6.857 Homework	**Problem Set 1**	# 1-1 - An edX Security Policy
Erica Du		
Skanda Koppula		
Jessica Wang		February 18, 2015

1 Objective

The purpose of this security policy is to outline the security goals and usages of a massive open online course platform (MOOC) like edX. In this document, we define policies regarding user roles and their permissions, as well as privacy and sharing of data.

2 Introduction

EdX is a non-profit, open source MOOC platform that aims to provide educational content to worldwide users. Users sign up on edX and gain access to online courses in a wide range of subjects, administered by accredited universities and organizations around the world. After successful completion of a course, a user may achieve a certificate of completion.

Courses, run by universities, are administered by staff that handle the operations of the course. These staff members control course content and the student experience.

3 Principals and Authorized Behavior

The platform has a hierarchy of roles. They are, in order: edX employees, universities, course staff, students, and everyone. Each have their own set of allowed functions:

3.1 edX employees

3.2 Universities

3.3 Course staff

(Instructors and TAs): authorize student enrollment into their course, authorize other instructors to co-administer their course, modify course meta-data (name, description) and content, send signed feedback to student, view the submissions of student enrolled in their course(s), modify their own profile information (email, login username, password), add new course, and award certificate of completion to select student users.

3.4 Students

A student must be able to request enrollment into a course, view content for an enrolled courses (to the extent allowed by course instructor(s)), submit digitally signed answers and feedback to (enrolled) course instructor, modify their own profile information (login username, password, email), and receive certificate of completions as determined by the instructor.

Honor Code Certificate Candidate
Verified Certificate Candidate

3.5 Everyone

4 Authentication

5 Security Goals

Out of the 3 general security goals, integrity is the most relevant and crucial for edX. Since edX-type platforms are metaphorical to a traditional classroom, courses need to maintain the integrity of their data, and unauthorized users such as students or administrators of other courses should not be able to modify a specific course's data. Confidentiality is another important goal - the information of edX users should not be compromised to unauthorized parties.

5.1 Integrity

Specific security goals that fall under integrity include the following.

- Ensuring that anyone with permissions of or lower than that of a student cannot change another student's data or course data.
- Ensuring that course staff cannot change data of courses they are not administrators of.
- Ensuring that students cannot change their own data in an abnormal way, e.g. unauthorized modification of test scores.
- Ensuring that students earn completion certificates honorably.

5.2 Confidentiality

- only students and teachers should be able to see grades

5.3 Availability

- not as pressing as say, a hospital service, not endangering lives

- everything/permissions should be maintained even if service becomes unavailable

Confidentiality/Integrity Details:: Specifically, by use of some security mechanism (perhaps, OAuth 2.0 or like), we intend that the platform forbids the student viewing or modify the content and progress of other users (except to view the content of their enrolled courses as allowed by their course instructors). The student should not be able to obtain completion certificates without proper authentication.

The teacher should not be able to view the submissions of students not currently in their course. They should not be able to update the content of the courses that they were not authorized to administer.

6.857 Homework	**Problem Set 1**	# 1-2 - Reused Pad Cipher
Erica Du		
Skanda Koppula		
Jessica Wang		February 18, 2015

A. They were encrypted using the same pad. The two words are ADVERSARIAL AND MATHEMATICS. We used the following script:

```
CT1_str = 'd2 6b a5 0d 27 6a 34 2d 8e 53 0e'
CT2_str = 'de 6e a7 00 30 74 34 2b 8e 51 11'

CT1 = [int(byte, 16) for byte in CT1_str.split(' ')]
CT2 = [int(byte, 16) for byte in CT2_str.split(' ')]

def xor(xs, ys):
    '''Perform pairwise XOR operation on two lists'''
    return [x ^ y for x, y in zip(xs, ys)]

X = xor(CT1, CT2)

dictionary = open('dictionary', 'r').read().split()
words = set()
for w in dictionary:
    if len(w) == len(X):
        try:
            words.add(w.upper().encode())
        except:
            print 'Problem encoding', w

for PT1_str in words:
    PT1 = [ord(byte) for byte in PT1_str]
    PT2 = xor(PT1, X)
    PT2_str = "".join([chr(byte) for byte in PT2])
    if PT2_str in words:
        pad = xor(PT1, CT1)
        print('PT1 = %s, PT2 = %s, pad = %s' % (PT1_str, PT2_str, pad))
```

B. There are four messages and three unique pads. Therefore at least two messages will have the same two pads applied to them ($\binom{3}{2} = 3$). If we XOR those two pads together, the pads will cancel. Once we obtain the pad-canceled XOR of two messages, we peel apart the two messages via crib-dragging.

First, to identify which of the six possible pairwise combinations have the pads cancel out, we compute all six and count the frequency of '0' in the hex representation of each cross. All lowercase ascii [a-z] share the same last four bits, so the XOR of two plaintext paragraphs will contain a signature with plenty of zeroes.

Now we try peeling apart the two messages via crib-dragging. We took a few common words like ' the ' or ' is ' and XOR'd them along every position in our message product. If the common word is in that position in one of the messages, bits of the other message will be revealed. We extrapolate and guess words from those bits, and repeat the process. `grep` us helped extrapolate possibilities for words given the dictionary. We were also thinking of XOR'ing big words and testing for resulting smaller words in the other message.

We wrote the following interactive script to speed up and automate entering a crib, XOR'ing along the positions, and choosing the correct position (if any).

```
import itertools
import sys
import re

def read_ciphertexts(filename):
    '''Reads ciphertexts from file and removes spaces'''
    ciphers = open(filename, 'r').read()
    ciphers = ciphers.split('\n')[::2]
    return [text.replace(' ', '').decode('hex') for text in ciphers]

def get_dictionary(filename):
    '''Read dictionary file and returns set of words'''
    dictionary = open(filename).read()
    return set(ciphers.split('\n'))

def is_int(i):
    '''Tests if variable is valid integer value'''
    try:
        int(i)
        return True
    except ValueError:
        return False

def str_xor(str1, str2):
    '''Perform pairwise XOR operation on two strings'''
    return ''.join(chr(ord(x) ^ ord(y)) for x,y in zip(str1,str2))

def get_cancelled_pad(ciphertexts):
    '''Performs pairwise XOR's between ciphertexts
    and determines which cross has the pad canceled out
    by frequency of zeroes'''

    best_i, best_j, best_freq = 0, 0, 0
    for i, text1 in enumerate(ciphertexts):
        for j, text2 in enumerate(ciphertexts):
            if i == j: continue
            count = sum([1 if c == '0' else 0 for c in str_xor(text1, text2).encode('hex')])
            if count > best_freq:
                best_i, best_j, best_freq = i, j, count
```

```
    print 'Using ciphertext cross with', best_i, 'and', best_j
    return str_xor(ciphertexts[best_i], ciphertexts[best_j])

def peel_apart_messages(canceled_pad, charset):
    '''Interactive function for user to guess crib repeatedly
        and make it easy to peel apart two messages'''

    message1 = '_'*len(canceled_pad)
    message2 = '_'*len(canceled_pad)

    while True:
        print 'Current Message 1:'
        print message1
        print 'Current Message 2:'
        print message2

        crib = raw_input("Enter crib: ")

        num_positions = len(canceled_pad) - len(crib) + 1
        results = [0]*num_positions
        for i in xrange(num_positions):
            pos_result = str_xor(canceled_pad[i:i+len(crib)], crib)
            results[i] = pos_result
            if re.search(charset, pos_result):
                sys.stdout.write(str(i)+': '+pos_result+'*''\t\t')
            else:
                sys.stdout.write(str(i)+': '+pos_result + '\t\t')
            if i%3 == 0: sys.stdout.write('\n')

        response = raw_input("Enter the position, 'n' for no match, or 'q' to quit: ")
        if response == 'none' or response == 'n':
            print 'No changes made'
        elif is_int(response) and int(response) < num_positions:
            response = int(response)
            one_or_two = raw_input("Is crib part of message 1 or 2? '1' or '2': ")
            if one_or_two == '1':
                message1 = message1[:response] + crib + message1[response+len(crib):]
                print results[response]
                message2 = message2[:response] + results[response] + message2[response+len(cri
            elif one_or_two == '2':
                message2 = message2[:response] + crib + message2[response+len(crib):]
                message1 = message1[:response] + results[response] + message1[response+len(cri
            else:
                print 'Invalid input!'
        elif response != 'quit' or response != 'q':
            break
        else:
            print 'Invalid input'
```

```
ciphertexts = read_ciphertexts('ciphers.txt')
canceled_pad = get_cancelled_pad(ciphertexts)

charset = '^[a-zA-Z0-9.,?! :;\'"]+$'
peel_apart_messages(canceled_pad, charset)
```

From this we discovered that we were able to descrypt parts of the **first** and the fourth messages.
The two messages:

```
Current Message 1:
_____________________ commander __________break every message, every____, instantly   ___
Current Message 4:
_____________________tHat will o__________ is the most difficult pro____ in the world ___
```

6.857 Homework	**Problem Set 1**	# 1-3 - Subverting Cryptography
Erica Du		
Skanda Koppula		
Jessica Wang		February 18, 2015

1. Insecure implementation: the implementation of a theoretically secure algorithms might have vulnerabilities

   - The OpenSSL library this year was revealed to have a buffer over-read bug, Heartbleed.
   - The NSA has inserted faulty code/backdoors in commonly used random number generators.

   DEFENSE: Do not use 3rd party code. Implement everything, like random number generators and even basic algorithms, on your own.

2. Improper use: users may misuse otherwise secure software and/or algorithms

   - Users may create short or weak keys
   - Users may re-use passwords

   DEFENSE: Create strong restrictions for user inputs to ensure strong key creation.

3. Hardware injection: tampering with physical devices

   - Keyloggers to track keystrokes and potentially passwords
   - Inserting backdoors into internet routers

   DEFENSE: Only use devices from a trusted resource, only use devices you create yourself.

4. Compulsion: Legal Retrieval by Government

   - Court orders to obtain private keys

   DEFENSE: Host data in a different country, start your own country without these rules. Get a good lawyer.

5. Social engineering: extracting personal information to gain useful information

   - Email or phone phishing

   DEFENSE: Adblock. Spam filters. Common Sense.

6. Indirect Computational Data

   - Using metadata of a message to learn information about the message
   - Timing attacks (i.e. side channel timing attacks) that determine message based on how long it takes per step of computation

   DEFENSE: Introduce more randomness (i.e. in length) into metadata and actual message.

7. Coercion: bribery and corruption

   - Pay NSA employees more than the government to spill secrets

- Give money in exchange for information

DEFENSE: Pay your employees a sufficient wage. Only hire people you trust.

8. Subversion: abuse trust commercial sector has in a body like the NSA **DEFENSE:** Don't put your trust in governmental bodies, or other bodies of "authority."