U.S. ARMY COMBAT CAPABILITIES DEVELOPMENT COMMAND

DATA & ANALYSIS CENTER (DAC)

STEGANOGRAPHY

Professional Development Event (PDE)

Salvador Melendez, Ph.D.

Computer Engineer

DEVCOM DAC

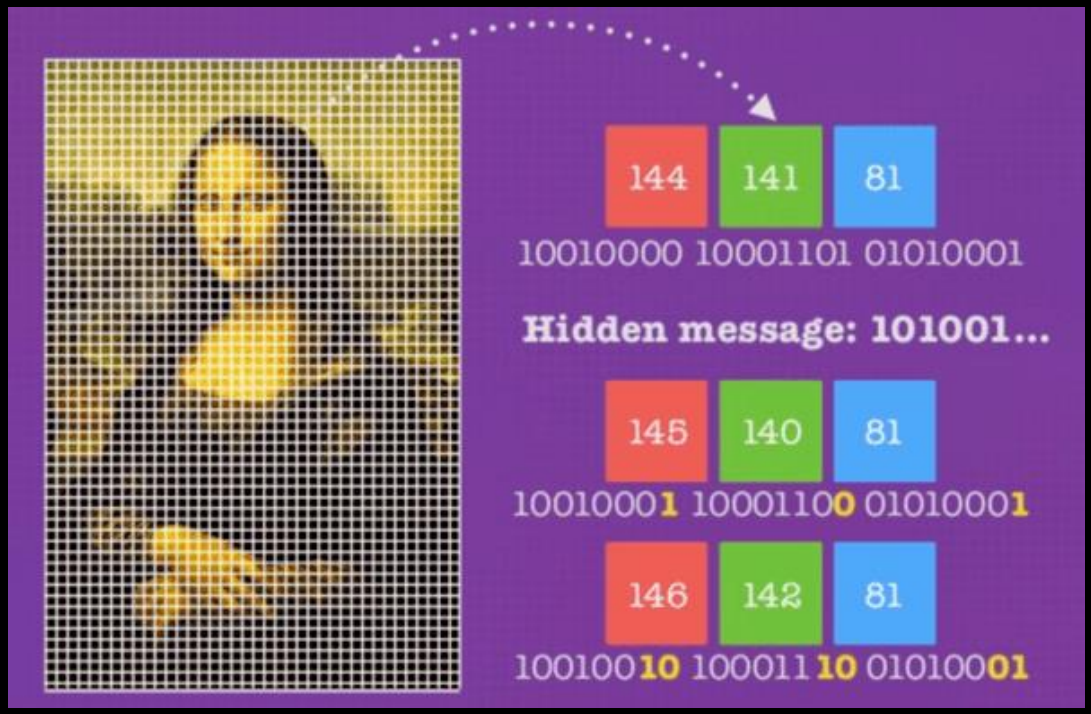
29 Jun 2020

BACKGROUND

- **What is Steganography?**
  - A technique used to hide data within an ordinary file (e.g. image, sound, text, etc.) to keep information secret from a naked eye.

- Steganography can be combined with Encryption to provide more security

- This PDE uses Steganography on Images.
  - Each pixel in the image has colors defined in RGB (Red, Green, Blue) format → Color intensity ranges from 0 to 255



APPROACH

- **Technique:**
  - Define an Alphabet to represent the characters in binary → 9 bits:

Character	Binary
Empty	000000000
Space	000000001
a	000000010
b	000000011
c	000000100
d	000000101
e	000000110
f	000000111
g	000001000
h	000001001
i	000001010
j	000001011
k	000001100
l	000001101
m	000001110
n	000001111
o	000010000
p	000010001
q	000010010
r	000010011
s	000010100
t	000010101
u	000010110
v	000010111
w	000011000
x	000011001
y	000011010
z	000011011

Character	Binary
A	000011100
B	000011101
C	000011110
D	000011111
E	000100000
F	000100001
G	000100010
H	000100011
I	000100100
J	000100101
K	000100110
L	000100111
M	000101000
N	000101001
O	000101010
P	000101011
Q	000101100
R	000101101
S	000101110
T	000101111
U	000110000
V	000110001
W	000110010
X	000110011
Y	000110100
Z	000110101
0	000110110
1	000110111

Character	Binary
2	000111000
3	000111001
4	000111010
5	000111011
6	000111100
7	000111101
8	000111110
9	000111111
~	001000000
`	001000001
!	001000010
@	001000011
#	001000100
$	001000101
%	001000110
^	001000111
&	001001000
*	001001001
(	001001010
)	001001011
_	001001100
-	001001101
=	001001110
+	001001111
[	001010000
]	001010001
{	001010010
}	001010011

Character	Binary
\	001010100
|	001010101
;	001010110
:	001010111
"	001011000
'	001011001
<	001011010
>	001011011
,	001011100
.	001011101
?	001011110
/	001011111

Why 9 bits?

APPROACH

- **Technique:**
  - We'll be using 9 bits to make it multiple to the number of colors (3) → RGB.
    - Each character will be hidden in 3 pixels.
    - Each pixel has a color in RGB.

Example:

message -->	h									e									l									l									o								
binary -->	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0
RGB -->	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B
pixel -->	#1			#2			#3			#1			#2			#3			#1			#2			#3			#1			#2			#3			#1			#2			#3		

APPROACH

- **Technique:**
  - We'll encode the bits by checking if they are Odd or Even:
    - Odd → 1 / Even → 0
  - We have 4 cases:
    - 1) The Pixel Value is Even and the Bit is '0' → Keep the Pixel Value Even
    - 2) The Pixel Value is Even and the Bit is '1' → Make the Pixel Value Even by adding 1
    - 3) The Pixel Value is Odd and the Bit is '0' → Make the Pixel Value Odd by substracting 1
    - 4) The Pixel Value is Odd and the Bit is '1' → Keep the Pixel Value Odd

		Pixel	
		Even	Odd
Bit	0	value	value-1
	1	value+1	value

APPROACH

Example:

message -->	h								
pixel -->	#1			#2			#3		
RGB -->	R	G	B	R	G	B	R	G	B
RGB Value -->	125	148	236	123	158	68	90	81	148
Odd/Even -->	Odd	Even	Even	Odd	Even	Even	Even	Odd	Even
binary -->	0	0	0	0	0	1	0	0	1
New Value -->	124	148	236	122	158	69	90	80	149

Character	Binary
Empty	000000000
Space	000000001
a	000000010
b	000000011
c	000000100
d	000000101
e	000000110
f	000000111
g	000001000
h	000001001
i	000001010

		Pixel	
		Even	Odd
Bit	0	value	value-1
	1	value+1	value

APPROACH

- **Technique:**
  - To decode the message, we reverse engineer it with the same method:

pixel -->	#1			#2			#3		
RGB -->	R	G	B	R	G	B	R	G	B
RGB Value -->	124	148	236	122	158	69	90	80	149
Odd/Even -->	Even	Even	Even	Even	Even	Odd	Even	Even	Odd
binary -->	0	0	0	0	0	1	0	0	1
message -->	h								

Character	Binary
Empty	000000000
Space	000000001
a	000000010
b	000000011
c	000000100
d	000000101
e	000000110
f	000000111
g	000001000
h	000001001
i	000001010
j	000001011
k	000001100
l	000001101

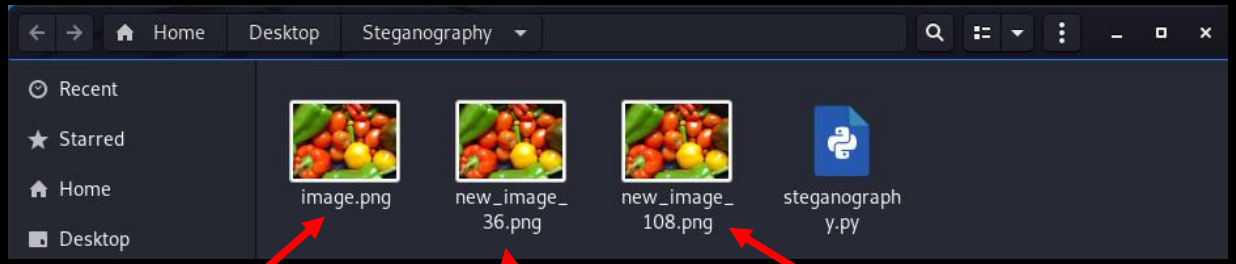
	Pixel	
	Even	Odd
Bit	0	1

APPROACH

- **Technique:**
  - How do we know WHERE is the END of the message?
    - We append the following sequence to the message → \n*

Example:

  - How do we keep images/messages in our working folder?



Original Image

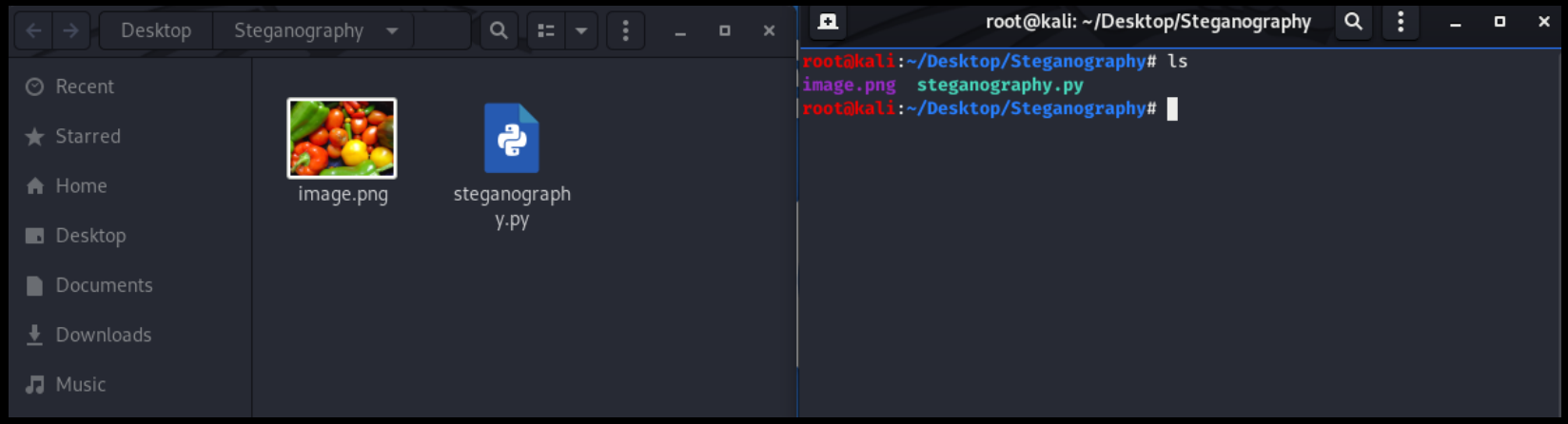
Encoded Message #36

Encoded Message #108

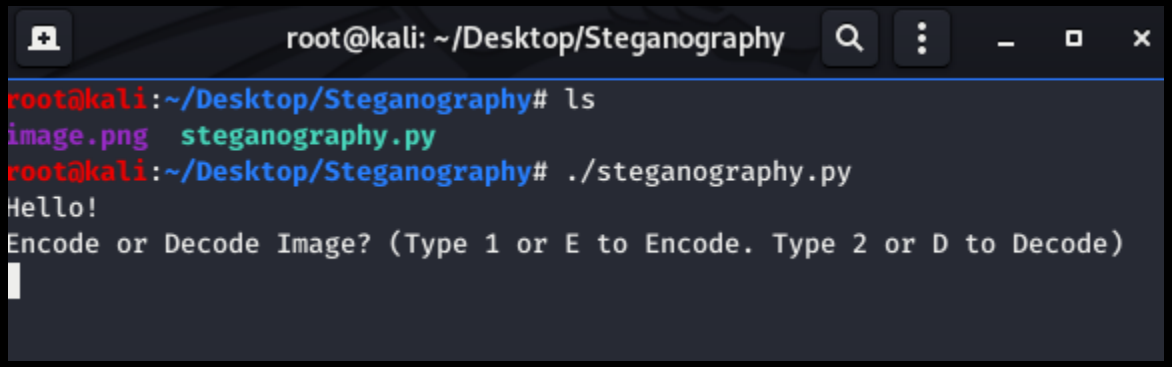
The number in the file name, tells you the COLUMN in which the message is hidden! The message is hidden VERTICALLY

IMPLEMENTATION

- **We'll implement Steganography on images with a Python script**
  - How do we run it?
    - Go to the Steganography Folder and make sure you have a '.png' image and the steganography.py script.

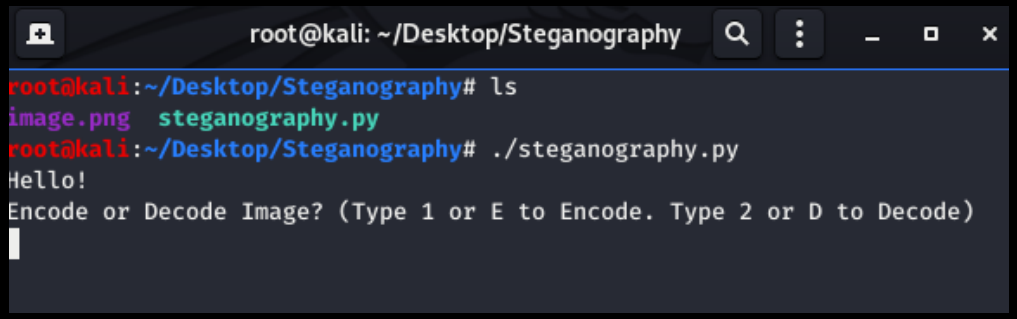


- Run the scrip by typing → python3 steganography.py

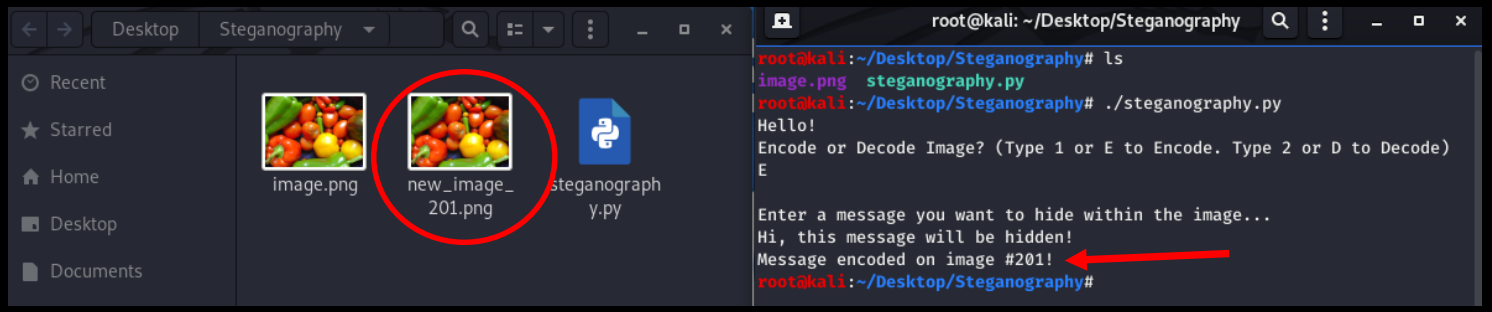


IMPLEMENTATION

- You will be prompted if you want to “Encode” or “Decode”.
  - Type “1” or “E” or “e” to Encode
  - Type “2” or “D” or “d” to Decode

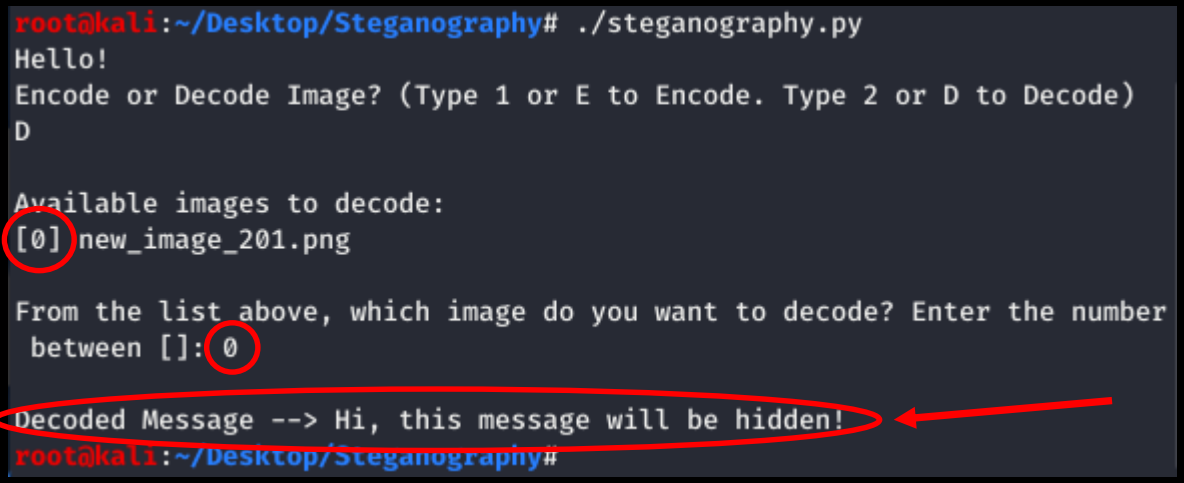


- When Encoding, just type any message you want to hide and press “enter” at the end.
  - A new image is created with a random number (remember it’s the column where the message is hidden)



IMPLEMENTATION

– When Decoding, just type the number between [ ] to select one of the available images



CONTACT INFORMATION

Salvador Melendez
UNCLASSIFIED email: salvador.melendez3.civ@army.mil