Digital Whisper
Technologic Papers

מאבטחים את הבית בפחות מ-80 שורות קוד

מאת צח ירימי (I, Code)

הקדמה

אחת הדרכים היעילות ללמוד תכנות היא קריאת קוד של אחרים תוך ניסיון להבין למה הקוד כתוב כך ומה כל שורה עושה. הבעיה היא שלהיכנס לפרוייקט גדול יכולה להיות משימה קשה מדי, ומצד שני קשה מאוד למצוא פרוייקטים מעניינים המכילים שורות קוד מעטות בלבד.

מסיבה זאת החלטתי להכריז על סדרת המאמרים "50 שורות של קוד". במסגרת מאמרים אלו, אציג ואסביר לפרטים קטעי קוד בעלי 50 שורות (פחות או יותר), המהווים תוכנית שלמה. או במילים אחרות - קוד שעושה משהו מעניין והוא לא חלק ממשהו אחר. בהסברים אשתדל להתייחס לשתי השאלות: מה הקוד עושה, ולמה דווקא ככה? המאמר שאתם עומדים לקרוא לקוח מתוך סדרה זו, והוא מכיל קצת פחות מ-80 שורות קוד.

ולפני שנצלול, מילה על הבלוג שלי.
icode.co.il הוא בלוג תכנות בעברית, המיועד למפתחים מנוסים וחדשים כאחד. הבלוג מכסה מגוון נושאים, החל מסקירת טכנולוגיות ספציפיות ועד לטיפים כלליים על תכנות. אז אם אתם אוהבים קוד כמוני, אשמח אם תבואו לבקר בבלוג, תקראו את המאמרים וכמובן תחוו דעתכם בתגובות!

ואחרי כל ההקדמות - קדימה, לעבודה!

לפני כמה שנים כשעברתי לדירה משלי, החלטתי שאני צריך מצלמת אבטחה, אז קניתי אחת. אבל רוב מצלמות האבטחה לא שוות הרבה בלי תוכנה שמלווה אותן, ומסתבר שהתוכנות (החינמיות) בשוק אף פעם לא עושות בדיוק מה שמצפים מהן. אבל איזה מזל שאנחנו המתכנתים יכולים לעשות דברים בעצמנו! אז ניצלתי את ההזדמנות כדי לכתוב עוד פוסט לסדרה, הפעם של תוכנת אבטחה פשוטה המבצעת:

1. צפייה במצלמת הרשת וזיהוי תנועה.
2. בדיקה האם אני בבית (האם מכשיר הטלפון שלי מחובר לרשת הביתית).
3. אם אני לא בבית, שליחת התראה לנייד עם קובץ הוידאו המכיל את התנועה.

נשמע הרבה ל-80 שורות קוד? גם אני חשבתי כך, אבל איזה מזל שיש את python.
אז בואו נעשה את זה!

החומרים הדרושים

את הקוד נכתוב, כאמור, בשפת python (גרסה 2.7) ונשתמש בספריות הבאות:

1. OpenCV - ספריית ראייה ממוחשבת (ולכידת וידאו).
2. Scapy - ספריית רשת. נשתמש בה לבדיקה האם מכשיר מחובר לרשת הביתית.
3. Pushbullet - לשליחת התרעות באמצעות השירות Pushbullet (ראו בהמשך).
4. numpy - ספריית מתמטיקה.
5. הספריות המובנות multiprocessing ו-datetime.

את ההתראות למכשיר הטלפון אנו נשלח באמצעות השירות Pushbullet, שהוא שירות חינמי המאפשר להעביר התראות בין מכשירים שונים. יש להירשם לשירות, ולהתקין את האפליקציה על המכשיר הנייד אליו תרצו לקבל את ההתראה. כנוסף, כדי שנוכל לשלוח התראות באמצעות ה-API של השירות, יש צורך לייצר Access Token. ניתן לעשות זאת במסך Settings באתר.

משתמשי Windows - התקנת ספריית scapy עלולה להיות מעט מורכבת. אך אני הצלחתי להתקין באמצעות הקבצים בקישור הבא. שימו לב שיש להתקין גם WinPcap, שהוא דרייבר לכרטיסי רשת.

שלב ראשון - צפיה במצלמה

הספריה OpenCV מאפשרת חיבור למצלמת רשת ולכידת תמונות (פריימים) ממנה. כדי לעשות זאת יש לייצר אובייקט מסוג VideoCapture. על הדרך, נחלץ גם את רוחב וגובה התמונה המתקבלת מהמצלמה:

```
cap = cv2.VideoCapture(0)
frame_size = (int(cap.get(3)), int(cap.get(4)))
```

במידה ויש לכם יותר ממצלמה אחת מחוברת למחשב, ניתן לבחור לאיזו מצלמה להתחבר ע"י שינוי הפרמטר של VideoCapture מ-0 למספר אחר.

כעת נרוץ בלולאה ובכל איטרציה נבקש תמונה מהמצלמה ע"י הפונקציה read של VideoCapture:

```
while cap.isOpened():
    success, frame = cap.read()
    assert success, "failed reading frame"
    now = datetime.datetime.now()
```

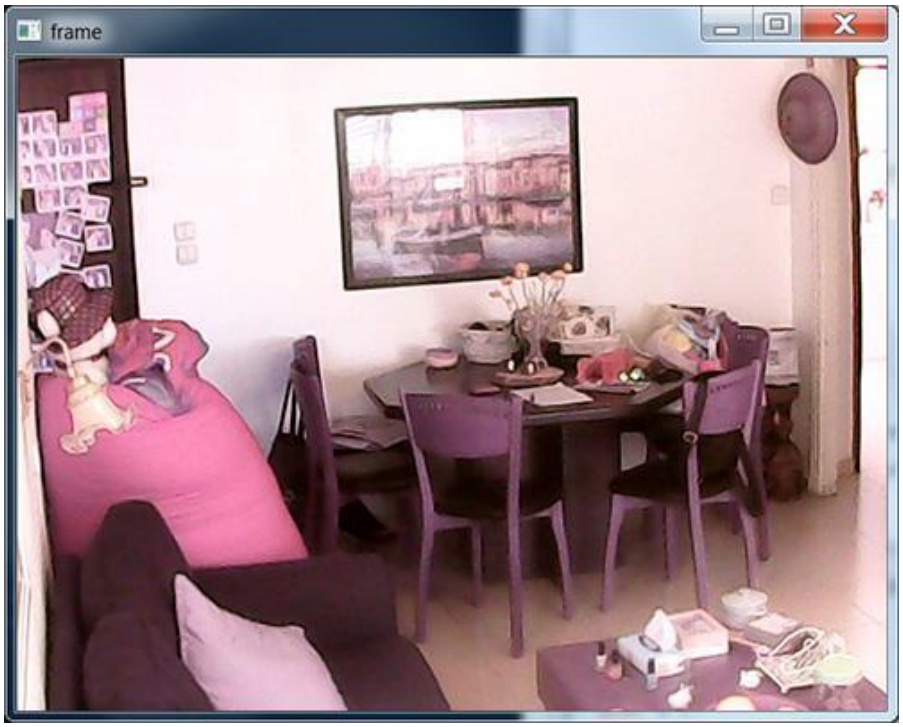
שימו לב שפונקציית read מחזירה שני ערכים: הצלחה/כשלון ואת התמונה עצמה. כמו כן שימו לב לשורת ה-assert, שנועדה לוודא שהצלחנו לקבל תמונה. אם לא הצלחנו, יזרק exception והתוכנית תצא. השורה האחרונה שומרת את הזמן הנוכחי במשתנה now, לו נזדקק בהמשך.

בהמשך (עדיין בתוך הלולאה) נציג למסך את התמונה שלכדנו, ולאחר מכן נשתמש בפונקציה waitKey כדי לבדוק האם נלחץ המקש q, ובמקרה זה לצאת מהלולאה (ומהתוכנית):

```
while cap.isOpened():
    ...

    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

נסו להריץ את הקוד למעלה. אם המצלמה שלכם מחוברת כהלכה אתם אמורים לראות חלון דומה לזה:



[לכידת תמונה ממצלמת הרשת]

שלב שני - זיהוי תנועה

עכשיו כשיש לנו זרם של פריימים מהמצלמה, נרצה לזהות תזוזה ולהתריע בהתאם. קיימות דרכים רבות לזהות תזוזה, שרובן ככולן מבוססות על השוואה של התמונה הנוכחית לתמונה / תמונות הקודמות. ההשוואה שנעשה בתוכנית שלנו מורכבת ממספר שלבים, שכולם יבוצעו באמצעות הספרייה OpenCV:

1. טשטוש התמונה והמרתה לגווני אפור, זאת כדי להפחית שינויים מזעריים בין התמונות הנגרמים ע"י המצלמה.
2. ביצוע diff (חיסור) בין הפריים הנוכחי לפריים הקודם ויצירת תמונה המורכבת מההבדלים ביניהם. הבדלים בין התמונות יהיו קיימים רק אם התרחשה תנועה כלשהי. אם לא היתה תנועה, הפריימים יהיו (כמעט) זהים.

3. סינון כל ההבדלים מתחת לסף מסויים, כדי להסיר "רעשים" מזעריים הנגרמים ע"י המצלמה עצמה גם אם אין תנועה.
4. אם התמונה הסופית מכילה פיקסלים שאינם שחורים, נניח שהייתה תנועה ונפעל בהתאם.

נתחיל בכתיבת פונקציה המקבלת שתי תמונות (שכבר טושטשו והומרו לגווני אפור) ומבצעת את שלבים 2-4, כלומר מחזירה ערך בוליאני המציין האם הייתה תזוזה. לצורך כך נשתמש שוב בספריית OpenCV:

```
def have_motion(frame1, frame2):
    delta = cv2.absdiff(frame1, frame2)
    thresh = cv2.threshold(delta, 25, 255, cv2.THRESH_BINARY)[1]
    return numpy.sum(thresh) > 0
```

השורה הראשונה בפונקציה מחשבת את ההבדלים בין התמונות ומייצרת את התמונה delta, המייצגת את ההבדלים. השורה השנייה מאפסת את כל הפיקסלים שערכם קטן מ-25, ואת כל השאר קובעת למקסימום (255). ניתן לשנות פרמטר זה כדי לקבוע את הרגישות לתנועה. וכך זה נראה:

שלבי זיהוי תנועה. משמאל, הפריים המקורי, עם היד שלי בזמן תנועה. באמצע, ההפרש בין הפריים המקורי לזה שקדם לו. בימין - ההפרש לאחר הרצת הפונקנציה threshold. כעת כל שנותר הוא להשתמש בפונקצייה have_motion בלולאה הראשית:

```
frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
frame_gray = cv2.GaussianBlur(frame_gray, (21, 21), 0)

if have_motion(prev_frame, frame_gray):
    last_motion = now
    # TODO: start recording to a video file

prev_frame = frame_gray
```

שתי השורות הראשונות ממירות את הפריים לגווני אפור ומטשטשות אותו. השורה האחרונה שומרת את הפריים הנוכחי למשתנה prev_frame, שבאיטרציה הבאה של הלולאה יהפוך לפריים הקודם (לאחר לכידת תמונה חדשה מהמצלמה). שימו לב שבמידה והיתה תנועה, אנו שומרים את הזמן הנוכחי למשתנה last_motion, לו נזדקק בהמשך.

שלב שלישי - שמירה לקובץ וידאו

כזכור, בעת זיהוי תנועה נרצה לשלוח וידאו של התנועה למכשיר הנייד שלנו. לשם כך נצטרך להקליט את התנועה ולשמור אותה לקובץ וידאו. גם כאן ספריית OpenCV מספקת את הכלים. נתחיל ביצירת אובייקט VideoWriter השומר לקובץ וידאו:

```
motion_filename = now.strftime("%Y_%m_%d_%H_%M_%S_MOTION.avi")
motion_file = cv2.VideoWriter(motion_filename, fourcc, 20.0, frame_size)
```

אתחול האובייקט מתבצע עם הפרמרטים הבאים:

1. שם קובץ הפלט, אותו אנו בונים בשורות הראשונה והשנייה לפי הזמן הנוכחי (מהמשתנה now).
2. אובייקט מסוג CV_FOURCC, המציין את **הקידוד** בו נרצה לשמור את הקובץ. חשוב לבחור את הקידוד כך שיתמך גם ע"י מערכת ההפעלה שלנו, וגם ע"י המכשיר הנייד אליו שולחים את הקובץ. עבורי הקידוד **XVID** עובד מצויין, אך יתכן שתזדקקו לקצת ניסוי וטעייה עם רשימת הקידודים הנמצאת באתר fourcc.org.

```
fourcc = cv2.cv.CV_FOURCC(*"XVID")
```

3. מספר פריימים לשניה.
4. tuple המכיל את רוחב וגובה הפריים, אותו יצרנו מוקדם יותר.

לפני שנמשיך ונכתוב פריימים לקובץ, חשוב להבין את הלוגיקה שאנו עומדים לבצע. כאשר אנו מזהים תנועה, נרצה לשמור לקובץ לא רק את הפריים הנוכחי, אלא מספר שניות נוספות של וידאו. כלומר נרצה שכל עוד יש תנועה נמשיך להקליט לקובץ, ובמידה והתנועה נעצרה - להמשיך להקליט מספר שניות נוספות, ואז לסגור את הקובץ ולשלוח אותו.

במילים אחרות, בתוך הלולאה שלנו נצטרך לבדוק: "האם אנחנו כרגע במצב הקלטה? אם כן - נשמור את הפריים הנוכחי לקובץ". נעשה זאת כך:

```
if motion_file is not None:
    motion_file.write(frame)
```

וכדי לסגור את הקובץ ולשלוח אותו, נחכה שהזמן שעבר מאז התנועה האחרונה יעלה על מספר שניות:

```
if motion_file is not None:
    motion_file.write(frame)
    if now - last_motion > MOTION_RECORD_TIME:
        motion_file.release()
        motion_file = None
        # TODO: send video file
```

כאשר את הפרמטר MOTION_RECORD_TIME נגדיר בתחילת הקובץ, למשל ל-10 שניות:

```
MOTION_RECORD_TIME = datetime.timedelta(seconds = 10)
```

כעת למעשה סיימנו לכתוב תוכנית המזהה ומקליטה תנועה לקובץ וידאו!

שלב רביעי - בדיקה האם אני בבית

גם כאן ישנן דרכים רבות למימוש. לדוגמה: אם למחשב שלנו יש חיבור Bluetooth, ניתן לבדוק אם המכשיר הנייד שלנו בסביבה ע"י סריקת מכשירים קרובים. אני בחרתי בדרך פשוטה של בדיקה האם המכשיר הנייד שלי מחובר לרשת ה-WiFi הביתית.

לפני שנתחיל, יש למצוא את כתובת ה-MAC של המכשיר הנייד שלנו. למי שלא מכיר, כתובת ה-MAC, או כתובת פיזית, היא מספר ייחודי הניתן לכל ציוד רשת בעולם. מספר זה לרוב מוטבע בחומרה עצמה ולא משתנה, ולכן ניתן להשתמש בו כדי לזהות מכשירים ספציפיים, כל עוד הם מחוברים לאותה רשת כמו המחשב שלנו. ניתן למצוא הוראות למציאת כתובת ה-MAC של סוגי מכשירים שונים ע"י חיפוש בגוגל:

```
find mac address <device name>
```

בנוסף נצטרך למצוא את כתובת ה-IP שלנו וה-subnet mask של הרשת הביתית שלנו, ע"י הרצת הפקודה ipconfig ב-command line של Windows (הוראות עבור מערכות הפעלה אחרות). את שני הנתונים האלו נמיר לטווח כתובות ה-IP של הרשת הביתית (בפורמט CIDR) ע"י שימוש בכלי הבא. זה אולי נשמע קצת מסובך, אבל לרוב כל שנצטרך לעשות הוא להחליף את המספר האחרון בכתובת ה-IP שלנו ב-0 ולהוסיף את המחרוזת "/24".

נשמור את שני פריטי המידע הנ"ל במשתנים:

```
DEVICE_MAC = "3d:f9:c2:d8:0f:d5"
SUBNET = "192.168.1.0/24"
```

כעת נשתמש בספרייה scapy כדי לבצע סריקת ARP. שאילתת ARP היא הדרך המקובלת להמרת כתובת IP ברשת המקומית לכתובת MAC. אפשר לדמיין שאילתת ARP כצעקה "מיהו בעל כתובת ה-IP הזאת?", אליה עונה בעל הכתובת בלבד: "אני הבעלים של כתובת ה-IP, וה-MAC שלי הוא...".

סריקת ARP שולחת שאילתות ARP עבור **כל אחת מכתובות ה-IP האפשריות ברשת** ואוספת את התשובות שהתקבלו. אם המכשיר שלנו מחובר לרשת (יש לו כתובת IP ברשת), הוא אמור לענות לפקודת ה-ARP עם כתובת ה-MAC שלו, וכך נדע שהוא מחובר. נכתוב קוד שמבצע את הסריקה הנ"ל, אוסף את התשובות ומחפש בהן את כתובת ה-MAC של המכשיר שלנו:

```
def is_device_connected(mac_addr):
    answer, _ = scapy.srp(
        scapy.Ether(dst="ff:ff:ff:ff:ff:ff") /
        scapy.ARP(pdst=SUBNET), timeout=2)
    return mac_addr in (rcv.src for _, rcv in answer)
```

לאחר הרצת הפונקציה scapy.srp, המשתנה answer יכיל את מערך התשובות שהתקבלו לשאילתות ה-ARP. השורה השנייה בודקת האם כתובת ה-MAC שסופקה נמצאת בתוך אחת התשובות.

שלב חמישי ואחרון - שליחת קובץ הוידאו

כעת נוודא שאנחנו לא בבית, ונשלח לעצמנו התראה עם קובץ הוידאו באמצעות השירות Pushbullet:

```
def push_file(filename):
    if is_device_connected(DEVICE_MAC):
        print "Device is connected, not sending"
        return
    print "Sending", filename
    pushbullet = Pushbullet("PUSHBULLET_API_KEY")
    my_device = pushbullet.get_device("My Device")
    file_data = pushbullet.upload_file(open(filename, "rb"), filename)
    pushbullet.push_file(device = my_device, **file_data)
```

שימו לב להחליף את המחרוזות PUSHBULLET_API_KEY ו-My Device בערכים המתאימים מתוך חשבון ה-Pushbullet שלכם.

כעת כל שנותר הוא לקרוא לפונקציה push_file כאשר סיימנו להקליט את קובץ הוידאו. **אבל...** חשוב לשים לב שהרצת הפונקציה לוקחת מספר שניות, מכיוון שסריקת ה-ARP והעלאת הקובץ הן פעולות ארוכות יחסית. אם נקרא לפונקציה push_file ישירות מהלולאה הראשית, התוכנית שלנו "תיתקע" עד ששליחת הקובץ תסתיים, ולכן נאבד מספר שניות של האזנה למצלמה.

כדי שדבר כזה לא יקרה (חור אבטחה!) נרצה להריץ את תהליך השליחה **במקביל** ללולאה הראשית. ניתן לעשות זאת ע"י שימוש ב-threads, אך ב-python מומלץ להשתמש דווקא בתהליך נפרד. נעשה זאת ע"י שימוש במחלקה Process מתוך הספריה המובנית multiprocessing:

```
if now - last_motion > MOTION_RECORD_TIME:
    ...
    Process(target = push_file, args = (motion_filename, )).start()
```

וכך נראית ההודעה שמתקבלת במכשיר:

ההודעה שהתקבלה במכשיר הנייד

מה הלאה?

כמובן שהתוכניות שהצגתי כאן היא בסיסית מאוד, וניתן לשפר חלקים רבים בה. הנה כמה רעיונות:

1. שמירת כל הוידאו לדיסק, גם אם אין תנועה. אפשר באיכות נמוכה יותר כדי לחסוך מקום בדיסק.
2. הגבלת גודל קובץ הוידאו הנשלח, למשל לדקה אחת.
3. הזרמת הוידאו בלייב למכשיר הנייד.
4. תמיכה במספר מצלמות / מכשירים ניידים במקביל.
5. זיהוי המכשיר הנייד בדרכים אחרות.
6. זיהוי פרצופים והתראה על פרצופים חשודים בלבד.

הקוד המלא

ניתן למצוא את הקוד המלא ב-GitHub של I, Code. אשמח אם תמשיכו לפתח את הפרוייקט, למצוא לי באגים, ולשלוח pull requests!