Assignment - PSO 2

Purdue CS426 - Computer Security - Prof. Spafford

Harris Christiansen - February 7, 2018
christih@purdue.edu

Problem 1

Humans are said to be the weakest link in any security system. Give an example for each of the following: (30 pts)

1. a situation in which human failure could lead to a compromise of encrypted data

- An admin inadvertently leaks the private keys for some encrypted data, and does not realize, so they fail to change the encryption key when necessary. This would result in a compromise of the encrypted data.

2. a situation in which human failure could lead to a compromise of identification and authentication

- A customer service agent permits unauthorized access to a users account, because they feel some inclination to trust an attacker without proper identification. This is an example of social engineering.

3. a situation in which human failure could lead to a compromise of access control

- Sensitive data is not properly protected by the user (left unencrypted or unlocked and unattended), resulting in an malicious party obtaining possession of the data. This would result in access control restrictions being bypassed by the attacker.

Problem 2

What is a hash function and what are the 3 properties that make a hash function cryptographically secure? What is a message digest? (10 pts)

What is a hash function?

- A hash function is a quick routine which can take in input of any length and produce a fixed size, deterministic value. Thus, all input fed into a hash function will produce that same length output (with that given hash function), and feeding the same input into a hash function multiple times will always produce the same value (the hash).

What are the 3 properties that make a hash function cryptographically secure?

- 1. It must be infeasible to determine a message given it's hash value (except by brute force).
- Any small change to the message should result in a significantly different hash value (to ensure you can't speed up brute force).
- Hashes should be relatively evenly distributed and have few collisions; that is to say it should be very uncommon for two messages to produce the same hash value.

What is a message digest?

- Message digests are a category of hash functions that produce short hash values typically 128 to 160 bits in length. They are referred to as message digests because they can be used to quickly verify the contents of a message are correct and uncorrupted.

Problem 3

Why is it considered a good practice to salt and hash a password before storing it either in a file or database? In a large group of users, what information would be leaked when an attacker obtains access to the database if a password were only hashed and not salted? (20 pts)

Read the following for hints:

- http://blog.moertel.com/posts/2006-12-15-never-store-passwords-in-a-database.html
- https://learncryptography.com/hash-functions/password-salting

Why is it considered a good practice to salt and hash a password before storing it either in a file or database?

- Passwords should **NEVER EVER** be stored or used in plaintext anywhere in a codebase, file, or database. Thus, it is necessary to transport and store the password as a cryptographic version of itself. Strong cryptographic hashes are a good choice because they are quick and are designed to not be decodable. You can then validate a password by comparing the known hash value against the hash of the user input.
- Even though hashing is designed not to be decodable, it can still be brute-forced. For this reason, it is important to salt your strings before hashing them. By adding a complex salt value:
  - you make the hash take significantly longer to brute-force
  - you reduce the likelihood of the hash being in a database of known hash plaintext values, which can be found on many online hash decoding databases.

In a large group of users, what information would be leaked when an attacker obtains access to the database if a password were only hashed and not salted?

- All users with low security passwords (7 characters or less, containing common words/patterns, etc) would immediately have their passwords be available to the attacker. No passwords could be declared safe, and all credentials should be assumed leaked.

Problem 4

Describe the difference between a symmetric block cipher and a symmetric stream cipher. What are the strengths and weakness of each when it comes to resistance to cryptanalysis and speed? Explain. (20 pts)

Readings:

RC4 Stream Cipher: https://people.cs.clemson.edu/~jmarty/courses/Spring-2017/CPSC424/papers/RC4ALGORITHM-Stallings.pdf
Prohibiting RC4 Cipher Suites: https://tools.ietf.org/html/rfc7465
Block cipher mode of operation: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
The AES-CBC Cipher Algorithm and Its Use with IPsec: https://tools.ietf.org/html/rfc3602
Understanding Cryptography Lecture Series: https://youtu.be/2aHkqB2-46k

- A symmetric **block cipher** encrypts fixed-size blocks of data:
  - Strength: Efficient at encrypting/decrypting data of a known size that can easily be split into blocks.
  - Weakness: Since blocks are a fixed-size, they can only be encrypted/decrypted when you have the entire block (which is a larger size to obtain).
  - Weakness: Any data smaller than the block size has to be padded with empty data to be encrypted.
  - Weakness: Require slightly higher amounts of memory to store/compute the entire block
  - Strength: Can chain blocks together, such that each block cannot be decrypted without the block before it.
- A symmetric **stream cipher** encrypts data byte-by-byte:
  - Strength: Allows data to be encrypted/decrypted as soon as data is available, making it more efficient when working with data of unknown size or data in a steam.
  - Weakness: Difficult to implement - can be slow or buggy

Problem 5

(30 pts)

Describe how you can use an asymmetric and symmetric cipher together to efficiently share keys and encrypt a stream of data.

- A symmetric cipher using a single key to cipher and decipher data, while an asymmetric cipher uses two keys, one to cipher and another to decipher data.
- If streaming data at a high speed, it will be necessary to use a symmetric cipher to encrypt the stream.
- **Solution:** Party A should share their public key for their asymmetric cipher. Another Party B should create a key for the symmetric cipher, encrypt it with Party A's public key, and send it to Party A. Party A can then decrypt that message with their private key, and now both parties securely have the key for the symmetric cipher. They can both freely stream data encrypted with that symmetric cipher key, and decipher it.

Explain how your solution is efficient in eliminating external channels for sharing a secret key and why your solution is optimal for encrypting a large stream of data at high speed.

- Because the key for the stream is never made available publicly and was transferred securely, we can be sure that no external channels can access our data.
- Because the stream is encrypted using a fast symmetric cipher, is it optimal for handing a large stream of data at a high speed.

Where is this problem commonly faced in the real world?

- Streaming of online and other secure media. Many television channels, news agencies, security cameras, and other large streams of data are transferred between parties online and need to be kept completely secure.
- This is also how https is implemented, as it performs/handles many of use cases.

Problem 6

Read the following:

- Cryptanalysis Attack Models: https://en.wikipedia.org/wiki/Attack_model
- Read Links under Types of cryptographic attacks: http://www.crypto-it.net/eng/attacks/index.html
- (Think of encrypted network traffic for email. What might be some known plaintext that could be used to break encryption?)

In your own words describe the following: (50 pts)

Each of the following is an attack model for cryptographic algorithms, in which the attacker has access to a variety of sources of information.

Known Plaintext Attack (KPA)

- In this model, an attacker knows the plaintext for a limited number of cipher-texts.
- These known plaintext/cipher-text pairs can be used to determine patterns in how text in enciphered.
- For example, with the simple substitution cipher, each character maps to another unique character. (e.x. the character 'e' would always encipher to 'p'). A KPA could easily identify this cipher, determine the code, and decipher other messages.

Chosen Plaintext Attack (CPA)

- In this model, an attacker can encipher their own chosen (and thus known) plaintext, and has access to the resulting cipher-text.
- This attack can be used to explore how the cipher works, allowing further discovery of patterns which can be used to break the cipher.

Ciphertext Only Attack (COA)

- In this model, an attacker has accessed to a bunch of enciphered cipher-text, but does not know the plaintext for any of them.
- This attack is the most important to secure against, by ensuring you use strong cryptographic algorithms.

- This attack is extremely common, resulting from hacks and database dumps, which often contain all users enciphered passwords and data.

Chosen Ciphertext Attack (CCA)

- In this model, an attacker can choose their own cipher-text, and has access to the decrypted plaintext.
- This information can be used to reverse-engineer and determine how the encryption algorithm works, and if any information about the plaintext is exposed in the cipher-text.
- This attack is important for security professionals and analysts to test, to ensure the cryptographic algorithm is secure.

Problem 7

Read the following:

- Side-channel attack: https://en.wikipedia.org/wiki/Side-channel_attack
- TEMPEST: https://en.wikipedia.org/wiki/Tempest_(codename)
- Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses: https://link.springer.com/content/pdf/10.1007%2F3-540-44499-8_24.pdf

Answer the following: (60 pts)

Pick two side channel attacks that can be used against cryptography. Describe how the attacks can be successful and what countermeasures you can use against the attack.

- Side channel attacks target weaknesses or known information with the physical implementation of the computer system, rather than the cryptographic algorithm itself.
- Because cryptographic algorithms run on machines, the way those machines execute the algorithm can reveal information about what was performed. For example:
  - The execution time and breakdown of time spent throughout the algorithm can reveal how big the encrypted data is, and even properties about the data which was encrypted over time.
  - Because cryptographic algorithms rely on random numbers which are produced by the machine, those values can be measured which makes the cipher less strong
- Some machines may be susceptible to **cache attacks**, which could leak key encryption data such as the plaintext or intermediate values. To avoid this, algorithms should use secure environments for performing computation, and configure caching policies not to hold onto sensitive data.
- **Timing attacks** can be used to the determine the length of the encrypted data, revealing information about the plaintext. This can be mitigated by making efforts to have consistent time spent on the algorithm, and avoiding attackers obtaining timing information.

From Weingart's article, pick two high technology attacks. Describe how the attacks can be successful and what countermeasures you can use against the attack.

- **Passive Probe Attacks** can be used to observe signals being executed, which can reveal the data itself, or information about the data (such as described above). This could expose key cryptographic secrets, such as the plaintext, key values, or patterns in these values.

  - They can be defended against by minimizing the variance of how the cipher executes given a variety of inputs. They can also be defended against by physical hardware intended to prevent unauthorized monitoring/probing.

- **Clock Glitching** can be used to change how the program behaves, sometimes in disastrous or unpredictable ways. As a result of shortening/lengthening the clock pulses, instructions can be skipped or processed unpredictably. This can cause cryptographic algorithms to run incorrectly, failing to produce a secure result.
  - This can be mitigated against by performing checks along the way and making sure data contains what it is supposed to. It can also be mitigated against by clock glitching resistant hardware (often realized using RF shields).

Problem 8

Frequency analysis on cipher text to recover plaintext: (100 pts)

file.txt frequencies

Singles: {'E': 258229, 'T': 189986, 'O': 180049, 'A': 166424, 'I': 144829, 'S': 143527, 'N': 141334, 'R': 138833, 'H': 137142, 'L': 97675}

Doubles: {'TH': 70002, 'HE': 48123, 'AN': 35387, 'ER': 35191, 'OU': 32506, 'IN': 32205, 'HA': 27774, 'ES': 27391, 'ND': 27240, 'ST': 27013}

Triples: {'THE': 32077, 'AND': 19008, 'YOU': 12718, 'HER': 10821, 'HAT': 10799, 'THA': 10611, 'ING': 10415, 'OUR': 9309, 'ETH': 9170, 'HIS': 8593}

ciphertext.txt frequencies

Singles: {'F': 110, 'X': 74, 'G': 59, 'S': 58, 'J': 58, 'P': 53, 'D': 44, 'I': 37, 'U': 36, 'T': 29}

Doubles: {'XJ': 31, 'JF': 25, 'JG': 18, 'FP': 17, 'FI': 17, 'GX': 17, 'PF': 16, 'SP': 12, 'SX': 11, 'ZF': 11}

Triples: {'XJF': 18, 'XJG': 9, 'JFP': 9, 'JGX': 8, 'FIU': 7, 'FPF': 7, 'DSX': 6, 'GZF': 6, 'VSP': 6, 'GXF': 5}

Discovered Plaintext

BUTINALARGERSENSEWECANNOTDEDICATEWECANNOTCONSECRATEWECAN
NOTHALLOWTHISGROUNDTHEBRAVEMENLIVINGANDDEADWHOSTRUGGLED
HEREHAVECONSECRATEDITFARABOVEOURPOORPOWERTOADDORDETRACTTH
EWORLDWILLLITTLENOTENORLONGREMEMBERWHATWESAYHEREBUTITCANN
EVERFORGETWHATTHEYDIDHEREITISFORUSTHELIVINGRATHERTOBEDEDICATE
DHERETOTHEUNFINISHEDWORKWHICHTHEYWHOFOUGHTHEREHAVETHUSFA
RSONOBLYADVANCEDITISRATHERFORUSTOBEHEREDEDICATEDTOTHEGREATTA
SKREMAININGBEFOREUSTHATFROMTHESEHONOREDDEADWETAKEINCREASED
DEVOTIONTOTHATCAUSEFORWHICHTHEYGAVETHELASTFULLMEASUREOFDEV
OTIONTHATWEHEREHIGHLYRESOLVETHATTHESEDEADSHALLNOTHAVEDIEDIN
VAINTHATTHISNATIONUNDERGODSHALLHAVEANEWBIRTHOFFREEDOMANDT
HATGOVERNMENTOFTHEPEOPLEBYTHEPEOPLEFORTHEPEOPLESHALLNOTPERI
SHFROMTHEEARTHABRAHAMLINCOLNNOVEMBER

Replacement Map

{'F': 'E', 'X': 'T', 'G': 'A', 'S': 'O', 'J': 'H', 'P': 'R', 'D': 'N', 'I': 'D', 'U': 'I', 'T': 'L', 'O': 'S', 'N': 'W', 'B': 'C', 'Z': 'V', 'V': 'F', 'Y': 'U', 'A': 'G', 'H': 'B', 'W': 'M', 'R': 'P', 'K': 'Y', 'C': 'K'}

Code Explanation

- getPlaintext() / getCiphertext() - Read the files
- formatText(text) - Return only [A-Z] in uppercase text
- Getting Frequencies: I split the text up into individual groups of single, double, and triple characters for the entire text. I then use a dictionary to count the number of occurrences of each.
- Deciphering: I use frequency analysis and simple substitution to determine which characters are which, and replace them. It was obvious that 'F' was 'E' and 'X' was 'T', but it was necessary to use data for the rest. Using the Shakesphere training data along with known top occurrences in the english language, it was possible to map all the characters.

Code

Attached after question 9, and in file frequency_analysis.py

The code can additionally be found at: http://files.harrischristiansen.com/0h2R2Y1h0l2G

Problem 9: Extra Credit

Read [The Library of Babel](https://web.archive.org/web/20171027213619/https://hyperdiscordia.church/library_of_babel.html): https://web.archive.org/web/20171027213619/https://hyperdiscordia.church/library_of_babel.html (20 pts)

Write a short explanation about how Borge's story is linked to Shannon's perfect security.

- In the world of Shannon's perfect security, the probability that the plaintext is x, given an observed cipher-text y, is the same as the probability the plaintext is x (without an observed cipher-text y).
- Thus, Borge's story is linked in that in order to find the secret (the hidden book), you must find infinitely many books (secrets) beforehand. Thus, no single book increases your probability of finding the secret.

How many books are in the library (use the author's description to calculate the number, if you can)?

- Each hexagonal gallery has 20 shelves, each of standard size, (which I assume to be approximately 36" x 84" x 18").
- I estimate ~130 books can fit on each shelf, thus 2600 books can fit in each gallery.
- There are "indefinite and perhaps infinite" galleries
- Total, there are infinite books in the library

Problem 8: Frequency Analysis Source Code

```
'''
    @ Harris Christiansen (Code@HarrisChristiansen.com)
    File Created: February 2018
    Purdue CS426 Computer Security - PSO 2 - https://github.com/harrischristiansen/cs426_pso2
    Problem 8 - Frequency Analysis
'''

import re, string

TEXT_DENY_PATTERN = re.compile('[^a-zA-Z]+')
NUM_TOP_FREQS = 10

################# Getting Text #################

def getPlaintext():
    with open("file.txt","r") as f:
        return f.read()

def getCiphertext():
    with open("ciphertext.txt","r") as f:
        return f.read()

def formatText(text): # Return only [A-Z] in uppercase text
    justLetters = TEXT_DENY_PATTERN.sub('', text)
    return justLetters.upper()

################# Single Character Frequency Analysis #################

def getFrequencies(items):
    counts = {}
    for item in items:
        value = ''.join(item)
        if value in counts:
            counts[value] += 1
        else:
            counts[value] = 1
    return dict(sorted(counts.items(), key=lambda k: counts[k[0]], reverse=True))

def getSinglesFrequencies(text):
    return getFrequencies(list(text))

################# Double Character Frequency Analysis #################

def getDoubles(text):
```

```
    doubles = []
    for i, v in enumerate(text):
        try:
            double = (v, text[i + 1])
            doubles.append(double)
        except IndexError:
            return doubles
    return doubles

def getDoublesFrequencies(text):
    doubles = getDoubles(text)
    return getFrequencies(doubles)

################# Triple Character Frequency Analysis #################

def getTriples(text):
    triples = []
    for i, v in enumerate(text):
        try:
            triple = v, text[i + 1], text[i + 2]
            triples.append(triple)
        except IndexError:
            return triples
    return triples

def getTriplesFrequencies(text):
    triples = getTriples(text)
    return getFrequencies(triples)

################# Complete Frequency Analysis #################

def getTopFrequencies(text, count=NUM_TOP_FREQS):
    singles = dict(list(getSinglesFrequencies(text).items())[:count])
    doubles = dict(list(getDoublesFrequencies(text).items())[:count])
    triples = dict(list(getTriplesFrequencies(text).items())[:count])

    return {
        'singles': (singles),
        'doubles': (doubles),
        'triples': (triples),
    }

def printTopFrequencies(text="", freqs=None):
    if freqs == None:
        freqs = getTopFrequencies(text)

```

```
    print("Singles: %s" % freqs["singles"])
    print("Doubles: %s" % freqs["doubles"])
    print("Triples: %s" % freqs["triples"])

################# Decipher #################

def getListSingleFreqs(text):
    freqs = getSinglesFrequencies(text)
    return list(freqs.keys())

def createFreqReplacementMap(source_freqs, target_freqs):
    replacements = {}
    for i, freq in enumerate(source_freqs):
        replacements[freq] = target_freqs[i]
    return replacements

def replaceBySingleFreq(text, target_freqs):
    freqs = getListSingleFreqs(text)
    replacements = createFreqReplacementMap(freqs, target_freqs)
    #print(replacements)

    result = ""
    for c in text:
        result += replacements[c]

    return result

################# Main #################

if __name__ == '__main__':
    plaintext = formatText(getPlaintext())
    #printTopFrequencies(plaintext)

    ciphertext = getCiphertext()
    #printTopFrequencies(ciphertext)

    #target_freqs = getListSingleFreqs(plaintext) # ['E', 'T', 'O', 'A', 'I', 'S', 'N', 'R', 'H', 'L', 'D', 'U', 'M', 'Y', 'C', 'W', 'F', 'G', 'B', 'P', 'K', 'V', 'X', 'J', 'Q', 'Z']
    target_freqs = ['E', 'T', 'A', 'O', 'H', 'R', 'N', 'D', 'I', 'L', 'S', 'W', 'C', 'V', 'F', 'U', 'G', 'B', 'M', 'P', 'Y', 'K']
    deciphered_text = replaceBySingleFreq(ciphertext, target_freqs)
    print(deciphered_text)
```