

- Anunțuri
- Bune practici
- Calendar
- Catalog
- Feed RSS
- IOCLA Need to Know
- Reguli și note
- Resurse utile

Cursuri

- Curs 01: Introducere
- Curs 01 - 02: Arhitectura sistemelor de calcul
- Curs 02 - 03: Arhitectura x86
- Curs 04: Reprezentarea datelor în sistemele de calcul
- Curs 05: Reprezentarea datelor în sistemele de calcul - C2
- Curs 06 - 07: Setul de instrucțiuni
- Curs 07: Declararea datelor
- Curs 08 - 09: Moduri de adresare
- Curs 09: Stiva
- Curs 10 - 11: Funcții
- Curs 12: C + asm
- Curs 13: Unelte, utilitare
- Curs 13 - 15: Buffer overflows, securitate
- Curs 16 - 17: Optimizări
- Curs 18 - 19: Virgulă flotantă

Laboratoare

- Laborator 01: Introducere
- Laborator 02: Toolchain
- Laborator 03: First baby steps
- Laborator 04: Rolul registrelor, adresare directă și bazată
- Laborator 05: Structuri, vectori. Operații pe siruri
- Laborator 06: Lucrul cu stiva
- Laborator 07: Apeluri de funcții
- Laborator 08: Interacțiunea C-assembly
- Laborator 09: Analiza statică și dinamică a programelor. GDB
- Laborator 10: Gestiunea bufferelor. Buffer overflow
- Laborator 11: Optimizări
- Laborator 12: Calcul în virgulă mobilă



Teme

- Tema 1 - Prefix AST
- Tema 2 - Stegano
- Tema 3 - Exploit ELFs, not elves

Table of Contents

- Tema 2 - Stegano
 - Enunț
 - Structură și detalii de implementare
 - 1. Bruteforce pe XOR cu cheie de un octet - 15p
 - 2. Criptare folosind XOR cu cheie predefinita - 10p
 - 3. Criptarea unui mesaj folosind Codul Morse - 15p
 - 4. LSB - 20p
 - 5. Decriptare LSB - 15p
 - 6. Aplicarea unui filtru pe imagine - 15p
 - Precizări suplimentare
 - Trimitere și notare
 - Resurse

Tema 2 - Stegano

- Deadline **HARD: 10.12.2019**
- Data publicării: 23.11.2019
- Responsabili:
 - Dorin Andrei GEMAN
 - Bogdan-Cristian FIRUȚI

Enunț

După o noapte de LANParty în facultate, ați descoperit o cameră secretă unde se aflau mai multe dispozitive ciudate. V-ați speriat de vocea paznicului care rasuna la 12 noaptea pe holul facultății și ați decis să plecați din acea încăpere, însă nu cu mâna goală. Ați luat un dispozitiv și chiar acum încercați să descoperiți care este rolul său. Din ce v-ați dat voi seama, dispozitivul nu este foarte vechi și pare să ruleze un program scris în ASSEMBLY. Programul primește o imagine ca input și afișează niște mesaje secrete sau niște mesaje în clar și o imagine și realizează o nouă imagine în care sunt incluse mesajele voastre. Totodată, în funcție de un anume parametru, schimbă tipul codificării. Spiritul competitiv v-a făcut să vă întreceti și de data aceasta în scrierea unui parametru asemanator cu cel descoperit, care să realizeze aceleași codificări/decodificări. Această metodă de criptare începe să devină din ce în ce mai populară deoarece puțini s-ar aștepta ca în interiorul unei imagini să se găsească un mesaj ascuns.

Structură și detalii de implementare

Tema este formată din mai multe subpuncte, fiecare subpunct constând fie în codificarea unui mesaj folosind o anumită metodă de criptare și introducerea sa în imagine, fie în decodificarea unei imagini folosind o anumită metodă. Subpunctele pot fi rezolvate independent, însă puteți refolosi fragmente din rezolvarea unui subpunct în rezolvarea altor subpuncte acolo unde considerați necesar.

1. Bruteforce pe XOR cu cheie de un octet - 15p

În mod uzual în criptografie, dimensiunea cheii de criptare va fi mai mică decât a datelor de intrare. Există mai multe mecanisme care permit folosirea unei chei mai scurte decât mesajul pentru a realiza criptarea. Cel mai simplu dintre acestea reprezintă construirea unei noi chei prin repetarea cheii actuale până se ajunge la dimensiunea necesară. Vulnerabilitatea acestei abordări constă în faptul că este suficientă pentru un potențial atacator obținerea unui substring (din mesajul decriptat) de lungime mai mare sau egală decât cheia inițială pentru a putea determina cheia prin brute force.

Pentru a exemplifica, vom folosi o cheie de un octet din care vom obține prin repetare o cheie de dimensiunea mesajului. Criptarea se va face apoi prin XOR între mesaj și cheia rezultată (operația XOR e reversibilă, adică x XOR y XOR y = x).

Pentru acest subpunct, funcția `bruteforce_singlebyte_xor` care trebuie implementată va primi imaginea și trebuie să afișeze cheia folosită în criptarea mesajului, linia la care a fost găsit mesajul, precum și mesajul decriptat în-place. Considerăm cheia pe un octet, iar voi trebuie să aplicați XOR între fiecare pixel din imagine și cheia. Folosiți bruteforce, adică încercați toate numerele care pot intra pe un octet, și căutați mesajul criptat după aplicarea cheilor. Prototipul acesteia este următorul:

```
int bruteforce_singlebyte_xor(int* img);
```

Output-ul va trebui să fie de forma:

```
original_message
key
line
```



Hint 0: mesajul decriptat este în limba franceză, conține "revient" și se află la începutul liniei. Va trebui să căutați pe fiecare linie până întâlniți caracterul cu valoarea 0 (terminator de string) sau până la finalul ei (dacă nu găsiți terminatorul).

Hint 1: la exercițiul următor veți avea nevoie de cheia și linia obținute aici, deci ar fi util să le salvați pe ambele în EAX

2. Criptare folosind XOR cu cheie predefinita - 10p

La punctul 1 am simulat primirea unui mesaj codificat cu o cheie neștiută de destinatar. Dupa cum probabil v-ați dat seama, cheia de la punctul 1 a fost folosită pe întreaga matrice de pixeli. (Persoana cu care comunicăm și-a ales o cheie și o linie, pe care le-ați aflat la punctul 1, a înlocuit valorile pixelilor de pe acea linie din matrice cu valoarea fiecărui caracter din mesajul său, iar apoi a aplicat cheia sa pe întreaga matrice de pixeli folosind operația XOR.)

Pentru a putea comunica cu prietenul nostru, va trebui să aplicați cheia găsită la punctul 1 pe întreaga matrice. Astfel, veți obține imaginea originală în care el și-a introdus mesajul. Trebuie să vă adăugați mesajul pe linia care urmează după mesajul său, iar apoi să criptați întreaga matrice de pixeli cu cheia voastră. Astfel, dacă interlocutorul va face și el bruteforce, când va găsi cheia voastră, va putea să își vadă atât mesajul său, cât și mesajul vostru. Cheia pe care trebuie să o folosiți la criptare o veți obține folosind următoarea formulă:

```
key = floor((2 * old_key + 3) / 5) - 4
```

Mesajul pe care dorim să îl transmitem este următorul:

```
C'est un proverbe français.
```

3. Criptarea unui mesaj folosind Codul Morse - 15p

Metoda a apărut în secolul 19 și era folosită pentru a trimite mesaje cu ajutorul telegrafului. A devenit foarte utilă în perioada războaielor, perioadă în care comunicațiile trebuiau să fie rapide și greu de interceptat/decriptat de către inamici. Codul Morse conține doar 4 tipuri de caractere: ".", "-", " " și "|". ""reprezintă un semnal scurt, "-" reprezintă un semnal mai lung, " " delimitează literele, iar "|" delimitează cuvintele.

La acest subpunct veți primi o imagine nealterată, un mesaj și un indice. Va trebui să criptați mesajul folosind caracterele speciale descrise mai sus (".", "-", " " și "|") și să adăugați șirul obținut în imagine prin înlocuirea valorilor pixelilor, începând de la indicele primit. Mesajul și indicele octetului se vor gasi în `argv[3]` și `argv[4]`.

Mesajul va fi codificat precum în imaginea următoare, folosind caracterele "." și "-". Caracterul "." va fi codificat prin "-.-.-" (2x dash, 2x dot, 2x dash). Veți pune " " (blank) după fiecare literă scrisă.

International Morse Code

- The length of a dot is one unit.
- A dash is three units.
- The space between parts of the same letter is one unit.
- The space between letters is three units.
- The space between words is seven units.

A	••	U	•••
B	••••	V	••••
C	•••••	W	••••
D	•••••	X	••••
E	•••	Y	••••
F	••••	Z	••••
G	••••		
H	••••		
I	•••		
J	••••		
K	••••	1	•••••
L	••••	2	•••••
M	••••	3	•••••
N	••••	4	•••••
O	••••	5	•••••
P	••••	6	•••••
Q	••••	7	•••••
R	••••	8	•••••
S	••••	9	•••••
T	••••	0	•••••

Exemplu:

```
Mesajul: "Ion, 22"
Codificarea: "... --- ..-.-.- | ..-.-. ...."
```

Prototipul funcției va trebui să fie următorul.

```
void morse_encrypt(int* img, char* msg, int byte_id);
```



Literele folosite în mesaj sunt toate uppercase.

4. LSB - 20p

Least Significant Bit este o tehnica de ascundere a unui text într-o imagine. Procedeu constă în luarea fiecărui bit din reprezentarea binară a textului ce se vrea ascuns și pus în locul celui mai nesemnificativ bit a mai multor octeți consecutivi. Voi va trebui să scrieți o funcție ce va primi ca parametri o imagine, un mesaj și un indice al unui byte de la care sa incepeti encodarea propriu-zisa. Mesajul si indicele octetului se vor gasi în `argv[3]` si `argv[4]`. Functia va trebui sa ascunda textul si sa afiseze noua imagine la stdout.

```
void lsb_encode(int* img, char* msg, int byte_id);
```

Exemplu:

```
"Bless0" Se transforma în format binar ->
01000010 01101100 01100101 01110011 01110011 00000000
Avem nevoie de 8x6 octeți pentru a stoca mesajul.
Daca primii 3 octeți au valorile(interpretate ca numere întregi):
101 130 150
Acestea vor deveni:
100 131 150
Primul bit din mesajul nostru este 0 -> 101 devine 100 pentru ca punem 0 pe LSB.
Al doilea bit din mesajul nostru este 1 -> 130 devine 131 pentru ca punem 1 pe LSB.
Al treilea bit din mesajul nostru este 0 -> 150 devine tot 150 pentru ca punem 0 pe LSB.
```

5. Decriptare LSB - 15p

Acum ca stim sa ascundem mesaje in imagini folosind LSB, vrem sa putem si extrage texte din imagini.

Pentru acest task va trebui sa extragem un mesaj pe care stim ca l-am primit prin intermediul unei imagini, incepand cu un anumit byte, si sa il afisam la stdout. Lungimea mesajului nu va depasi 20 de caractere.

Antetul funcției va fi următorul:

```
void lsb_decode(int* img, int byte_id);
```

Hint: Nu se cunoaste lungime textului. Ne vom opri la intalnirea a 8 LSBiti cu valoarea 0 (terminator de sir).

6. Aplicarea unui filtru pe imagine - 15p

Pentru a se asigura că mesajele sunt și mai greu de decodificat, unii oameni folosesc o metodă puțin mai ingenioasă. După ce adaugă mesajele criptate prin cât mai multe metode în imagine, aplică un filtru pe ea pentru a fi și mai greu de obținut.

În tema aceasta va trebui să aplicați filtrul *BLUR*. Pentru a calcula noua valoare a unui pixel, se va calcula suma dintre valoarea curentă a pixelului și pixelii vecini (sus, jos, stânga, dreapta) și se va împărți la numărul valorilor(5). Se va pastra doar catul din rezultatul mediei aritmetice.

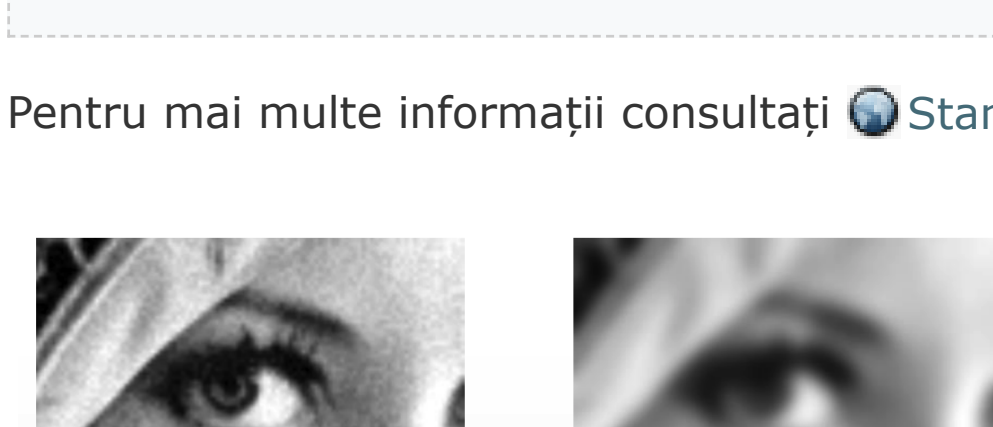
Pentru simplitate, vom aplica filtrul doar pentru pixelii ce nu sunt pe laturile imaginii. Functia va primi ca parametru o imagine si va afisa la stdout imaginea blurata.

```
void blur(int* img);
```

Exemplu:

```
24 87 16 72 51
73 24 31 25 21
59 89 15 89 75
87 74 69 62 55
45 54 94 13 53
- after blur -
24 87 16 72 51
73 60 22 47 21
59 52 58 53 75
87 74 62 57 55
45 54 94 13 53
```

Pentru mai multe informații consultați  Stanford,  ImageFiltering sau  ImageProcessing.



Precizări suplimentare

Toate variabilele, datele, rezultatele parțiale, în afară de cele deja declarate, trebuie stocate pe stivă, nu în secțiunea de date.

Aveți voie (și se recomandă) să implementați oricâte funcții adiționale care să vă ajute în rezolvarea mai multor task-uri, însă structura din main (în sensul de apeluri de funcții) trebuie să rămână neschimbată (excepție făcând rezervarea de spațiu pe stivă pentru variabile auxiliare, și apeluri de funcții auxiliare).

Scheletul de cod deschide fișierul "inputX.pgm", unde X este un parametru de intrare primit de către program ce determină task-ul testat. Fișierul `inputX.pgm` conține imaginea necesară task-ului respectiv, atât șirurile de codificat cât și cheia aferentă unde este cazul.

Se garanteaza faptul ca in testele noastre, mesajele incap in imaginile primite ca input.

Pentru claritate, structura fisierelor de input este urmatoarea:

```
P2
5 5
96
22 80 41 49 53
74 85 95 21 77
84 96 34 95 60
42 92 58 85 92
26 11 41 47 75
```

Prima linie contine magic number-ul pentru fisierele de tip plain pgm, "P2" scris în Format ASCII.

A doua linie contine dimensiunile imaginii - N numarul de coloane și M numarul de linii.

A treia linie contine valoarea maxima din matrice.

În continuare vom avea NxM valori separate cu spații și newline-uri care reprezinta imaginea noastra.

Trimitere și notare

Temele vor trebui încărcate pe platforma  vmchecker (în secțiunea IOCLA) și vor fi testate automat.

Arhiva încărcată va fi o arhivă .zip care trebuie să conțină:

- fișierul sursă ce conține implementarea temei: `tema2.asm`
- README ce conține descrierea implementării

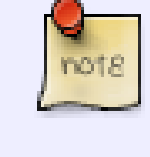
Punctajul final acordat pe o temă este compus din:

- punctajul obținut prin testarea automată de pe vmchecker - 90%
- README + coding style - 10%.

Se va ține cont de:

- claritatea codului
- denumire variabile, funcții și indentare consecventă
- comentarii
- nume sugestive pentru label-uri
- fișier README

Temele care nu trec de procesul de asamblare (build) nu vor fi luate în considerare.




Mașina virtuală folosită pentru testarea temelor de casă pe vmchecker este descrisă în secțiunea Mașini virtuale din pagina de resurse.



Vă reamintim să parcurgeți atât secțiunea de depunțiuni cât și regulamentul de realizare a temelor.

Resurse

Arhiva ce conține fișierele de la care puteți începe implementarea este  aici.



Data ultimei actualizări a checker-ului: 23.11.2019 17:53