function is defined as:

$$h(a_3a_2a_1a_0) = ({a_3}^4 + {a_2}^3 + {a_1}^2 + a_0) \bmod 100$$

1. How many different passwords are possible in this system?

The number of possibilities for each digit is 10 and there are 4 digits. We are looking for the number of permutations, with repetition. The answer is given by the product rule rule,

$$\text{Number of Different Passwords } = 10 \times 10 \times 10 \times 10 = 10^4 = 10000$$

2. Calculate $h(7819)$

$$h(7819) = (7^4 + 8^3 + 1^2 + 9) \bmod 100 = (2401 + 512 + 1 + 9) \bmod 100 = 2923 \bmod 100 = 23$$

The rest of the assignment will be presented using the PDF output form Jupyter Notebook.

A1_Q5

January 28, 2019

I will do parts 3, 4, and 5 together.

3. Find a password x such that $h(x) = h(7819)$ but $x \neq 7819$
4. How many different passwords will have hash value equal to $h(7819)$?
5. Suppose an attacker wants to access John's account using an online attack. What is the probability the attacker will guess John's password if no hashing is involved, and when hashing is involved?

```
In [1]: import numpy as np
```

First, we must implement the hash function

$$h(a_3a_2a_1a_0) = ({a_3}^4 + {a_2}^3 + {a_1}^2 + a_0) \bmod 100$$

```
In [2]: '''Hash Function without Salting'''
        def hash1(num):
            '''
            Function implements hash function defined in assignment
            Parameter, num: password to be hashed
            Returns hashed value of password
            '''
            a0 = num % 10
            a1 = int(num/10) % 10
            a2 = int(num/10**2) % 10
            a3 = int(num/10**3) % 10

            return (a3**4 + a2**3 + a1**2 + a0) % 100
```

```
In [3]: '''Calculate hash and print'''
        accepted_hash = hash1(7819)
        print('Software is looking for hash value =', accepted_hash )
```

```
Software is looking for hash value = 23
```

Next, we will iterate passwords that range from 0 to 9999 and compare their hash values to the accepted value of 23.

```
In [4]: password = 0 # Start iteration from 0
        # array to store passwords with matching hashes
        working_pswd_list = np.array([], dtype=int)

        while password < 10000: # iterate from 0-9999
            current_hash = hash1(password) # calculate hash value
            # integer comparison as hashed values are ints
            if current_hash == accepted_hash:
                # if a match, then add to array
                working_pswd_list = np.append(working_pswd_list, password)
                password += 1
            else: password += 1
        # verify that all elements of array have correct hash val
        for password in working_pswd_list:
            assert hash1(password) == 23
```

```
In [5]: print('#3. First password found with hash value equal to h(7819):', \
              working_pswd_list[0])

        print('\nArray containing all passwords with hash values equal to h(7819):')
        print(working_pswd_list)
```

```
#3. First password found with hash value equal to h(7819): 47

Array containing all passwords with hash values equal to h(7819):
[  47  146  236  607  616  623  827  832 1046 1145 1235 1479 1606 1615
 1622 1819 1826 1831 2007 2016 2023 2106 2115 2122 2467 2591 2780 3066
 3165 3259 3336 3538 3541 3651 3855 3929 3934 4083 4182 4364 4403 4412
 4566 4672 4748 4876 4962 5299 5387 5459 5589 5691 5776 5895 5985 6052
 6151 6243 6300 6502 6511 6627 6632 6793 6836 7046 7145 7235 7479 7606
 7615 7622 7819 7826 7831 8052 8151 8243 8300 8502 8511 8627 8632 8793
 8836 9275 9561 9743 9871 9958]
```

```
In [6]: print('\n#4. Number of Passwords with hash value equal to h(7819) =', \
                len(working_pswd_list))
```

```

#4. Number of Passwords with hash value equal to h(7819) = 90
```

5 - The probability of guessing John's password with no hashing can be found by considering his unique password is one possibility out of 10000 possible passwords, this gives:

```
In [7]: print('Probability of guessing password with no hashing = ', \
                (1.0/10**4) * 100, '%')
```

```
Probability of guessing password with no hashing =  0.01 %
```

The probability of guessing John's password using hashing is found by noting that there are 90 passwords that have the same hash value. Thus,

```
In [8]: print('Probability of Guessing with Hashing =', \
              100*len(working_pswd_list)/10000.0, '%' )

Probability of Guessing with Hashing = 0.9 %
```

6 Suppose the password system is used with a 2 digit salt $(s1, s0)$.The salt will be simply added to the hash value (integer addition) and (mod 100) operation will be used to make it into a 2 digit number. In other words, the hash function is now:

$$h(s_1s_0, a_3a_2a_1a_0) = (10s_1 + s_0 + {a_3}^4 + {a_2}^3 + {a_1}^2 + a_0) \bmod 100$$

Explain how adding salt affects the chance of success for an attacker who tries to guess the password. Explain your answer using the password 7819.

First, we must redefine the hash function so that it includes the salt.

```
In [9]: def hash2(salt, num):
            '''
            Function implements hash function (with salt) defined in assignment
            Parameters
                salt: hashing salt
                num: password to be hashed
            Returns hashed value of password
            '''
            s0, s1 = salt %10, int(salt/10) % 10
            a0 = num % 10
            a1 = int(num/10) % 10
            a2 = int(num/10**2) % 10
            a3 = int(num/10**3) % 10

            return (10*s1 + s0 + a3**4 + a2**3 + a1**2 + a0) % 100

In [10]: user_password2 = 7819
         salt = 39

         accepted_hash2 = hash2(salt, user_password2)
         print('Software is looking for hash value =', accepted_hash2)

Software is looking for hash value = 62
```

Just as before, we will iterate passwords that range from 0 to 9999 and compare their hash values to the accepted value of 62.

```
In [11]: password2 = 0 # start with password 0000
         # array to store passwords with matching hashes
         working_pswd_list2 = np.array([])
```

```
        while password2 < 10000:
            current_hash2 = hash2(salt, password2)
            if current_hash2 == accepted_hash2:
                working_pswd_list2 = np.append(working_pswd_list2, password2)
                password2 += 1
            else: password2 += 1
```

```
In [13]: print('Number of Possible Successful Hashes =', len(working_pswd_list2))
         print('Probability of Guessing with salting =', \
               100*len(working_pswd_list2)/((2**len(str(salt)))*10000.0), '%' )
```

```
Number of Possible Successful Hashes = 90
Probability of Guessing with salting = 0.225 %
```

We see, as expected, that the number of passwords with a hash value equal to h(7819) is the same as before. What has changed is that the probability of guessing the password has decreased. It is still larger than without hashing, however it is significantly smaller than hashing without the use of salt.

Salting increases the number of possible passwords by a factor of 2^b, for a salt length of b bits. Sure, the number of passwords with the correct hash value remains the same, but there are so many more passwords to guess from, that the attacker has a much smaller chance at accessing John's account.

References

[1] The Age. Hack to the future. www.theage.com.au. Accessed: Jan 21, 2019.

[2] CIA. The Farewell Dossier - Duping the Soviets. www.cia.gov. Accessed: Jan 21, 2019.

[3] Haroldjoseph Highland. Random Bits & Bytes. *Computers & Security*, 9(3):196–208, 1990.

[4] IMDb. The William Safire. www.imdb.com. Accessed: Jan 21, 2019.

[5] W. Safire. The Farewell Dossier. www.nytimes.com. Accessed: Jan 21, 2019.