**זיהוי תוכנות זדוניות על פי הצגה בינארית**

מגיש: אסף הראל

ת״ז: 326191442

מורה: גד לידרור

בית הספר: תיכונט

Icon

Description automatically generated

# **מבוא**

## רקע לפרויקט

במשך השנים אנו הופכים ליותר ויותר תלויים בטכנולוגיה ודבר זה פותח חלון לאנשים בעלי כוונות זדוניות לנצל זאת וליצור תוכנות זדוניות (וירוסים) שיפגעו במחשבים שלנו ויגנבו מאיתנו מידע.

כיום יש הרבה תוכנות ״אנטי-וירוס״ שמטרתן לזהות תוכנות זדוניות שהוחדרו למחשב אך גם אותן האקרים מצליחים לעקוף בכך שהם יוצרים תוכנה זדונית שונה מאוד מהתוכנות הקיימות.

**מטרת הפרויקט** היא לענות על הצורך לתוכנת אנטי-וירוס שמסוגלת לזהות גם תוכנות זדוניות חדשות שעוד לא נראו בעולם.

הזיהוי נעשה בעזרת רשת נוירונים שלוקחת קבצי exe (executable), הופכת אותם לתמונה בעזרת ייצוג בינארי, מתמצתת מהם את התכונות החשובות ומזהה האם הם תוכנה זדונית או לא.

**המוצר העתידי** יהיה תוכנת אנטי-וירוס שתסרוק את כל קבצי ה-exe במחשב, תעבד את הקובץ ותכניס את ייצוגו הבינארי לתוך המכונה ותקפיץ התראה למשתמש אם קובץ נמצא כחשוד.

**קהל היעד** של הפרויקט הוא כל אחד, בין אדם פרטי לחברה גדולה.

בעולם של היום יש מעל 2 מיליארד מחשבים שנעים בין מחשב פרטי של קופאית בסופר לשרתים הענקיים של גוגל וכל אחד מהם חשוף לאימתני ההאקרים והתוכנות הזדוניות על בסיס יומי, למען האמת, 30% מהמחשבים בארצות הברית ו-57% מהמחשבים בדרום קוריאה ״נדבקו״ בתוכנות וירוסים (ואלו כמובן רק המחשבים שמצאו בהם).

אני בחרתי לחקור את הנושא הזה מכיוון שאני חושב שכל התחום של האנטי-וירוסים נשאר קבוע כבר הרבה מאוד שנים ואין זה יותר ממתבקש שככל שהטכנולוגיה מתקדמת וההאקרים מתקדמים גם ככה ההגנות עלינו המשתמשים יתקדמו.

## תהליך המחקר

**פונקציית hash (גיבוב)** – פונקציה שלוקחת פיסת מידע, ומוציאה פלט באורך קבוע הייחודי רק לה.

**טבלת hash** – מבנה נתונים מילוני (מבנה של ״מפתח-ערך״) שמשתמשים בפלט של פונקציית ה-hash בתור המפתח שלו ועל פי זה מקבלים מזהה ייחודי לכל פיסת מידע

**דוגמה ל-hash:**

נבצע פונקציית hash מסוג sha256 לכל הטקסט שכתוב בעמוד הקודם 🡨 “79ff06cba15ed7d0750b3d0258125cbd328bb03369a1cc4f43b009453554c6bf”

כמו שאפשר לראות, פונקציית ה-hash הפכה עמוד שלם למחרוזת של 64 תווים.

**תחילה חקרתי כיצד האנטי-וירוס המסורתי עובד:**

לתוכנה יש מאגר גדול מאוד של מזהים שהתקבלו על ידי הפעלת פונקציית ה-hash על תוכנות זדוניות קיימות שאותם מצליבים עם מזהה של קובץ על המחשב ובודקים אם הוא במאגר.

**החסרונות בדרך הפעולה הנוכחית**

אם תוכנת האנטי-וירוס נתקלת בתוכנה זדונית שונה ממה שקיים במאגר שלה אז היא לא תזהה

אותו, מכאן שאם האקר יהיה מתוחכם ויצירתי מספיק הוא יוכל לעקוף את ההגנות של המחשב.

**קבצי הרצה (exe) –** קובץ המורכב מפקודות מכונה (הוראות ישירות למעבד של המחשב) שתורגמו מקוד של תוכנית, כל התוכנות על המחשב הם מסוג exe.

**הצגה בינארית –** הצגה של קובץ בתור תמונה, הקוד שעושה זאת מחלק 5 טווחי מספרים בינאריים לצבעים שונים:

1. 0x00 – שחור
2. low – ירוק
3. ascii – כחול
4. high – אדום
5. 0xff – לבן

**דוגמה להצגה בינארית:**

A picture containing text

Description automatically generatedA computer screen capture

Description automatically generated with medium confidenceחלק מהערכים הבינאריים של קובץ וירוס

ההצגה הבינארית שלו

**כיצד זיהוי בעזרת הצגה בינארית פותר את החסרונות של אנטי-וירוס מסורתי?**

בעזרת הצגה בינארית אפשר להמיר את כל קבצי ההרצה (exe) לתמונות PNG וליצור מכונה שתלמד לזהות תוכנות זדוניות ועל פי מחקר שנערך באנגליה, מכונות מסוג זה מצליחות לזהות תוכנות זדוניות שעוד לא נראו במאגרים של האנטי-וירוסים הרגילים.

# **מדריך למפתח**

## הצגה בינארית

**מבנה הפרויקט:**

**color.py:**

*import* string

class ColorClass:

def \_\_init\_\_(self, data):

self.data = data

s = list(set(data))

s.sort()

self.symbol\_map = {v: i *for* (i, v) in enumerate(s)}

def point(self, x):

*return* self.getPoint(x)

def getPoint(self, x):

c = self.data[x]

*if* c == 0:

*return* [0, 0, 0]

*elif* c == 255:

*return* [255, 255, 255]

*elif* (0 < c < 32) and ((c != 9) or (c != 10) or (c != 13)):

*return* [104, 172, 87]

*elif* chr(c) in string.printable:

*return* [55, 126, 184]

*return* [228, 26, 28]

def \_\_len\_\_(self):

*return* len(self.data)

**hilbert.py:**

*from* . *import* utils

*import* numpy *as* np

def transform(entry, direction, width, x):

*assert* x < 2 \*\* width

*assert* entry < 2 \*\* width

*return* utils.rrot((x ^ entry), direction + 1, width)

def itransform(entry, direction, width, x):

*"""*

*Inverse transform - we simply reverse the operations in transform.*

*"""*

*assert* x < 2 \*\* width

*assert* entry < 2 \*\* width

*return* utils.lrot(x, direction + 1, width) ^ entry

*# There is an error in the Hamilton paper's formulation of the inverse*

*# transform in Lemma 2.12. The correct restatement as a transform is as follows:*

*# return transform(rrot(entry, direction+1, width), width-direction-2, width, x)*

def direction(x, n):

*assert* x < 2 \*\* n

*if* x == 0:

*return* 0

*elif* x % 2 == 0:

*return* utils.tsb(x - 1, n) % n

*else*:

*return* utils.tsb(x, n) % n

def entry(x):

*if* x == 0:

*return* 0

*else*:

*return* utils.graycode(2 \* ((x - 1) / 2))

def hilbert\_point(dimension, order, h):

*"""*

*Convert an index on the Hilbert curve of the specified dimension and*

*order to a set of point coordinates.*

*"""*

*# The bit widths in this function are:*

*# p[\*] - order*

*# h - order\*dimension*

*# l - dimension*

*# e - dimension*

hwidth = order \* dimension

e, d = 0, 0

p = [0] \* dimension

*for* i in range(order):

w = utils.bitrange(h, hwidth, i \* dimension, i \* dimension + dimension)

l = utils.graycode(w)

l = itransform(e, d, dimension, l)

*for* j in range(dimension):

b = utils.bitrange(l, dimension, j, j + 1)

p[j] = utils.setbit(p[j], order, i, b)

e = e ^ utils.lrot(entry(w), d + 1, dimension)

d = (d + direction(w, dimension) + 1) % dimension

*return* p

def hilbert\_index(dimension, order, p):

h, e, d = 0, 0, 0

*for* i in range(order):

l = 0

*for* x in range(dimension):

b = utils.bitrange(p[dimension - x - 1], order, i, i + 1)

l |= b << x

l = transform(e, d, dimension, l)

w = utils.igraycode(l)

e = e ^ utils.lrot(entry(w), d + 1, dimension)

d = (d + direction(w, dimension) + 1) % dimension

h = (h << dimension) | w

*return* h

class Hilbert:

def \_\_init\_\_(self, dimension, order):

self.dimension, self.order = dimension, order

@classmethod

def fromSize(cls, dimension, size):

*"""*

*Size is the total number of points in the curve.*

*"""*

x = np.math.log(size, 2)

*if* not float(x) / dimension == int(x) / dimension:

*raise* ValueError("Size does not fit Hilbert curve of dimension %s." % dimension)

*return* Hilbert(dimension, int(x / dimension))

def \_\_len\_\_(self):

*return* 2 \*\* (self.dimension \* self.order)

def \_\_getitem\_\_(self, idx):

*if* idx >= len(self):

*raise* IndexError

*return* self.point(idx)

def dimensions(self):

*"""*

*Size of this curve in each dimension.*

*"""*

*return* [int(np.ceil(len(self) \*\* (1 / float(self.dimension))))] \* self.dimension

def index(self, p):

*return* hilbert\_index(self.dimension, self.order, p)

def point(self, idx):

*return* hilbert\_point(self.dimension, self.order, idx)