עמוד פתיחה

# תוכן עניינים

[תוכן עניינים 1](#_Toc55946402)

[תיאור הבעיה 2](#_Toc55946403)

[הפתרון 3](#_Toc55946404)

[מרכיבי התוכנה 4](#_Toc55946405)

[אפליקציית Updater 4](#_Toc55946406)

[אפליקציית Launcher 4](#_Toc55946407)

[אפליקציית Installer 5](#_Toc55946408)

[מימוש הפתרון 6](#_Toc55946409)

[אפליקציית Updater 6](#_Toc55946410)

[מודול rsa\_signing 6](#_Toc55946411)

[מודול settings 6](#_Toc55946412)

[מודול registry 6](#_Toc55946413)

[מודול messages 7](#_Toc55946414)

[מודול updater 8](#_Toc55946415)

[מודול service 9](#_Toc55946416)

[הירככיית קבצים 10](#_Toc55946417)

[אפליקציית Launcher 11](#_Toc55946418)

[אפליקציית Installer 12](#_Toc55946419)

[הוראות התקנה ושימוש 14](#_Toc55946420)

[הוראות שימוש למפתח 14](#_Toc55946421)

[דרישות להכנת קובץ התקנה 14](#_Toc55946422)

[הכנת קובץ התקנה ויצירת שרת עדכון 14](#_Toc55946423)

[הכנת עדכון גרסא 14](#_Toc55946424)

[הפצת עדכון גרסא 14](#_Toc55946425)

[בניית שרת עדכון נוסף 14](#_Toc55946426)

[הוראות שימוש למשתמש 14](#_Toc55946427)

[התקנה 14](#_Toc55946428)

[שימוש 14](#_Toc55946429)

[אבחון בעיות 14](#_Toc55946430)

[בדיקות וטסטים 15](#_Toc55946431)

[מצגת 16](#_Toc55946432)

# תיאור הבעיה

כשמתכנת יחיד או חברה קטנה מפתחתים תוכנה, הם רוצים להתמקד בפיתוח השירותים והפיצ'רים שהתוכנה מספקת. אבל, הם הרבה פעמים נתקלים בהמון דברים נלווים שצריך לפתח לתוכנה. מנגנונים כמו: ניהול משתמשים, אימות ורכישת רישיון, הפצת עדכוני תוכנה, שליחת מידע בעת קריסה ועוד...

כל המנגנונים האלה יכולים להיות הכרחיים לחלק מהתוכנות, והן יכולות לעכב משמעותית מפתח או חברה בעלת כמות אנשים מוגבלת שאין לה את משאבי העבודה ליצור אותם. הבעיות הללו יכולות להיות מאוד מורכבות לפעמים, בשביל להבטיח שבעתיד, כשהיקף השימוש בתוכנה יגדל, אותן מערכות ימשיכו לעבוד ללא שינויים משמעותיים וללא הפרעות ללקוחות.

הפרויקט שלי יתמקד באחד מהמנגנונים הללו, ואציג כמה דברים מרכזיים שחשוב שיהיה למנגנון זה בשביל להבטיח שימוש נרחב היקף שיוכל להתאים להרבה מאוד תוכנות (ואולי אפילו לכולן עם התאמות קטנות).

המנגנון שאותו אספק בפרויקט הוא מנגנון ל**הפצת עדכוני תוכנה** בפשטות, יעילות ובטיחות, תוך כדי שמירה על אפשרות להרחבה (Scalability). בנוסף, הוא יספק גם תמיכה בעדכוני תוכנה עבור רשתות סגורות (כדוגמאת ארגונים וכו') בעלי DMZ אשר יכול לגשת לרשת. יש לציין כי ברוב מערכות ההפעלה המבוססת Unix יש Repositories שונים שמאגדים בתוכם עדכונים לתכונות ותכנים אחרים, ולכן הפרויקט יתמקד במערכת ההפעלה Windows, ובעיקר בגרסתה החדשה והאחרונה (כנראה), Windows 10.

דבר אחד שחשוב שיהיה באפליקציית הפרויקט הוא אינטגרציה נוחה של בין אפליקציית הפרויקט לתוכנת המפתח, שמעוניין להפיץ עדכונים עבורה. כלומר, שלמפתח יהיה קל ליצור עדכון תוכנה ולהפיץ אותו לשאר הלקוחות.

הדבר השני הוא יעילות הפצת העדכון. כלומר, בעת הפצת עדכון הוא יגיע במהירות לשאר הלקוחות, בעומס מינימלי על שרת העדכונים ובצורה אמינה. כמובן שישנן מגבלות שלא תלויות תוכנה, כמו מהירות העלאה של שרת העדכונים או המרחק הפיזי בין השרת ללקוח, אך המימוש ינסה למצוא פתרון לצמצום המגבלות הללו.

דבר נוסף הוא אבטחת עדכוני התוכנה. כידוע, קוד זדוני אשר מורץ ע"י המשתמש הוא מקור לפרצת אבטחה חמורה. על כן, ישנו חשש שתוקף זדוני יזייף עדכון תוכנה ובכך יגרום למשתמש להריץ תוכנה שלדעתו היא מוכרת ובטוחה, אך בפועל יורץ קוד שתוקף זדוני שלח. על כן, כל עדכון תוכנה וכל מנגון עדכון הגרסאות צריך לספק אבטחה מפני איום כזה או אחר.

לבסוף, אולי בהתחלה לא יהיה צורך בכך, אך כנראה שלאחר שימוש זמן באפליקציית הפרויקט עם תוכנה שמפתח בנה, הוא יצטרך להרחיב את היקף השירות שלו. אם מדובר בגדילה של כמות המשתמשים, או התפשטות של החברה למקומות שונים בעולם. הפתרון צריך לספק אופציה פשוטה, שתהיה שקופה למשתמשי הקצה (המשתמשים הפשוטים לפחות, אם מדובר בחברה אז עובדי IT אולי יצטרכו לעשות התאמות במקרים בהם הרשת היא סגורה) על מנת לתמוך בהקמת עוד שרתי הפצה וניהול הפצת עדכוני הגרסאות כך שכולם מסונכרנים בצורה תקינה.

# הפתרון

תוכנת ה-Updater הינה מעטפת לתוכנה קיימת אשר מאפשרת למפתח התוכנה ליצור ולהפיץ עדכונים לתוכנה שלו עבור מערכת ההפעלה Windows. ה-Updater בנוי בצורה גנרית כך שכל מפתח יוכל להתאים אותו עבור התוכנה אותה הוא פיתח, ובכך להיות פטור מהצורך של בניית מנגנון עדכונים מהיר ומאובטח, שאותו מספקת לו תוכנת ה-Updater.

כאשר המפתח סיים לפתח את התוכנה שלו והוא מעוניין להפיץ אותה למשתמשים חדשים, הוא משתמש ב-Updater על מנת ליצור קובץ התקנה לתוכנה. ביצירת קובץ ההתקנה, המפתח יבחר כתובת לשרת עדכונים, ועוד קונפיגורציות אחרות, שיאפשרו ללקוחות שמתקינים את התוכנה לדגום את שרת העדכונים הרשמי ולהוריד ממנו עדכוני תוכנה.

בנוסף לכך, כאשר לקוח מסוים מוריד עדכון תוכנה, הוא מעדכן על כך את שאר המחשבים ברשת שלו. כל מחשב באותה הרשת שמקבל את ההודעה על עדכון התוכנה יוכל להוריד אותה מהלקוח שכבר הוריד את העדכון, ובהמשך לבחור האם להתקין את העדכון או להשאר בגרסא הישנה. דבר זה יוצר יתרון משמעותי מאוד בעת הפצת עדכון, ומאפשר שיפור ביצועים בהפצת עדכון התוכנה והורדה משמעותית מהעומס על שרת העדכון הראשי.

בנוסף, הפתרון הזה מאפשר גם Scalability. במידה והמפתח ירצה ליצור מספר שרתי עדכון שונים במקומות שונים בעולם, כל שעליו לעשות הוא להתקין את התוכנה על השרתים ולהגדיר את אותם שרתים באותו ה-WAN, כך ששרת העדכון הראשי יוכל לעדכן אותם באמצעות broadcast על עדכון תוכנה. בעת עדכון תוכנה, המפתח יוכל בלחיצת כפתור להפיץ את העדכון ובנוסף ליידע את כל שאר שרתי העדכון להוריד את העדכון אליהם, והם כבר יפיצו אותו באותה הדרך שכל לקוח אחר יפיץ את העדכון ברשת שלו.

כיוון ששיטת ההפצה אינה מסתמכת על "מקור מוסמך", אלא שכל לקוח יכול לשלוח קובץ עדכון לתוכנה, נדרש בפתרון גם מנגנון אבטחה המונע מתוקף זדוני לשלוח קובץ עדכון מזויף ובכך ליצור backdoor למחשבו של לקוח אחר. לכן, כל הודעה, שמקורה משרת העדכונים הרשמי, נחתמת בחתימה דיגיטלית בעזרת מפתח הצפנה אסימטרי המבוסס על שיטת ההצפנה RSA. בנוסף, כל קובץ עדכון נחתם גם הוא בחתימה דיגיטלית באותה הצורה. כאשר מפתח התוכנה יוצר את קובץ ההתקנה בפעם הראשונה, מפתחות ההצפנה מוגרלים בשרת. המפתח הפרטי נשמר לוקאלית על שרת העדכון, והמפתח הציבורי מצורף לקובץ ההתקנה כך שכל לקוח יוכל לאמת באצמעותו הודעות המתקבלות משרת העדכון כבר מהרגע הראשון להתקנה (מפתח ההצפנה לא נשלח ברשת ולכן לא יכול להישלח ע"י תוקף זדוני שהגריל מפתח הצפנה משלו).

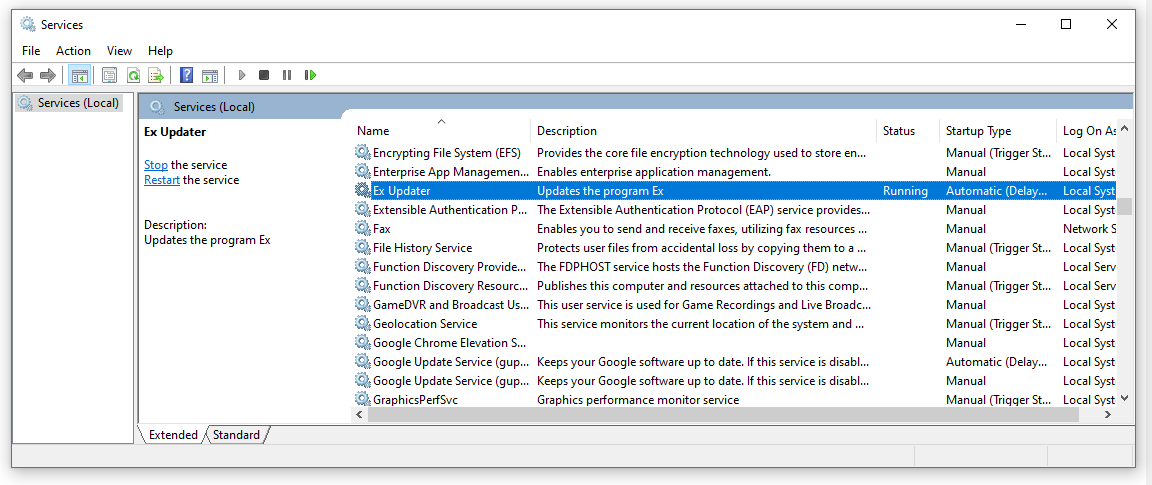
יתר על כן, על מנת שמנגנון ההפצה יהיה מהיר ויעיל, התוכנה לא מחכה שכל לקוח יריץ אותה בנפרד ורק בהרצה שלה העדכון יורד אליו ויופץ ממנו לשאר הלקוחות. במקום זאת, בעת התקנת התוכנה נוצר Service אשר רץ על המחשב תמיד. ה-Service הזה מאזין בקביעות להודעות ברשת ומוריד גרסאות חדשות כשהוא מקבל הודעה על הוצאת גרסא חדשה. לאחר שהגרסא מורדת, הוא מפיץ בתורו הודעה על גרסא חדשה וכן הלאה...

כאשר הלקוח מריץ את התוכנה לאחר שיצאה גרסא חדשה, תופיע לו הודעה שגרסא חדשה הורדה אל המחשב שלו, והיא תשאל אותו האם הוא מעוניין להתקין אותה. במידה ויבחר לעשות זאת, הגרסא החדשה תותקן והגרסא הקודמת תמחק. אחרת, שתי הגרסאות ישארו עד שהמשתמש יבחר להתקין את הגרסא החדשה. בעת הרצת התוכנה, נבדקות כל הגרסאות שהורדו אל המחשב ונמחקות גרסאות שלא נמצאות בשימוש.

# מרכיבי התוכנה

## אפליקציית Updater

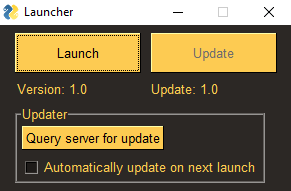
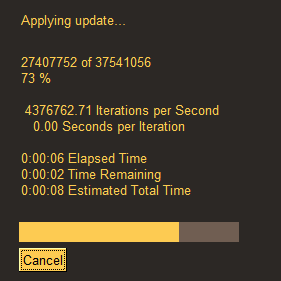
האפליקצייה המרכזית של התוכנה. היא אחראית על קבלת עדכוני גרסא, הורדה שלהם, והפצה של העדכונים לשאר המחשבים ברשת. האפליקצייה היא ליתר דיוק Windows Service שמוגדר לפעול בעליית המחשב והוא תמיד רץ על מנת שיוכל לקבל את עדכוני הגרסא ברגע שהם מופצים, מבלי לפספס אותם, בדומה לאיך ש-Google מעדכנת את התוכנות שלה.



## אפליקציית Launcher

אפליקציית ה-Launcher הינה תוכנה נפרדת בלתי תלויה ב-Updater. היא מותקנת ביחד עם התוכנה ויוצרת קובץ הרצה שאותו המשתמש מריץ כשהוא מעוניין להריץ את התוכנה. כשהמשתמש מריץ את ה-Launcher, נפתח מולו חלון (התמונה משמאל) עם פרטים לגבי הגרסא הקיימת, הגרסא המעודכנת ביותר, ואפשרות לבחירה האם הוא מעוניין שהתוכנה תמיד תתעדכן לפני ריצה. בנוסף, הוא יכול לתשאל את שרת העדכונים שלו אם קיימת גרסא חדשה. במידה וכן, ה-Updater כבר ידאג להוריד אותה ולהפיץ אותה, ובהמשך המשתמש יוכל להתקין אותה בפעם הבאה שהתוכנה מורצת.

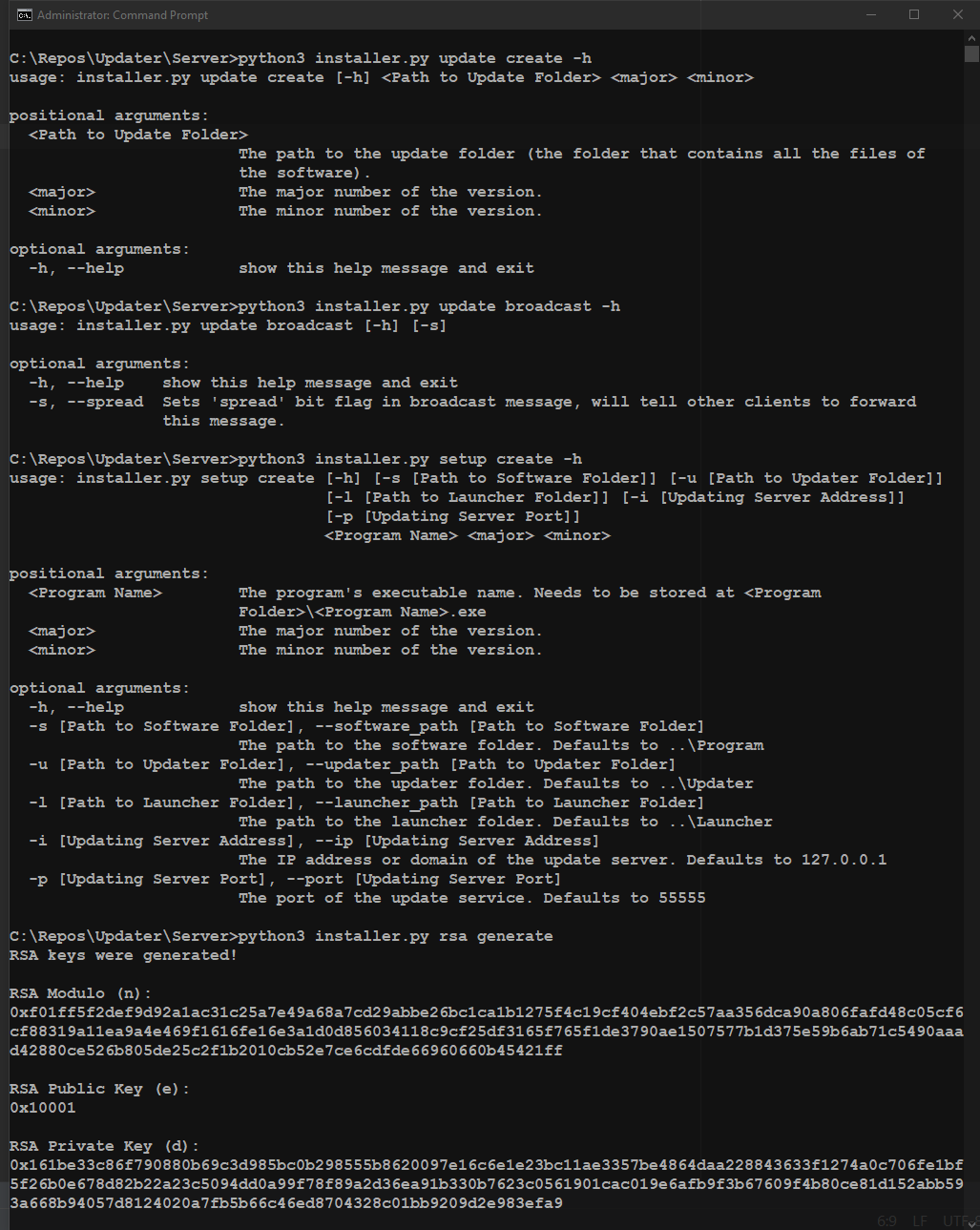
כאשר המשתמש בוחר לעדכן גרסא (ע"י לחיצה על Update, לדוגמא), ולהתקין את הגרסא המעודכנת, יופיע לו חלון עם progress bar בדומה לחלון מימין:

## אפליקציית Installer

התוכנה הזו היא למעשה מיועדת לשימוש המפתח. בעזרתה הוא יכול ליצור קובץ התקנה עבור התוכנה אותה הוא פיתח, לבחור מפתחות הצפנה עבור תקשורת עם השרת, ליצור עדכוני תוכנה ולהפיץ אותם ללקוחות ברשת. התוכנה מיועדת לרוץ על שרת העדכונים ובעת ריצתה היא מגדירה בו קונפיגורציות מתאימות ע"פ פעולותיו של המפתח. משום כך, התוכנה הזו אינה הופכת לקובץ Executable רגיל של Windows, אלא נותרת קובץ Python שהמפתח יריץ בעזרת ה-Interpreter.

השימוש בתוכנה מתבצע באמצעות ה-CMD והרצת פקודות מסוימות הנתמכות ע"י ה-Installer:



# מימוש הפתרון

## אפליקציית Updater

### מודול rsa\_signing

המודול האחראי על אבטחת המידע שמועבר בין השרת ללקוח ובין הלקוחות. הוא מאפשר לחתום חתימות דיגיטליות באמצעות RSA ולאמת אותן. המודול מייצא את הפונקציות הבאות:

* הפונקציה sign – מאפשרת לחתום על מידע באמצעות מפתח הצפנה פרטי.
* הפונקציה sign\_hash – מאפשרת לחתום על אובייקט hash באמצעות מפתח הצפנה פרטי.
* הפונקציה validate – מאפשרת לוודא חתימה על מידע באמצעות מפתח ציבורי.
* הפונקציה validate\_hash – מאפשרת לוודא חתימה על אובייקט hash באמצעות מפתח ציבורי.

### מודול settings

המודול אחראי על טעינת הגדרות התוכנה בעת תחילת ריצתה ושמירה של הגדרות לאחר שינויים שלהם. בפעם הראשונה חלק מההגדרות יאותחלו לפי פרמטרים דיפולטים, אך לאחר מכן כל ההגדרות ישמרו בקובץ בשם settings.json שישמור את הנתונים בפורמט JSON. חלק מהאפליקציות האחרות גם נעזרות בקובץ הזה בשביל הגדרות שהן צריכות להשתמש בהן. הוא מייצא את הפונקציות הבאות:

* הפונקציה init\_settings - מאתחלת את ההגדרות בתחילת הריצה.
* הפונקציה load\_settings – טוענת את ההגדרות מקובץ ההגדרות settings.json.
* הפונקציה save\_settings – שומרת את ההגדרות בקובץ ההגדרות settings.json.

### מודול registry

המודול אחראי על ביצוע פעולות מול ממשק ה-registry של Windows. בעזרתו מודולים אחראים יכולים לשמור ולאחזר נתונים מה-registry. רוב הנתונים הנשמרים ב-registry תמיד נטענים משם בעת השימוש בהם, כיוון ששם בוודאות ישמרו הנתונים המעודכנים ביותר (במידה ואפליקציה אחרת שינתה את הגדרה מסוימת, ההגדרות באפליקציה אחרת לא יעודכנו אוטומטית, אבל ה-registry כן). הפונקציות שהמודול מייצא הן:

* הפונקציה create\_key – יוצרת registry key חדש.
* הפונקציה key\_exists – בודקת אם registry key בנתיב נתון קיים.
* הפונקציה delete\_key\_recursive – מוחקת registry key ואת כל תכולתו.
* הפונקציה exists – בודקת אם value קיים בנתיב נתון.
* הפונקציה delete – מוחקת value בנתיב הנתון.
* הפונקציה get\_all\_sub\_values – מחזירה רשימה של כל ה-values תחת registry key נתון.
* הפונקציה get\_all\_sub\_keys – מחזירה רשימה של כל ה-keys תחת registry key נתון.
* הפונקציה get\_value – מאחזרת ערך של value מסוים, וממירה אותו למשתנה פייתוני.
* הפונקציה set\_value – שומרת משתנה פייתוני ב-registry תחת registry key נתון.

### מודול messages

המודול מגדיר את סוגי ההודעות השונות הנשלחות בין השרתים והלקוחות. ראשית, הוא מגדיר Enum בשם MessageType המכיל את סוגי ההודעות (type), כך שכל הודעה מקבלת מספר ייחודי. בנוסף, עבור כל סוג הודעה מוגדר construct משלה, אובייקט שמאפשר להמיר את ההודעה למשתני השפה וחזרה (serialize). כמובן שלכל סוג הודעה מבנה שונה, אך ישנה הודעה כללית בשם GENERIC\_MESSAGE שמגדירה פורמט זהה לכל ההודעה, שבאמצעותו ניתן להבין באיזו הודעה מדובר. יתר על כן, לכל ההודעות יש תמיד גודל קבוע, המוגדר ב- settings.MESSAGE\_SIZE.

סוגי ההודעות השונים המוגדרים בקובץ הם:

* VERSION\_UPDATE\_MESSAGE – הודעה המודיעה ליעדה כי מופצת גרסא. במידה והגרסא המופיעה בהודעה אינה חדשה יותר מהגרסאת העדכון הקיימת, ה-Updater יתעלם ממנה. מבנה ההודעה:
  + type – סוג ההודעה, ע"י ה-Enum המוגדר ב-MessageType.
  + header\_signature – חתימה דיגיטלית של ההודעה.
  + major – גרסאת ה-major של הגרסא המופצת.
  + minor – גרסאת ה-minor של הגרסא המופצת.
  + size – גודל הגרסא המופצת בבתים.
  + update\_signature – חתימה דיגיטלית של הגרסא המופצת.
  + spread – flag אשר קובע האם מקבל ההודעה צריך להפיץ את אותה ההודעה לשאר הרשתות אליהם הוא מחובר.
* SERVER\_UPDATE\_MESSAGE – הודעה המודיעה ליעדה כי יש לעדכן את כתובת שרת העדכונים או את ה-port עליו יש להאזין לעדכונים. ההודעה הזאת היא מוצא אחרון במקרים בהם אחד מהפרטים הללו כבר לא ניתן לשימוש (לדוגמא, אם כתובת ה-DNS נלקחה). יש לציין שהשימוש בהודעה הוא מסוכן כיוון שאם אחד מהיעדים לא קיבל את העדכון שצריך להחליף פורט, הוא כנראה לעולם לא יקבל אותו כיוון שהוא יאזין על פורט ישן, לכן יש להשתמש בה בזהירות ורק כמוצא אחרון. מבנה ההודעה הוא:
  + type – סוג ההודעה, ע"י ה-Enum המוגדר ב-MessageType.
  + signature – חתימה דיגיטלית של ההודעה.
  + address\_id – מספר ייחודי של ההודעה. הוא קיים בשביל שאדם זדוני לא יוכל לשלוח הודעת עדכון ישנה ובכך לגרום ללקוחות "לעדכן" לפרטי שרת עדכון ישן. כל עוד ה-address\_id אינו יותר גדול מה-address\_id האחרון שהתקבל, ההודעה לא תתקבל ע"י ה-Updater.
  + address – כתובת DNS או כתובת IP של שרת העדכונים שאליו צריך לפנות לצורך עדכוני גרסאות.
  + port – הפורט שעליו יש להאזין ושבו מופצות כלל ההודעות.
  + spread – flag אשר קובע האם מקבל ההודעה צריך להפיץ את אותה ההודעה לשאר הרשתות אליהם הוא מחובר.
* REQUEST\_VERSION\_MESSAGE – הודעה אשר נשלחת ע"י לקוח ובה הוא מבקש משרת או לקוח אחר שישלח לו גרסא מסוימת. היא נשלחת לאחר קבלת הודעה מסוג VERSION\_UPDATE\_MESSAGE. לפני שליחת ההודעה הזו, הלקוח מתחיל האזנה ב-TCP על פורט אקראי, ושולח את מספר הפורט הנ"ל בהודעה. היעד של ההודעה מצופה לשלוח את קובץ הגרסא אל מקור ההודעה בפורט שעליו הוא מאזין. כיוון שהגרסא נשלחת ב-TCP, ישנה אמינות טובה לכך שהקובץ יגיע בשלמותו, או לכל הפחות שנוכל לזהות שגיאות של העברה כשהן קורות. מבנה ההודעה הוא:
  + type – סוג ההודעה, ע"י ה-Enum המוגדר ב-MessageType.
  + crc32 – מקור ההודעה הוא אינו משרת עדכון רשמי, לכן הלקוח אינו יכול לחתום עליה, לכן במקום signature מופיע כאן crc שמאפשר לבדוק שלא קרתה טעות בהעברת ההודעה.
  + listening\_port – פורט ה-TCP שעליו מאזין הלקוח לקבלת קובץ הגרסא.
  + major - גרסאת ה-major של הגרסא שהלקוח מבקש.
  + minor - גרסאת ה-minor של הגרסא שהלקוח מבקש.
* REQUEST\_UPDATE\_MESSAGE – הודעה אשר נשלחת ע"י לקוח ובה הוא מבקש משרת שישלח לו את מספר הגרסא החדש ביותר. בעצם, בעזרת ההודעה הזו הלקוח דוגם את השרת ובודק אם יצאה גרסא חדשה יותר. ההודעה הזו אינה נשלחת ע"י ה-Updater, אלא ע"י ה-Launcher. ה-Updater של השרת הוא זה שמקבל את ההודעה ומשיב עליה בהודעה מסוג VERSION\_UPDATE\_MESSAGE הנשלחת ל-Updater של הלקוח.

### מודול updater

המודול המרכזי של ה-Updater. במודול הזה נמצאת כל הלוגיקה של התוכנה האחראית על תקשורת ועדכון והפצה של גרסאות.

ראשית, הוא מכיל מחלקה די פשוטה בשם Version המייצגת גרסא, כלומר, מספר major ומספר minor, שביחד מייצרים מספר גרסא. ישנן מספר פונקציות סטטיות שמאפשרות לקבל את מספר הגרסא המותקנת או מספר הגרסא המעודכנת מה-registry ולעדכן אותם גם כן.

שנית, הוא מכיל מחלקה מורכבת שמייצגת את אפליקציית ה-Updater. המחלקה הזאת היא גדולה ואחראית על קבלת הודעות, פרסור שלהן, אימות שלהן (חתימה או CRC), עיבוד שלהן ושליחת הודעת תגובה בהתאם. בתחילת הריצה של אפליקציית Updater, נוצר מודל חדש של המחלקה ורצה הפונקציה שלו בשם setup\_listener, הפונקציה יוצרת שרת UDP שמאזין על פורט הניהול של התוכנה, הפורט שאליו שרת העדכון שולח הודעות על גרסאות חדשות והודעות נוספות. לאחר מכן, ניתן לקרוא לפונקציה receive\_message שאחראית על קריאת הודעה משרת ה-UDP. אם ההודעה התקבלה כראוי, היא תשמר כ-member של אובייקט המחלקה ביחד עם השולח. לאחר מכן, ניתן לקרוא לפונקציה handle\_message שמוואדת את גודל ההודעה (הרי שלכל ההודעות גודל זהה), מוודאת שה-type שלה הוא תקין ואז, בהתאם לסוג ההודעה, מאמתת את החתימה/CRC שלה. לאחר שכל הבדיקות הגנריות עברו בהצלחה, הפונקציה קוראת לפונקציית הטיפול המתאימה בהתאם לסוג ההודעה:

* עבור הודעות מסוג REQUEST\_UPDATE – רק שרתים רשמיים אמורים לענות על הודעות אלו. ולכן אם מקבל ההודעה הוא לקוח, הוא פשוט יתעלם מההודעה. אחרת, מקבל ההודעה הוא שרת עדכון ונקראת הפונקציה handle\_request\_update. הפונקציה הזאת מאחזרת מה-registry את הגרסא העדכנית ביותר שקיימת ושולחת עבורה הודעה מסוג VERSION\_UPDATE אל שולח ההודעה המקורית.
* עבור הודעות מסוג SERVER\_UPDATE – נקראת הפונקציה handle\_server\_update, שמפרסרת את ההודעה ובודקת אם ה-address\_id יותר מעודכן ממה ששמור כרגע ב-registry (אם הוא לא, תתעלם מההודעה). לאחר מכן, אם השרת ביקש להפיץ את ההודעה בעזרת flag ה-spread, תקרא הפונקציה broadcast\_message שמפיצה את ההודעה האחרונה שהתקבלה לשאר הרשתות בפורט הניהול של התוכנה. לבסוף, הפונקציה מעדכנת ב-registry את כתובת השרת ואת פורט הניהול ומאתחלת את השרת בעזרת cleanup\_listener ו-setup\_listener.
* עבור הודעות מסוג VERSION\_UPDATE – נקראת הפונקציה handle\_version\_update, שמפרסרת את ההודעה ובודקת אם הגרסא המופצת היא עדכנית (כלומר, שהגרסא הקיימת כרגע היא יותר ישנה). לאחר בדיקה זו, במידה והגרסא אכן עדכנית, תקרא הפונקציה download\_update (הסבר עליה למטה). לבסוף, אם השרת ביקש להפיץ את ההודעה בעזרת flag ה-spread, היא תופץ בעזרת הפונקציה broadcast\_message.
* עבור הודעות מסוג REQUEST\_VERSION – הטיפול בהודעה זו הוא מעט שונה. זוהי ההודעה היחידה אשר מטופלת בצורה מקבילית, בעזרת thread נוסף שנוצר ברגע שהיא מתקבלת. ה-thread החדש מקבלת כפרמטרים את ההודעה ואת שולח וההודעה ומריץ את הפונקציה send\_version\_update. הפונקציה הזו מפרסרת את ההודעה שהתקבלה, בודקת אם קיימת הגרסא המבוקשת בהודעה, ובמידה וכן, מתחילה לשלוח אל פורט ה-TCP שעליו מאזין השולח את קובץ העדכון ב-chunk-ים. משום שכמה לקוחות צפויים לבקש את עדכון הגרסא, יש צורך לטפל בהם במקביל בעזרת thread לכל אחד. בכל שאר הפונקציות לא נרצה לעשות זאת משום שבעת טיפול בהודעה המידע הקיים עלול להשתנות וזה עלול ליצורrace condition במקרים מסוימים.

הפונקציה download\_update היא הפונקציה שנקראת לאחר קבלת VERSION\_UPDATE. כאשר היא נקראת, היא פותחת שרת TCP בפורט אקראי ומתחילה להאזין עליו לחיבור. לאחר מכן, היא שולחת הודעה מסוג REQUEST\_VERSION לשולח ההודעה המקורית שהודיע על קיום גרסא חדשה. לאחר שהצד השני יקבל את הודעת ה- REQUEST\_VERSIONהוא יתחיל לשלוח לנו את קובץ העדכון. אנו מקבלים את המידע שנשלח ושומרים אותו בקובץ עד אשר כל הקובץ מתקבל. לאחר מכן, אנו מאמתים את מקור הקובץ ע"י בדיקה של החתימה הדיגיטלית שלו, ובמידה והוא מאומת, אנו שומרים את המידע על הגרסא החדשה ב-registry ומסיימים את פעולת הפונקציה. כעת הגרסא החדשה הורדה ומחכה להתקנה. מי שבפועל יגרום להתקנת הגרסא החדשה יהיה ה-Launcher, ע"פ בקשה של המשתמש.

### מודול service

המודול הראשי של אפליקציית ה-Updater. זה המודול שרץ בתור Windows Service. כאשר התוכנה מותקנת, מריצים את ה-EXE של הסקריפט הזה עם פרמטרים מסוימים שמתקינים את ה-service ומאפשרים לו לרוץ בעליית המחשב. במודול הזה מוגדרת מחלקה שמגדירה את ה-service של ה-Updater, מחלקה אשר יורשת ממחלקה ServiceFramework של הספריה pywin32. כאשר ה-service מורץ, נוצר אובייקט של המחלקה ומורצת הפונקציה SvcDoRun שלו. היא מאתחלת את ה-service ומריצה את פונקציית ה-start שלו. לאחר כמה אתחולים נוספים, נוצר אובייקט של המחלקה Updater ולאחר מכן נכנס ה-service ל-cycle loop שלו שבו הוא מטפל ב-Event-ים.

הטיפול ב-Event-ים מתבצע בעזרת פונקציות ה-WinAPI המתאימות. ראשית, יוצרים Event עבור ה-socket של ה-updater ומקשרים בין ה-socket ל-event באמצעות WSAEventSelect. לאחר מכן, קוראים לפונקציה WaitForMultipleObjects שבעזרתה ממתינים, ללא הפסקה, על ה-event של ה-socket ועל ה-event של ה-service. אם ה-event של ה-service מתרחש, סימן שגורם חיצוני (המשתמש או משהו אחר במערכת) ביקש מה-service לסיים את ריצתו. במצב כזה ה-service מדווח על סיום ריצתו ויוצא מה-cycle loop. אם ה-event של ה-socket מתרחש, אז משתמשים באובייקט ה-updater שיצרנו בשביל לקרוא את ההודעה מה-socket בעזרת הפונקציה receive\_message ולאחר שזו התקבלה בהצלחה מריצים את הפונקציה handle\_message שדואגת לטיפול בהודעה בהתאם.

### הירככיית קבצים

כאשר המשתמש מתקין את התוכנה, הוא בוחר מיקום שבו היא תשמר על המחשב שלו, תיקיית ההתקנה. בזמן ההתקנה נוצרת במיקום הזה תיקייה בשם Updater שבה נשמרים כל הקבצים הקשורים ל-Updater כמו: קובץ הריצה שלו service.exe, קובץ ההגדרות settings.json, וכל קבצי הגרסאות הרלוונטיים, הנשמרים בפורמט zip.

בנוסף, נשמר קובץ בשם log.txt (באותה התיקייה עם service.exe) שמכיל את כל הודעות ה-logging של ה-service. הוא מאוד שימושי לצורך מעקב אחר הפעולות שה-service מבצע (בעצם כמעט הפלט היחיד שיש ל-service).

בתיקיית ההתקנה, ביחד עם התיקייה Updater, קיימת גם תיקייה בשם התוכנית המותקנת. שם נשמרים כל קבצי התוכנית המקורית של המפתח, וכך מופרדת ההיררכיה של התוכנה המקורית וה-updater (שלא תהיה התנגשות בשמות קבצים). בתיקיית ההתקנה גם נשמר ה-launcher, שעליו יורחב בהמשך.

כאשר לקוח מוריד עדכון גרסא חדש, הוא נשמר בתיקייה של ה-Updater, בתור קובץ zip המכיל בשמו את מספר הגרסא, ובנוסף נשמר ערך ב-registry עבור הגרסא הזו. ה-launcher אחראי על התקנה של עדכוני הגרסא המורדים ע"י ה-updater ומחיקת עדכונים ישנים.

## אפליקציית Launcher

אפליקציית ה-Launcher היא לא יותר מתוכנית פייתון פשוטה ש"קומפלה" ע"י PyInstaller ל-exe. כאשר היא מורצת ללא פרמטרים, היא מציגה למשתמש חלון GUI שמציג מידע אודות הגרסא הנוכחית, עם אפשרות לעדכון גרסא קיימת, הרצת התוכנה, תשאול שרת העדכונים וכו'. יש לה גם כמה אופציות מעניינות לשימוש ב-CMD, שמאפשרים לבצע כל פעולה שהמשתמש יכול לבצע ב-GUI, דרך ה-CMD.

לאחר פרסור הפרמטרים שהתקבלו בעזרת argparse, מורצת הפונקציה setup עם הפרמטרים. בפונקציה נבדק כל אחד מהפרמטרים והפונקציה מטפלת בכל המקרים של הפרמטרים השונים:

* אם התקבל הפרמטר versions (-v), הפונקציה תדפיס את הגרסא המותקנת והגרסא העדכנית ביותר שקיימת ותצא מבלי להריץ דבר נוסף.
* אם התקבל הפרמטר update\_file (-u), הפונקציה תבדוק אם אכן מדובר בקובץ, ותנסה לעדכן את הגרסא בעזרת הקובץ הזה.
  + הפונקציה update מבצעת את הפעולה הזו.
* אם התקבל הפרמטר check\_update (-c), הפונקציה תתשאל את שרת העדכונים השמור ב-registry ותשלח לו הודעה מסוג REQUEST\_UPDATE. השרת בתקווה יקבל את ההודעה ויחזיר אליה בתגובה הודעה מסוג VERSION\_UPDATE, שאותה יקבל ה-service של ה-updater שידאג להוריד את הגרסא במידה והיא חדשה יותר מהגרסא העדכנית ביותר.
  + הפונקציה query\_server מבצעת את הפעולה הזו.
* אם התקבל הפרמטר apply\_update (-a), הפונקציה תבדוק אם קיימת גרסא עדכנית יותר מהגרסא המותקנת ובמידה וכן, תתקין אותה.
  + הפונקציה install\_update מבצעת את הפעולה הזו.
* אם לא התקבל אף פרמטר, כפי שקורה כאשר מריצים את התוכנה דרך ה-explorer ולא דרך ה-CMD, יפתח חלון ה-GUI לעבודה מול ממשק גרפי.

בכל אחד מהמקרים האלה, אם לא מסופק הפרמטר silent (-s), עלול להפתח חלון GUI שמדווח על פעולות התוכנה (לדוגמא, progress bar במקרה של התקנת גרסא). בנוסף, ניתן לקרוא לפרמטר no\_launch (-n) בשביל למנוע את ריצת התוכנית המקורית לאחר ריצת ה-launcher.

מעבר לפעולות המתוארות מעלה, רוב הלוגיקה של ה-Launcher היא עבודה מול ממשק גרפי באמצעות הספריה PySimpleGUI.

דבר נוסף ואחרון שה-launcher עושה לפני שהוא מריץ את התוכנה המקורית, הוא להריץ את הפונקציה cleanup\_old\_updates. מכיוון שה-service של ה-updater מוריד גרסאות עדכון חדשות, וה-launcher מתקין אותן, אז הוא זה שאחראי על המחיקה שלהן. בנוסף, ה-updater גם הוא משתמש בקבצים הללו ושולח אותם לשאר הלקוחות בתהליך של הפצת גרסא, לכן בעת ניסיון המחיקה ה-launcher עלול להתקל בשגיאה כיוון שהקבצים נמצאים בשימוש. לכן, הפונקציה cleanup\_old\_updates אחראית לבדוק את כל הגרסאות הקיימות ב-registry ובעזרתו למחוק את כל הגרסאות שכבר לא בשימוש. אם היא נכשלת (כפי שעלול לקרות), היא מדלגת על כך בתקווה שבפעם הבאה הקובץ יהיה פנוי למחיקה, התכנון של ה-updater וה-launcher מוודא שקובץ עדכון מסויים לא יהיה נעול להרבה זמן, כך שהדבר הוא סביר לחלוטין.

## אפליקציית Installer

ה-Installer הוא הכלי היחיד שאותו מריץ המפתח. גם הוא, בדומה ל-Launcher, תוכנת פייתון אשר מורצת בעזרת ה-CMD, אבל כיוון שהמפתח מריץ אותו, אין צורך להמירו ל-exe שכן המפתח יכול להריץ את התוכנה בעזרת Python המותקן על השרת. ה-Installer מורכב מכמה סוגים שונים של פקודות, שכל אחת אחראית על חלק שונה, ולכל אחת יש תת-פקודות בחלק זה. הפקודות הראשיות של ה-Installer הן: rsa, update, server, setup. כאשר כל אחת מהן מבצעת את הדברים הבאים:

* rsa – אחראי על הצגת מפתחות ה-RSA השמורים בשרת וגינרוץ מפתחות חדשים
  + תת-הפקודה rsa show מציגה את המפתחות באמצעות הפונקציה show\_rsa\_keys.
  + תת-פקודה rsa generate מייצרת מפתחות חדשים באמצעות generate\_rsa\_keys.
* update – אחראית על ייצור קבצי עדכון (גרסאות חדשות) והפצה שלהן.
  + תת-הפקודה update create מייצרת קובץ עדכון באמצעות create\_update.
  + תת-הפקודה update broadcast מפיצה את הגרסא העדכנית ביותר (שיוצרה באמצעות update create) באמצעות הפונקציה broadcast\_update\_version. היא בעצם שולחת ברשת הודעה מסוג VERSION\_UPDATE שאותה מקבלים ה-service-ים.
* server – אחראית על הצגת פרטי שרת העדכון והפצת שינויים בפרטי שרת העדכון.
  + תת-הפקודה server show מציגה את פרטי השרת הרשומים ב-registry באמצעות הפונקציה show\_server\_information.
  + תת-הפקודה server update שולחת ב-broadcast עדכון על פרטי שרת העדכון לשאר הרשת (כולל לשרת עליו מורצת הפקודה) באמצעות הפונקציה send\_server\_information. כפי שנאמר קודם, ייתכן שלא כל הלקוחות/שרתים יקבלו את עדכון הפרטים ולכן חלקם עלולים לאבד תקשורת עם השרת/שאר הלקוחות, ולכן לא מומלץ להשתמש בפקודה זו, אלא רק כמוצא אחרון.
* setup – אחראית על יצירת קובץ התקנה עבור תוכנה חדשה.
  + תת-הפקודה setup clean דואגת לנקות שאריות מהרצות קודמות של פקודת ה-setup. לדוגמא, דברים כמו מפתחות RSA, תוצרים של PyInstaller (של ה-Launcher וה-service), תוצרים של InnoSetup וכו'...
  + תת-הפקודה setup create היא כנראה הפונקציה הכי גדולה וחשובה ב-Installer. היא מקבלת נתיב לקבצי התוכנה של המפתח, נתיב לקוד של ה-Launcher, נתיב לקוד של ה-Updater, פרטי גרסא ופרטים של שרת העדכון. הפעולות שהיא מבצעת, בפונקציה create\_installer הן אלו:

1. יוצרת קובץ הגדרות settings.json שמכיל את הפרטים לגבי התוכנה: מספר גרסא, כתובת שרת עדכון, פורט ניהול הגרסאות ושם התוכנה.
2. מגנרצת מפתחות rsa חדשים (כפי שנעשה בפקודה rsa generate).
   * יש לשים לב שהמפתחות הישנים נדרסים!
3. מריצה את PyInstaller על ה-updater, בכדי ליצור את הקובץ service.exe.
4. מריצה את PyInstaller על ה-Launcher, בכדי ליצור את launcher.exe.
5. מעתיקה את קבצי תוכנת המפתח, על מנת לבנות מהם קובץ התקנה.
6. יוצרת סקריפט בשם setup\_script.iss, שהוא סקריפט בפורמט המותאם לתוכנה InnoSetup שבעזרתו היא יודעת כיצד ליצור את קובץ ההתקנה.
   * הסקריפט נוצר באמצעות פרטים מסוימים על התוכנה וסקריפט קבוע שמשמש כ-template לסקריפט שמיוצר ע"י התוכנה.
7. לבסוף, מריצים את InnoSetup עם הסקריפט setup\_script.iss שמייצר עבורינו קובץ התקנה בפורמט הבא: X\_n.m\_setup.exe כאשר X מציין את שם התכונה, n מציין את מספר ה-major ו-m מציין את מספר ה-minor.
   * לאחר הרצת הפקודה, על המפתח להריץ את קובץ ההתקנה ולהתקין באמצעותו את התוכנה. לאחר מכן, השרת עליו רצה הפקודה והותקנה התוכנה מתפקד באופן רשמי כשרת עדכון שבאמצעותו ניתן ליצור עדכונים ולהפיץ אותם (ועוד...).

כמובן שרוב הפקודות הנ"ל מקבלות עוד פרמטרים בדרכים כאלו ואחרות. הסבר מפורט עליהם נמצא בפרק "הוראות התקנה ושימוש", אך אני ממליץ להשתמש ב-flag של -h (--help) על כל פקודה או תת-פקודה כדי להבין כיצד להשתמש בה ומה מייצג כל דבר.

# הוראות התקנה ושימוש

ראשית, יש לציין כי אם אתה רק מעוניין לבדוק את הפונקציונאלית של הפרויקט, ולא להשתמש בו עבור תוכנה משלך, כדאי שתקרא את החלק [הזה](#_בדיקות_וטסטים) שבו יש רשימה של מכונות וירטואליות המקונפגות בצורה רשתית ומאפשרות בדיקה של התוכנה. זה מקל מאוד על בניית ה-setup הרשתי והקמת המכונות.

## הוראות שימוש למפתח

### דרישות להכנת קובץ התקנה

הדרישות להרצת ה-Installer על השרת והכנת קובץ התקנה מופיעות כאן. ניתן להסתכל על חלק ה-Setup בקובץ ה-README.md של ה-Installer (בתיקיית Server) בשביל הוראות כיצד להתקין את הדרישות.

* מכונה בעלת מערכת הפעלה Windows 10 x64 בעל חיבור רשתי.
  + נבדק על: Windows 10 Pro 64 bit - version 2004, OS Build 19041.508.
  + כדאי לוודא שמערכת ההפעלה מעודכנת או לכל הפחות בגרסא שעליה נבדקו התוכנות.
* פייתון 3 (נבדק על 3.8.2 – 64 ביט)
  + Pip מעודכן עבור גרסאת הפייתון המצויינת.
* C++ Build Tools for Visual Studio (ראה הוראות Setup ב-README.md לתכנים ספציפיים).
  + נבדק עם גרסת 2019.
* התקנת כל הספריות הבאות בפייתון (בעזרת ה-pip, ראה הוראות Setup ב-README.md):
  + pycryptodome – נבדק על גרסא 3.9.8
  + construct – נבדק על גרסא 2.10.56
  + validators – נבדק על גרסא 0.18.1
  + netifaces – נבדק על גרסא 0.10.9
  + colorama – נבדק על גרסא 0.4.3
  + pywin32 – נבדק על גרסא 228
  + pyinstaller – נבדק על גרסא 4.0
  + pysimplegui – נבדק על גרסא 4.29.0
* העתקת הקובץ pywintypesXX.dll (XX היא גרסת הפייתון שלך) מהתיקייה:

<Python Install Folder>\Lib\site-packages\pywin32\_system32

אל התיקייה:

<Python Install Folder>\Lib\site-packages\win32

* התקנת התוכנה InnoSetup – נבדק על גרסא 6.0.5
  + ה-Installer אינו יודע למצוא לבד היכן מותקנת התוכנה, לכן יש לעדכן בקובץ installer.py, בשורה 31, את המשתנה INNO\_SETUP\_INSTALL\_FOLDER לתיקיית ההתקנה שלה.

### הכנת קובץ התקנה ויצירת שרת עדכון

לאחר ווידוא כל הדרישות מעלה, ניתן ליצור קובץ התקנה לתוכנה שלך. נניח כעת כי אתה מעוניין ליצור קובץ התקנה לתוכנה בשם Foobar. וכי כל הקבצים הרלוונטיים לתוכנה נמצאים בתיקייה C:\Foobar. בנוסף, נניח כי הקובץ הראשי שאיתו פותחים את התוכנה נקרא foobar.exe והוא נמצא בנתיב C:\ Foobar\ foobar.exe. בשביל להכין עבור התוכנה קובץ התקנה ולהפוך את השרת שלנו לשרת עדכון עבור התוכנה Foobar, נבצע את הפעולות הבאות:

1. בקובץ Updater\settings.py, בשורות 20,21, יש לעדכן את המשתנים הבאים:
   * SOFTWARE\_NAME – שם התוכנה, בדוגמא שלנו נבחר ב- “Foobar”.
   * PROGRAM – שם קובץ ההרצה הראשי, בדוגמא שלנו נבחר ב- “foobar.exe”.
2. פתח cmd כאדמין בתיקיית ה-Installer (תיקייה בשם Server).
3. ליתר בטחון, נקה את כל התכנים הלא רלוונטיים מקבצי התקנה קודמים בעזרת הפקודה הבאה:

python3 installer.py setup clean

* + זהירות! הפקודה הזאת עשויה למחוק דברים שלא ניתן לשחזר אותם!

1. להרצת ה-Installer והכנת קובץ ההרצה, הרץ את הפקודה הבאה (הסבר פרמטרים מטה):

python3 installer.py setup create -s C:\Foobar Foobar <major> <minor> -i <SERVER\_DNS> -p <PORT>

* + בחרנו ב- C:\Foobar בתור תיקיית התוכנה שלנו.
  + בחרנו ב-Foobar בתור שם התוכנה שלנו (שעבורה קיים foobar.exe בתיקיית התוכנה).
  + Major מציין את מספר הגרסא הראשי.
  + Minor מציין את מספר הגרסא המשני.
  + Server\_DNS הוא כתובת ה-IP או שם ה-Domain של שרת העדכון
  + PORT הוא פורט הניהול של התוכנה שבו השרת והלקוחות יתקשרו ב-UDP.
  + זהירות! הפקודה הזאת תמחק מפתחות RSA ישנים במידה ונוצרו לפני כן עבור התוכנה הזו!

1. לאחר סיום הרצת ה-Installer, ובהנחה שהורץ כראוי, אמור להופיע בתיקייה Output שבתוך תיקיית ה-Server קובץ חדש בפורמט הבא: <SOFTWARE\_NAME>\_<major>.<minor>\_setup.exe. זהו קובץ ההתקנה של התוכנה Foobar!
2. לבסוף, בשביל להפוך את המכונה שעליה עבדנו לשרת עדכון, יש להתקין את התוכנה כפי שמשתמש רגיל היה עושה. ראה [הוראות התקנה למשתמש](#_התקנה) במידת הצורך.

### הכנת עדכון גרסא

לאחר הכנת קובץ ההתקנה, אתה כנראה תרצה לעדכן את התוכנה שלך. לצורך כך תצרך ליצור את קובץ העדכון לתוכנה. ניתן לעשות ע"י ביצוע הפעולות הבאות על שרת העדכון (אותו שרת שבו יצרת את קובץ ההתקנה):

1. לעדכן את תיקיית התוכנה בקבצים של הגרסא החדשה.
   * בדוגמא שלנו, יש לעדכן את כל הקבצים בתיקייה C:\Foobar לגרסא החדשה.
2. פתח cmd כאדמין בתיקיית ה-Installer (תיקייה בשם Server).
3. ליצירת קובץ העדכון, הרץ את הפקודה הבאה:

python3 installer.py update create C:\Foobar <major> <minor>

* + בחרנו ב- C:\Foobar בתור תיקיית התוכנה שלנו.
  + Major מציין את מספר הגרסא הראשי של העדכון.
  + Minor מציין את מספר הגרסא המשני של העדכון.
  + שים לב! הגרסא הזו חייבת להיות מתקדמת יותר מהגרסא הקודם (כלומר, שה-major גדול מקודמו, או שה-major שווה לקודמו וה-minor גדול מקודמו).
  + שם התוכנה (ועוד הרבה פרמטרים) נלקחים מקובץ ההגדרות settings.json או מה-registry

1. לאחר סיום הרצת הפקודה, ובהנחה שהורצה כראוי, אמור להופיע קובץ חדש בשם update.zip בתיקיית ה-updater במיקום שבו הותקנה התוכנה (בשלב 6 של [הכנת קובץ התקנה ויצירת שרת עדכון](#_הכנת_קובץ_התקנה)). הקובץ הזה הוא קובץ העדכון החדש. הוא רק נוצר, עדיין לא הופץ (למרות שלקוחות יכולים לבקש להוריד אותו מהשרת ביוזמתם).

### הפצת עדכון גרסא

לאחר הכנת קובץ עדכון לגרסא חדשה, אתה כנראה תרצה להפיץ אותו כדי שכל המחשבים (שרתים או לקוחות) באותה הרשת של שרת העדכון יקבלו את העדכון מבלי שיצטרכו לבקש אותו. ניתן לעשות זאת ע"י ביצוע הפקודה הבאה (באותו חלון שבו יצרנו את קובץ העדכון):

python3 installer.py update broadcast -s

באופן כללי, הפקודה הזאת שולחת broadcast לכל הרשתות של השרת, עליו היא רצה, ומדווחת על הגרסא העדכנית ביותר. הדגל -s שהוספנו אומר לכל המחשבים (לקוחות/שרתים) המקבלים את ההודעה להפיץ אותה הלאה לשאר הרשתות שהם מחוברים אליהם (וגם בהן המקבלים יפיצו אותה, וכן הלאה וכן הלאה...).

### בניית שרת עדכון נוסף

שדג

## הוראות שימוש למשתמש

### התקנה

שדג

### שימוש

שדג

### אבחון בעיות

שדג

# בדיקות וטסטים

שדג

# מצגת

שדג