

המחלקה למתמטיקה ומדעי המחשב

סמינר בתכנות מערכות דפנסיבי 20928

**תכנות דפנסיבי:**

**אבטחת מודל שרת - לקוח**

מוגש ע"י משה גרשנפלד

בהדרכת ד"ר ליאוניד בירנבוים

אוקטובר 2020

**תוכן עניינים**

מבוא.............................................................................................................2

פרק 1: ++C...................................................................................................3

היסטוריה...................................................................................................3

מחלקות ואובייקטים...................................................................................3

ירושה ........................................................................................................4

פולימורפיזם...............................................................................................5

פונקציות וירטואליות...................................................................................6

הרשאות.....................................................................................................8

שיפורי בטיחות ב-++C בהשוואה ל-C............................................................9

איומי אבטחה והתגוננות..............................................................................9

גלישת חצץ................................................................................................9

Use After Free.......................................................................................11

פרק 2: פייתון...............................................................................................12

היסטוריה.................................................................................................12

מחלקות ואובייקטים.................................................................................12

ירושה.......................................................................................................13

פולימורפיזם.............................................................................................15

פונקציות..................................................................................................18

הרשאות...................................................................................................19

מבני נתונים..............................................................................................19

הבדלים משמעותיים נוספים בין פייתון ל-++C.............................................20

איומי אבטחה והגנות מפניהם.....................................................................20

פרצות המנצלות את המבנה של PyPi..........................................................21

פרצות המנצלות תלויות שאינם מעודכנות...................................................21

פרצת Timing Attack...............................................................................22

פרק 3: תקשורת............................................................................................24

פרוטוקול TCP/IP......................................................................................24

איומי אבטחה והתגוננות............................................................................26

סודיות................................................................................................... 26

אי-התכחשות.........................................................................................28

אימות...................................................................................................28

פרק 4: איומי אבטחה מערכתיים....................................................................30

נוזקה.......................................................................................................30

התקפת מניעת שרות מבוזרת(DDos)...........................................................32

פרק 5: מימוש מערכת שרת לקוח...................................................................35

מימוש ללא הגנות......................................................................................35

מימוש עם הגנות........................................................................................40

סיכום..........................................................................................................43

מקורות........................................................................................................44

**מבוא**

במבט פשטני, היה צורך להפריד בין לימוד כתיבת קוד לבין לימוד אבטחת האפליקציה. מתכנת היה אמור להתמקצע בכתיבת קוד והכרת השפות השונות לעומק ולהשאיר את תחום האבטחה למומחי האבטחה שימצאו ויתקנו את פרצות האבטחה באפליקציה.

הניסיון מלמד שהפרדה בין אבטחה לכתיבת קוד שגויה ולא יעילה. רוב פרצות האבטחה, מקורם בפרצות ידועות שניתנות להגנה ברמת הקוד ע"י בחירת שפה והגנה מתאימה. הגנה מפרצות אלה קלה וזולה יותר לביצוע ברמת המתכנת מאשר ע"י בדיקת מומחי אבטחה. לכן, על כל מתכנת להכיר את פרצות האבטחה הנפוצות, ולתכנן את ההגנה כבר בעת כתיבת הקוד [1].

בעבודה זו אתמקד בכמה פרצות אבטחה נפוצות בשפת ++C ופייתון, וכן בפרצות אבטחה הקיימים ברשת, ובפרצות אבטחה כלליות.

בפרק הראשון של העבודה, אעביר סקירה קצרה על שפת ++C, וכן אפרט על שתי פרצות אבטחה נפוצות בשפה, שהם גלישת חצץ ו-Use after free, ואת דרכי ההגנה מהם.

בפרק השני אסקור בקצרה את שפת פייתון, וכן אפרט על פרצות אבטחה המנצלות את השימוש הנפוץ במאגר החבילות PyPi, ועל פרצת Timing Attack, וכן את דרכי ההגנה מפרצות אלו.

בפרק השלישי ארחיב על פרוטוקול התקשורת TCP/IP להעברת מידע ברשת, וכן אפרט על בעיות הבטיחות ברשת ומניעתם.

בפרק הרביעי ארחיב על סוגי הנוזקות הקיימות והגנה מפניהם, וכן על מתקפת DDos ומניעתה.

בפרק החמישי והאחרון אציג אפליקציית שרת לקוח, כאשר השרת ממומש בשפת פייתון והלקוח בשפת ++C. בשלב הראשון אציג את קוד האפליקציה ואציין את פרצות האבטחה הקיימות בה מבין פרצות האבטחה שמניתי בפרקים א-ד, ובשלב השני אציג קוד מתוקן ופתרונות לבעיות הבטיחות.

**פרק 1: ++C**

**היסטוריה:**

שפת ++C פותחה כהרחבה של שפת C בשנת 1980 ע"י ביארן סטרוסטרפ ממעבדות בל. בגרסה הראשונה היא נקראה "C with Classes" [2].

השפה הייתה מבוססת על שפת C, ובנוסף מיישמת את העקרונות של תכנות מונחה עצמים ע"י השימוש במחלקות [2].

בשנת 1980 שונתה שם השפה ל-++C . השינוי נועד לדייק את תיאור השפה, כשפה המבוססת על שפת C, אך שפה שגם מתרחבת באופן תמידי בכלים חדשים שאינם ב-C [2].

נציין חלק מהכלים שהתווספו לשפה במהלך השנים:

תכנות גנרי, ירושה מרובה, טיפול בחריגות, פונקציות אנונימיות ועוד [3].

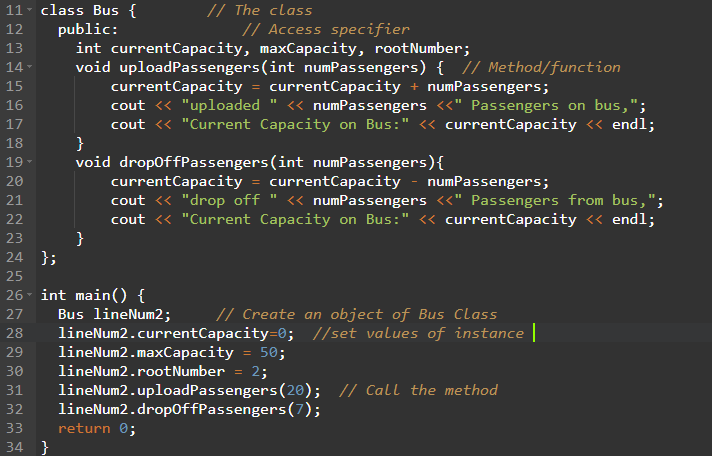
בשנת 1998 נקבע תקן מוסכם לשפה, ובכל כמה שנים מתפרסמת גרסה חדשה. הגרסה האחרונה שפורסמה - 20++C, פורסמה בשנת 2019 [3].

**מחלקות ואובייקטים:**

מחלקה מייצגת ישות לוגית, כגון אוטובוס או אוניברסיטה. בכל מחלקה קיימים משתנים שהם התכונות של הישות הלוגית, כגון מספר הקו של האוטובוס ומספר הנוסעים המקסימלי באוטובוס, וכן מספר התלמידים באוניברסיטה. כמו כן בכל מחלקה קיימות פונקציות שמייצגות פעולות של הישות, כגון העלאת והורדת נוסעים, וכן הוספת סטודנט לאוניברסיטה.

אובייקט הוא מופע ספציפי של המחלקה, לדוגמה: קו 2 הוא מופע ספציפי של הישות "אוטובוס", וכן "האוניברסיטה הפתוחה" היא מופע של הישות "אוניברסיטה".

נראה דוגמה של מחלקה המייצגת אוטובוס אשר יש לה 3 תכונות: מספר הנוסעים המקסימלי, מספר הנוסעים שנוסעים כרגע, וכן מספר הקו. וכן האוטובוס יכול לעשות 2 פעולות שהם הורדת והעלאת נוסעים. בנוסף נראה מופע ספציפי של האוטובוס.



++C מוגדרת כשפת מונחה עצמים, והמחלקות והאובייקטים הם המסגרת הבסיסית של שפת תכנות מונחה עצמים.

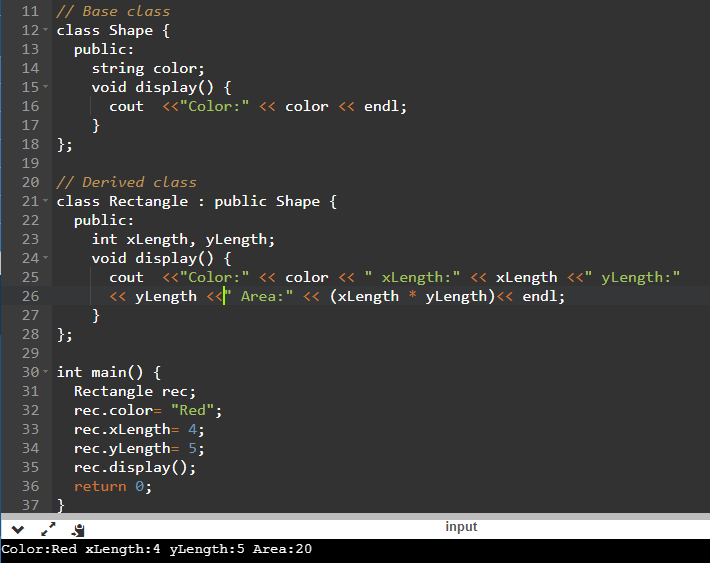
היתרונות של שפה שהיא שפת תכנות מונחה עצמים היא בפרודוקטיביות הרבה של המשתמשים בה, בגלל הקלות בהבנת ותיקון הקוד, וכן בגלל האפשרות לביצוע שימוש חוזר בקוד [2], [4].

**ירושה**

ניתן להגדיר מחלקה חדשה שיורשת את כל התכונות והפונקציות של מחלקה קיימת. מבצעים ירושה זו, כאשר המחלקה היורשת היא "סוג של" המחלקה המורישה, לדוגמה: כאשר המחלקה המורישה מייצגת צורה, והמחלקה החדשה מייצגת מלבן, הישות מלבן הוא "סוג של" צורה, ולכן נבצע ירושה למלבן, שירש את התכונות והפעולות של הישות שמייצגת צורה.

במקרה של ירושה, המחלקה היורשת יכולה לשנות את הפונקציות שירשה ע"י דריסת הפונקציה.

נראה דוגמא לירושה ודריסה, ע"י מחלקה המייצגת צורה ומחלקה המייצגת עיגול. כאשר המחלקה המייצגת עיגול יורשת מהמחלקה המייצגת צורה את התכונה של צבע, ודורסת את הפונקציה:



השימוש בירושה מאפשר לקיים את אחד התכונות החשובות של תכנות מנחה עצמים והיא האפשרות לשימוש חוזר בקוד.

C++ מאפשר גם ירושה מרובה (ירושה מכמה מחלקות במקביל), בשונה משפות אחרות של תכנות מנחה עצמים כגון JAVA שלא מאפשרת ירושה מרובה [4].

**פולימורפיזם**

פולימורפיזם מאפשר לכתוב קוד בצורה כללית ללא צורך בכתיבת קוד בצורה ספציפית [4].

משמעות המילה פולימורפיזם היא צורות רבות, ופולימורפיזם מאפשר לבחור אחד מתוך הצורות הרבות.

בשפת ++C אפשר לחלק את הפולימורפיזם ל-2:

1. פולימורפיזם בזמן הידור ע"י העמסת פונקציות והעמסת אופרטור.

2.פולימרפיזם בזמן ריצה ע"י פונקציות וירטואליות.

נראה דוגמה לפולימורפיזם בהעמסת פונקציות:



לאובייקט חיה יש שתי מתודות eat, כאשר הם שונים רק במספר הארגומנטים. בזמן ההידור המהדר בוחר את המתודה המתאימה לפי מספר הארגומנטים.

**פונקציות וירטואליות**

כאשר מפעילים פונקציה על אובייקט בקוד במקרה שיש כמה פונקציות שונות עם אותם שמות וסוג החזרה וכן הארגומנטים שלהם שווים, בחירת הפונקציה שתופעל יכולה להתבצע בשני זמנים שונים, הבחירה יכולה להתבצע בזמן ההידור(Static Binding) או בזמן הרצת התוכנית(Dynamic Binding).

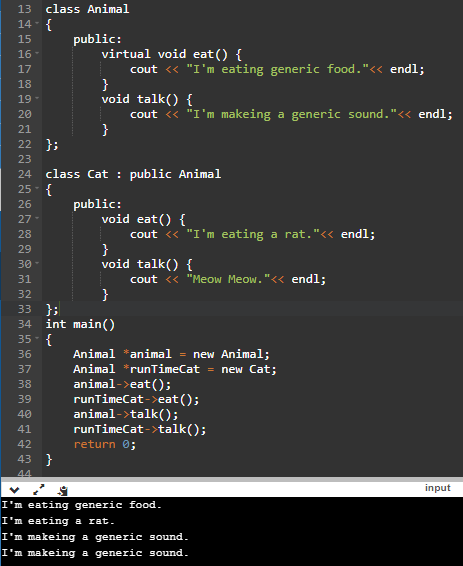
אם הבחירה מתבצעת בזמן ההידור, המהדר בוחר את הפונקציה שתופעל, והמהדר בוחר לפי המשתנה של האובייקט.

אם הבחירה מתבצעת בזמן הריצה, הפונקציה נבחרת לפי סוג האובייקט בזמן ריצה.

בפונקציה וירטואלית כאשר האובייקט הוא מצביע, הבחירה מתקיימת בזמן ריצה ובפונקציה שאינה וירטואלית הבחירה מתבצעת בזמן ההידור.

פונקציה מוגדרת וירטואלית אם מוסיפים את המילה virtual. אחרת, הפונקציה מוגדרת בברירת מחדל כnonvirtual-.

בחירת הפונקציה בזמן ריצה מאפשרת לכתוב קוד טוב יותר ע"י בחירת הפונקציה המתאימה לאובייקט בפועל.

נראה דוגמה:

בדוגמה הזאת הפונקציה eat היא פונקציה וירטואלית והפונקציה talk היא פונקציה לא וירטואלית. runTimeCat הוא מצביע מסוג Animal אבל בפועל הוא מסוג Cat. כאשר קוראים על runTimeCat את הפונקציה הוירטואלית, הפונקציה הנבחרת היא של הסוג שהיא בזמן הריצה(cat), אבל כשקוראים עליו את הפונקציה הלא וירטואלית, הפונקציה הנבחרת היא של סוג המצביע( Animal).

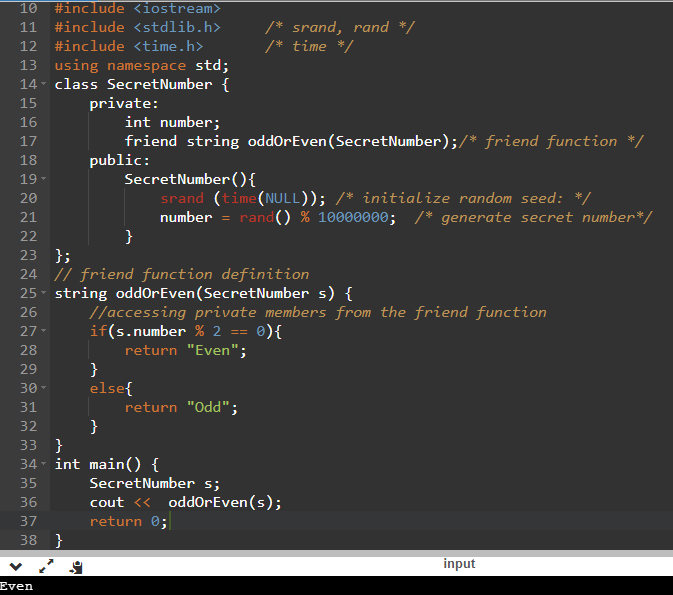
**הרשאות**

קיימים 3 משתני גישה [4] המגדירים את היכולת של משתמשים לגשת לפונקציות ומשתנים במחלקה:

* Public: כולם יכולים לגשת
* Private: רק המחלקה עצמה יכולה לגשת
* Protected: רק המחלקה עצמה ומחלקות שיורשות ממנה יכולות לגשת

בנוסף יש אפשרות להגדיר מחלקה או פונקציה כחבר(Friend) , כך שהמחלקה או הפונקציה שהוגדרו כחבר יכולים לגשת גם למשתנים שהם private.

נראה דוגמה לפונקציה המוגדרת כחבר של מחלקה, כאשר למחלקה יש משתנה המוגדר private, והפונקציה ניגשת למשתנים אלו:



המחלקה SecretNumber מייצגת מספר סודי. המספר מוגדר כפרטי ומקבל ערך רנדומלי. הפונקציה oddOrEven היא פונקציה גלובלית, המחזירה האם המספר הסודי הוא זוגי או אי זוגי. הפונקציה מוגדרת כפונקציית חבר במחלקה SecretNumber, ולכן יכולה לגשת למספר שמוגדר פרטי.

**שיפורי בטיחות ב-++C בהשוואה ל-C:**

נציין 2 שיפורי אבטחה משמעותיים שיש ב-++C בהשוואה ל-C:

1. כמוס\הרשאות: ע"י שימוש במחלקות, המימוש של האובייקט מוסתר. וכן משתמש חיצוני יכול לגשת למשתני המחלקה רק באופן מבוקר ע"י פונקציה מתאימה.
2. גלישת חצץ: פרצת האבטחה קיימת גם ב++C , אך בשימוש בספריות מיוחדות כגון TSL, וע"י טיפול בחריגות שהתווספו לשפת ++C, אפשר להוסיף בדיקה וטיפול בגלישה במקרה שהתגלתה, כך שלא יהיה גלישת חצץ [3].

**איומי אבטחה והתגוננות:**

אפרט 2 פרצות אבטחה נפוצות בשפת ++C, ואת דרכי ההגנה מהם:

גלישת חוצץ

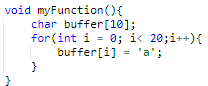
גלישת חוצץ היא שגיאת תכנות נפוצה, אשר מתקיימת כאשר נכתב לחוצץ יותר מידע מגודל החוצץ. שגיאה זו מאפשרת לדרוס מידע שמאוחסן בזיכרון אחרי החוצץ. במקרה הטוב המידע שנדרס הוא "זבל" ובמקרה הגרוע נדרס מידע סודי או קריטי. פורץ יכול גם להשתמש בפרצת האבטחה להשתלת קוד זדוני בזיכרון [5], [6].

הפרצה מתאפשרת רק בשפות שהם לא Type-safe כגון C ו- C++ ,אבל בשפות שהם Type-safe אין אפשרות לגלוש למידע שמחוץ לחוצץ, כי החלק שמחוץ לחוצץ לא מוגדר, ובשפות אלו אין אפשרות לגשת למידע שאינו מוגדר [7].

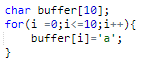
גלישת זיכרון לא גורמת בהכרח לפריצת אבטחה. מבחינת השפה אין קביעה מה יתבצע בעת גלישת חצץ, וכל קומפיילר מגדיר לכל סוג של גלישת חצץ איך התוכנית תגיב.

נפרט סוגים שונים של גלישת חוצץ [6], [7]:

* גלישת מחסנית: גלישת חוצץ שקורת במחסנית הקריאות. במקרה הגרוע, פורץ יכול לדרוס את כתובת החזרה של המחסנית כך שיצביע על קוד של קוד זדוני, ויגרום להרצת הקוד הזדוני. נראה דוגמא לגלישת חוצץ במחסנית:



* Off-By-One: גלישת חוצץ שחרגה וגלשה מהחוצץ רק ביחידה אחת. שגיאה זו היא אחת מהטעויות הנפוצות ביותר של מתכנתים צעירים(אבל לא רק). השגיאה נפוצה בעיקר בקוד של תנאי בלולאה.

לדוגמה:

* גלישת ערימה: גלישת חוצץ שקורת בזיכרון הערימה. בדרך כלל בזיכרון ערימה אין כתובות חזרה, ולכן השתלת קוד זדוני קשה יותר לביצוע. נראה דוגמא לגלישה בזיכרון ערימה:



* Format String Attack [8]:

פרצה זו משתמשת בשגיאת תכנות ובכשל שקיים בFormat String- שלא בודק את הקלט שמכניס המשתמש.

ניקח לדוגמה את הפונקציה printf שמשתמשת ב- format string. שימוש נכון בפונקציה הוא: printf("%s", input)"" , אבל מתכנת עצלן שכותב printf(input)"" ומשמיט את הארגומנט הראשון, אכן יקבל ברוב המקרים תוצאה דומה למקרה הראשון. אבל לא בכל המקרים יתקבל תוצאה דומה, לדוגמה, אם בארגומנט נמצא מציין תבנית(format specifier), בגלל שאין משתנה בארגומנט השני, ה"מציין תבנית" יוחלף במשתנה רנדומלי הנמצא במחסנית ולא במשתנה ספציפי. אחד מהמשתנים במחסנית הוא כתובת החזרה של המחסנית.

במקרה שהארגומנט input שומר קלט מהמשתמש, פורץ יכול להכניס את הקלט "%x%x%x%x%x%x",וכך הפורץ יוכל לגלות את כתובת החזרה של המחסנית ולנסות לדרוס את הכתובת בכתובת של מצביע לקוד זדוני.

הגנה מגלישת חוצץ:

אין הגנה הרמטית מגלישת חוצץ, ויש לנסות להגן ע"י שימוש באסטרטגיות שונות [6], [7]. נפרט כמה מהם:

* שימוש בספריות בטוחות: קיימים ספריות ידועות שאינם בטוחות כי אינם מבצעות בדיקה האם קיים חריגה מגבול המערך, ולכן מומלץ לא להשתמש בהם(gets scanf, strcpy), אלא להשתמש בספריות בטוחות. קיימות ספריות בטוחות מיוחדות למחרוזות ומערכים, שבהם קורות רוב הטעויות שמובילות לגלישה, ולכן יש להקפיד להשתמש בספריות הבטוחות האלו.( "The Better String Library", Vsr ו-Erwin).
* Deep Packet Inspection (DPI): אסטרטגיה זו נועדה להגן ממתקפה דרך הרשת שמתאפשרת בגלל גלישת חוצץ. DPI סורק את המידע המגיע דרך הרשת(פאקטים), וחוסם מתקפות המשתמשות בנקודות תורפה ידועות של גלישת חצץ.
* Executable space protection: אסטרטגיה זו מונעת את האופציה להחדיר קוד זדוני למחסנית או לערמה. כאשר מתבצע ניסיון להריץ קוד בזיכרון שהוסף ע"י המשתמש, התוכנית תזרוק חריגה(exception). מיקרוסופט פיתחה שתי תוכנות שמשתמשות באסטרטגיה זו – Buffer Shield ו-Stack Defender.
* Address Space Layout Randomization: אסטרטגיה זו מקשה על הפורץ לדרוס כתובות חזרה או מידע חשוב בזיכרון. ע"פ אסטרטגיה זו, בכל הפעלה של התוכנית, כתובות הזיכרון של קובץ ההרצה, המחסנית והערימה יהיו במיקום שונה בזיכרון, כך שהפורץ לא יכול לדעת את המיקום של הקוד שהוא מעוניין לדרוס.
* Pointer Protection: חלק גדול מהפריצות מתבצעות ע"י כך שהפורץ יודע את הכתובות של המצביעים (כתובות החזרה או פונקציות), ולכן בשיטה הזו, הקומפיילר מסתיר את הכתובות האמיתיות של המצביעים ע"י ביצוע XOR למצביעים.

Use-After-Free (UAF):

UEF היא פרצת אבטחה הקשורה בשימוש לא נכון בזיכרון דינאמי. לאחר שחרור זיכרון שהוקצה למצביע, המתכנת לא מבצע איפוס למצביע (השמה לnull-),וכך המצביע עדיין מצביע למקום בזיכרון [9], [10].

בגישה למצביע שבוצע לו שחרור זיכרון, ההתנהגות מוגדרת כUndefined behavior , ובדרך כלל גורמת לקריסה.

אם לאחר שחרור הזיכרון הוקצה הזיכרון למידע אחר, מידע זה יכול להימחק במקרה שמתבצע שימוש חוזר במצביע. וכן פורץ יכול לגלות את המידע או להשתיל קוד שיגרום להשתלטות על המחשב, לדוגמה: כאשר הזיכרון הוקצה למצביע לפונקציה, וע"י שימוש במצביע הראשון, נכניס בזיכרון כתובת של קוד זדוני, כך שבמקום הפונקציה יורץ הקוד הזדוני.

הגנה מ-UEF:

בכדי להימנע מ-UAF עדיף להשתמש במצביע חכם (נוסף לשפה בגרסה 11++(C. מצביע חכם משחרר את הזיכרון באופן אוטומטי, ולכן כאשר משתמשים במצביעים חכמים, לא קוראים כלל לפונקציית free. אסטרטגיה זו מונעת הרבה טעויות שגורמות ל-UAF, אבל קיימים מצבים, שגם בשימוש במצביע חכם יהיה UEF [11].

הגנה נוספת מתאפשרת ע"י תוכנות שסורקות ומגלות UEF בקוד. נציין כמה תוכנות כאלה:

Microsoft's Visual C/C++ debugger, Valgrind, Polyspace.

**פרק 2: פייתון**

**היסטוריה**

בסוף שנות השמונים Guido Van Rossum התחיל לפתח את שפת פייתון, כאשר הבסיס לשפה הייתה שפת ABC [12].

השם של השפה "פייתון" נבחר על שם סדרת טלוויזיה מצליחה Monty Pythons Flying Circus שאותה Guido מאד חיבב [12].

העיקרון המנחה של השפה היא, שתהא שפה קריאה, וכן שפה חסכונית בכתיבת קוד, כך שתאפשר כתיבת תוכנית בדרך קצרה [13].

בשנת 1999 פורסם מסמך Zen of Python שבו מפורטים 19 עקרונות של השפה, וביניהם:

* " Simple is better than complex "
* " Special cases aren't special enough to break the rules. "
* " There should be one—and preferably only one—obvious way to do it. "

בשנת 1991 שוחררה הגרסה הראשונה של פייתון, הגרסה השנייה שוחררה בשנת 2000, וב- 2008 שוחררה הגרסה השלישית והאחרונה עד עתה [12].

פייתון 2 איננה בפיתוח עוד, וכן הופסקה לה התמיכה מהשנה הנוכחית (2020) [12].

פייתון 3 לא מותאם לאחור לפייתון 2, כך שלא בהכרח קוד שרץ בפייתון 2 ירוץ בפייתון 3 [12].

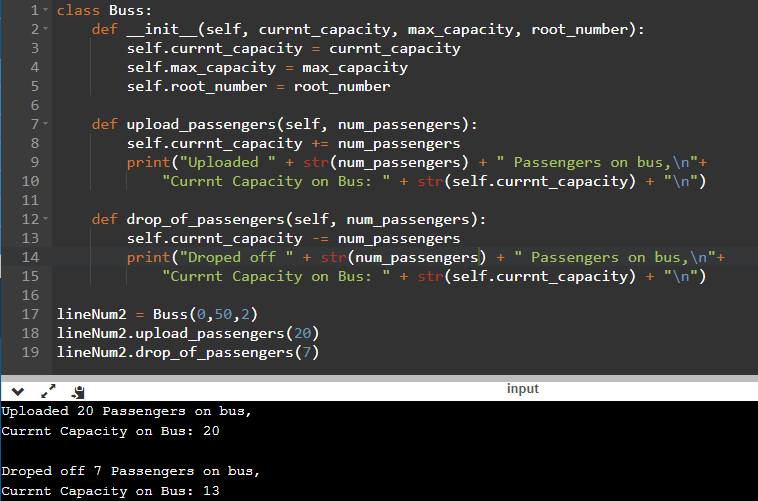
**מחלקות ואובייקטים**

פייתון היא שפת תכנות מונחה עצמים, המשתמשת במחלקות ואובייקטים, כמו ב-++C.

בשונה מ++C שצריכה לעבור הידור של כל התוכנית, בפייתון ישנה אפשרות להריץ שורה בודדת של קוד, אשר מורצת מיידית. בנוסף, פייתון מאפשרת לבצע שינויים במחלקה גם אחרי שהמחלקה נוצרה [14].

בפייתון "Everything is an Object" ולכן גם פונקציות ומחלקות הם אובייקטים, וקיים רק טיפוס אחד – אובייקט. בשונה מ++C שהיא בעלת טיפוסיות סטטית(statically typed), המתאפיינת בכך שלכל משתנה יש טיפוס נתונים ויכול להכיל רק ערכים מאותו טיפוס, וכן שההתאמה בין הערך לטיפוס נבדקת בזמן ההידור, בפייתון הטיפוס הוא דינמי(dynamic typing), קיים טיפוס יחיד – אובייקט, ולכן יכול להכיל את כל הערכים(int,מחלקה, פונקציה וכדומה), ובדיקת התקינות מתבצעת רק בזמן ריצה(אם נקבע הערך של משתנה ע"י פעולה מתמטית על מחרוזת ומספר – התוכנית תקבל שגיאה בזמן ריצה) [14].

נראה דוגמה של מחלקה ואובייקט המייצגים אוטובוס שהראנו בפרק 1, בפייתון:

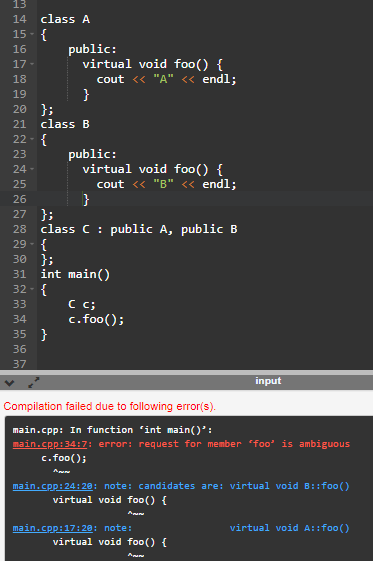


**ירושה**

בדומה ל++C פייתון מאפשר ירושה מרובה.

ירושה מרובה יוצרת אתגר כאשר המחלקה היורשת יורשת פונקציות בעלות אותו שם ממחלקות שונות. מסיבה זו קיימות שפות שאינם מאפשרות ירושה מרובה כגון גאוה ו#C.

ב++C יתקבל שגיאת קומפילציה בקריאת פונקציה במחלקה שירשה את הפונקציה משני מחלוקות שונות. נראה דוגמה:

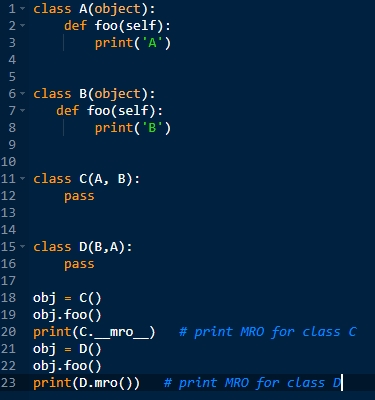
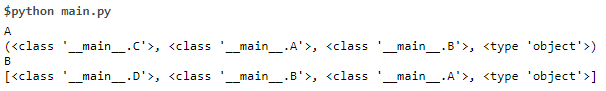


פייתון מתמודד עם הבעיה באופן שונה ע"י method resolution order (MRO).

MRO היא רשימה של מחלקות שהמחלקה הנוכחית ירשה אותם. הקדימות נקבעת לפי הסדר שבהם המחלקות מופיעות בקוד כאשר לכל אחד מהם נעשה אלגוריתם חיפוש לעומק(DFS). בנוסף אם קיימת מחלקה שמחלקה אחרת שיורשת ממנה נמצאת אחריה ברשימה אז מוציאים אותה מהרשימה.

לכל מחלקה פייתון מחשב את ה-MRO שלה, וכאשר קיימת פונקציה בעלת שם דומה ניתנת עדיפות למחלקה שנמצאת ראשונה ברשימת ה-MRO.

נראה דוגמה:



בדוגמה הזאת המחלקה C יורשת קודם ממחלקה A ולכן הפונקציה של מחלקה A נבחרת, אבל מחלקה D יורשת קודם ממחלקה B ולכן היא נמצאת לפני A ברשימה וזו גם הסיבה שהיא הפונקציה שנבחרת בקריאת D.

**פולימורפיזם**

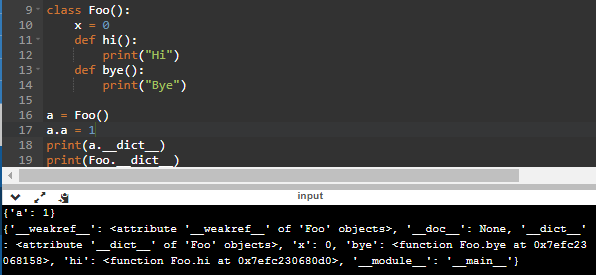
בשונה משפת ++C שפונקציה שהיא לא וירטואלית נבחרת בזמן ההידור ע"פ סוג המשתנה, בפייתון כל המשתנים הם מסוג אובייקט ולכן אי אפשר לבחור את הפונקציה המתאימה ע"פ סוג המשתנה. לכן בפייתון כאשר ישנם פונקציות עם אותם שמות ומספר הארגומנטים שלהם שווים, בחירת הפונקציה נעשית תמיד בזמן ריצה ע"פ סוג האובייקט בפועל.

להבנת האלגוריתם של פייתון לבחירת הפונקציה נצטרך להסביר קודם על המילון שנשמר לכל אובייקט וכן על Descriptor:

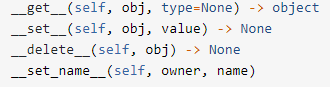
מילון -\_\_dict\_:

לכל אובייקט בפייתון קיים מילון שבו נשמרות התוכנות של האובייקט בזמן ריצה. המפרש של פייתון משתמש במילונים של האובייקטים למצא התאמה לפונקציה\תוכנה המבוקשת.

נראה דוגמא למילון של מחלקה ושל מופע של מחלקה:

בדוגמה הזאת המילון של המופע a מכיל רק את המשתנה a ולא את המשתנה x, כי x היא תכונה של המחלקה Foo ולא של a, ולכן x מופיע רק במילון של המחלקה ולא במילון של המופע.

Descriptor:

descriptor הוא אובייקט המממש את הפונקציות של descriptor protocol. הפונקציות הם:

ישנם 2 סוגי Descriptor:

1. Data Descriptor: אובייקט המממש את הפונקציות \_\_set\_\_ או \_\_delete\_\_
2. Non-Data Descriptor: אובייקט המממש את הפונקציה \_\_get\_\_.

עכשיו נפרט את מהלך בחירת הפונקציה\תוכנה בפייתון:

1.אם המופע הוא Data Descriptor, נפעיל את הפונקציה \_\_get\_.

2.אחרת, נבדוק **במילון** של **המופע** אם מופיע תוכנה בעלת אותה שם ונבחר אותה.

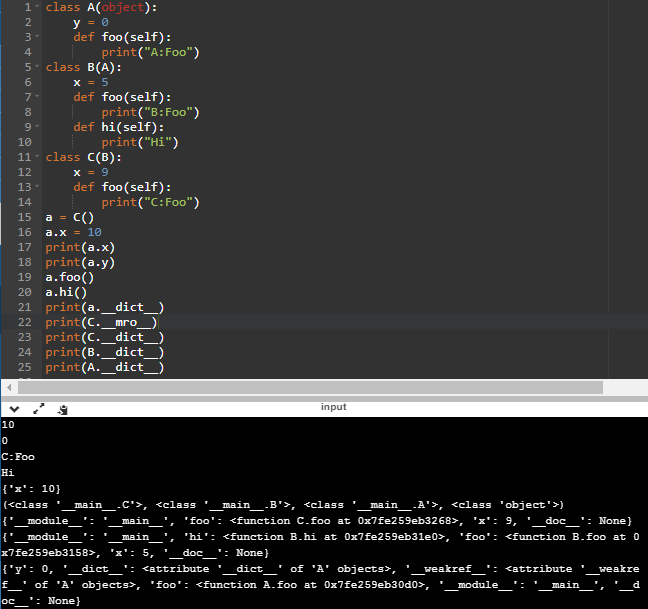
3.אחרת, אם המופע הוא Non-Data Descriptor,נפעיל את הפונקציה של \_\_get\_\_.

4.אחרת, נחפש לפי הסדר של רשימת ה**MRO** – של **המחלקה** של המופע. לכל מחלקה נחפש **במילון** שלה האם קיימת תוכנה בעלת שם דומה ונפעיל אותה.

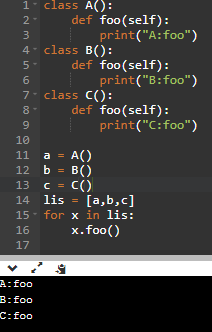
5.אחרת, אם קיים פונקציה \_\_getattr\_\_ , נשלח לפונקציה הזאת.

6. אחרת, נחזיר שגיאה.

נראה דוגמה הממחישה את הבחירה הזאת:



בדוגמה הזאת, בקריאה לa.x בודקים קודם במילון של המופע ולכן יודפס 10. בקריאה a.y, בודקים במילון של המופע ולא מוצאים ולכן עוברים למילון של C שנמצא ראשון ברשימת MRO אבל גם שם אין תוכנה בעלת שם y, ולכן עוברים למילון של B שנמצא אחריו ברשימה ומוצאים את התוכנה y. ועל זו הדרך מתבצעת גם בחירת הפונקציות של foo ו-hi.

נראה עוד דוגמא כללית של פולימורפיזם, שהמפרש בוחר בזמן ריצה את הפונקציה לפי המופע:

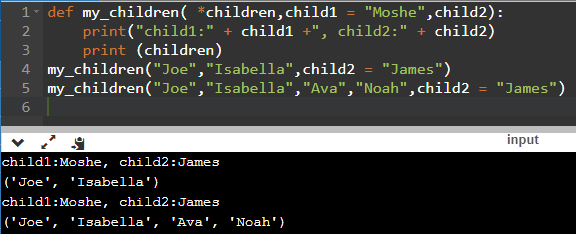
**פונקציות**

פייתון מאפשרת לקבוע אופציית ברירת מחדל לארגומנט של הפונקציה, כך שאם נקרא לפונקציה ללא הארגומנט, משתנה הארגומנט יקבל את ההשמה של ברירת המחדל.

בקריאת הפונקציה אפשר להשתמש בkeyword arguments- שמשפר את קריאות הקוד, וגם חוסך כתיבת ארגומנטים במקרה שיש להם אופציית ברירת מחדל.

בנוסף, פונקציות בפייתון יכולות לקבל מספר לא קבוע של ארגומנטים, וכן מספר לא קבוע של keyword arguments [15].

נראה דוגמה של פונקציה שמקבלת ארגומנטים מסוגים שונים:



בדוגמה זו, הארגומנט הראשון של הפונקציהchildren , מוגדר כארגומנט ללא מספר קבוע של ארגומנטים. במימוש בשורה 4 הפונקציה קיבלה 2 ארגומנטים ובשורה 5 הפונקציה קיבלה 4 ארגומנטים.

הארגומנט השני של הפונקציה child1, הוא ארגומנט מסוג keyword arguments עם אופציית ברירת מחדל. והארגומנט השלישי הוא גם ארגומנט מסוג keyword arguments אבל הוא ללא אופציית ברירת מחדל.

**הרשאות**

אחד מההבדלים הגדולים בין פייתון לC++ הוא באפשרות לגישה למשתנים במחלקה. בפייתון אין דרך למנע גישה ולהסתיר מידע במחלקה מגישה של המשתמש. אבל קיימת מוסכמה שמשתנה או פונקציה שמתחילה בקו תחתון, המתכנת יתייחס אליה כמשתנה private [14].

**מבני נתונים**

בפייתון קיימים 4 מבני נתונים שונים [16]: רשימה, טאפל (tuple), קבוצה, מילון(dictionary).

ההבדל המשמעותי בין טאפל לרשימה, שטאפל אינו ניתן לשינוי , ואי אפשר להוסיף או למחוק ולשנות מהטאפל, משא"כ ברשימה ניתן להוסיף לרשימה קיימת, וכן ניתן לשנות פריטים ברשימה קיימת.

מבנה הנתונים מסוג קבוצה, מיוחד בכך, שאין חזרה על איברים הנמצאים בקבוצה, וכן ניתן לעשות על הקבוצה פעולות מתמטיות כגון איחוד חיתוך וחיסור . בנוסף אין חשיבות לסדר האברים, כך שלא ניתן לגשת לאיבר בקבוצה לפי מיקום האיבר בקבוצה [16].

מילון הוא מבנה הנתונים המשמעותי והחשוב ביותר מבין מבני הנתונים [16], ויכול לשמש בתור טבלת גיבוב. מילון מורכב ממפתח וערך המשויך אליו, ועל המפתחות להיות ייחודיים (הערכים המשויכים אליהם אינם צריכים להיות ייחודיים). וכן אפשר לגשת לערך ע"י המפתח המשויך אליו.

נראה דוגמא של 4 מבני הנתונים:

**הבדלים משמעותיים נוספים בין פייתון ל++C**

* פייתון היא שפת סקריפטים – מתרגמת לשפת מכונה רק בזמן ריצה.
* אין צורך להכריז על משתנה לפני שימוש בו, בניגוד ל++C.
* בפייתון קיים מנגנון איסוף זבל, בשונה משפת ++C שלא קיימת לה מנגנון איסוף.

**איומי אבטחה והגנות מפניהם**

חבילות צד שלישי

מפתחים בשפת פייתון נעזרים בחבילות קוד שפותחו ע"י צד שלישי. המאגר הרשמי והעיקרי לחבילות צד שלישי בפייתון הוא PYPI.

כל מפתח יכול להוסיף חבילות ל-PYPI, ולא מתבצעת בקרה ובדיקה על החבילות הנכנסות למאגר.

פרצות המנצלות את המבנה של PYPI:

המבנה של המאגר מקל על פורצים להחדיר קוד זדוני לקוד [17]. נראה שלושה דרכים שהפורץ יכול לנצל את המבנה של PYPI, להחדיר קוד זדוני לחבילה:

* Bait attack: באסטרטגיה זו, הפורץ כותב חבילה, ומשתיל לתוכה קוד זדוני, וע"י פיתיון גורם למשתמש להוריד את החבילה. הפיתיון יכול להיות ע"י יצירת חבילה שהפונקצונליות שלה תעניין משתמשים, או ע"י יצירת חבילה בעלת שם דומה לחבילה פופולרית אחרת, וכדומה.
* Direct attack: באסטרטגיה זו, הפורץ לא כותב חבילה חדשה, אלא משתלט על חשבון של מפתח של חבילה שנמצאת במאגר, ומוסיף לה קוד זדוני. הפורץ נעזר במבנה של PYPI, שחלק גדול מאד מהחבילות שנמצאות בה "מתות", ולא נעשה בהם עדכון במשך זמן רב. מה שמאפשר לפורץ להוסיף קוד זדוני, ובעל החשבון לא יבחין בכך.
* Influencer attack: בשיטה זו, הפורץ מנצל את מבנה החבילות בפייתון, שכל חבילה תלויה בחבילות אחרות, כך שהפורץ לא צריך להחדיר קוד זדוני לחבילה עצמה, אלא מחדיר קוד זדוני לחבילה אחרת, שהחבילה הזאת תלויה בה.

הגנה מפני פרצות המנצלות את המבנה של PYPI:

בכדי לסתום את הפרצות הללו, מומלץ שמתכנתי pip (pip היא מנהלת ההתקנות של החבילות) יוסיפו פיצ'רים לתוכנה שלהם, וכן לדאוג שמפתחי החבילות והמשתמשים בחבילות ישמרו על מספר כללים [17].

ל-pip מומלץ להוסיף 2 פיצ'רים שיזהירו את המשתמשים בעת הורדת חבילות. הפיצ'ר הראשון, יבצע סריקה בעת התקנת החבילות, ויזהיר את המשתמש אם קיימת חבילה פופולרית יותר, בעלת שם דומה לחבילה שהמשתמש מעוניין להוריד, כך המתכנת ימנע מלהוריד את החבילה במקרה שטעה באיות של שם החבילה. הפיצ'ר השני יזהיר את המשתמש במקרה שהחבילה שהוא מעוניין להוריד לא שונתה במשך תקופה ארוכה, כך יוכל המשתמש להימנע מ- Direct attack.

בנוסף להוספת הפיצ'רים, מפתחי החבילות צריכים להיות ערניים לפריצות בחשבון ולשינויים בקוד, וכן להקשות על פריצה לחשבון ע"י שינוי תקופתי של סיסמת החשבון, ובחירת סיסמאות ייחודיות[17].

המשתמשים בחבילות צריכים להקפיד על איות נכון של החבילות, ולבדוק לעומק חבילות חשודות [17].

פרצות המנצלות תלויות שאינם מעודכנות:

כאשר מתכנת מוצא באג בחבילה, הוא מתקן את הקוד, ומעדכן את החבילה. משתמש שהוריד את החבילה לפני שהמתכנת תיקן את הקוד, נשאר עם הבאג בחבילה כל עוד שלא הוריד שוב את החבילה המעודכנת.

פורץ יכול לנצל מצב זה, ע"י תקיפת חבילות שתלויות בחבילה שבוצע לה עדכון בגלל באג, ולקוות שהמתכנת לא הוריד מחדש את החבילה המעדכנת. כך נשאר הבאג בתוכנה, והפורץ יכול לנצל אותה [18].

הגנה מפני פרצות המנצלות תלויות שאינם מעודכנות

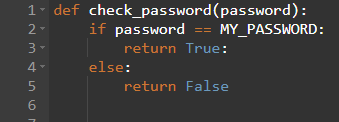
ההגנה הטובה ביותר היא, להשתמש בכלים ייעודים שמעדכנים באופן אוטומטי את התלויות [18].

Timing Attack

Timing attack היא התקפה המנצלת פגמים בקוד למצא רמזים לפצח סיסמאות [19].

התקיפה מתבצעת ע"י שהתוקף מודד את זמני התגובה של התוכנית לקלטים שונים שהוא מכניס ולפי זמני התגובה השונים התוקף מגלה רמזים שעוזרים לפצח את הסיסמא.

נראה התקפה בסיסית של Timing Attack על קוד פגום:



פעולת ההשווה ('==') משווה את האות הראשונה של password לאות הראשונה של MY\_PASSWORD וכל עוד שהאותיות שוות היא משווה בין האותיות באינדקס העוקב. בפעם הראשונה שמתגלה אי שוויון, הפעולה '==' מחזירה False ולא ממשיכה להשוות את האותיות באינדקסים הבאים.

פורץ יכול לנצל זאת ע"י שימדוד את הזמן שלוקח לתוכנית להחזיר False . לדוגמא הפורץ יכול להכניס סיסמא בת שתי אותיות, ויעשה זאת X פעמים, כאשר X הוא מספר התווים השונים שהסיסמא מקבלת. באות הראשונה הפורץ יכניס כל פעם תו שונה ובאות השנייה יכניס תמיד תו קבוע. באחד מהסיסמאות שהפורץ הכניס, התו הראשון תואם לסיסמא האמיתית, ולכן התוכנית תשווה גם בין האותיות במקום השני ותבצע סה"כ שתי השוואות, אבל בכל שאר הסיסמאות התו הראשון לא תואם לסיסמא האמיתית, ולכן התוכנית תבצע רק השוואה אחת ולכן כאשר הפורץ ימדוד את הזמנים הוא יגלה שתוכנית אחת רצה יותר זמן מהשאר, וכך יגלה את התו הראשון של הסיסמא. הפורץ יכול להמשיך בדרך הזאת ולגלות את כל הסיסמא.

**מניעת ההתקפה:**

הדרך למנע את ההתקפה היא ע"י שהתוכנית תחזיר תשובה אחרי זמן קבוע – זמן שלא תלוי בקלט. לדוגמא, תמיד נשווה את כל הקלט אפילו אם כבר גילינו אי שוויון, וכן תמיד נשווה בין שתי מחרוזות באורך שווה, כך שלא יתגלה אורך הסיסמא.

אופציה נוספת היא, שאם לוקח לתוכנית להחזיר תשובה במקרה הגרוע ביותר X זמן, אז תמיד התוכנית תחזיר תשובה רק לאחר X זמן.

פייתון בגרסה 3.3 הוסיפה את הפונקציה hmac.compare\_digest(a, b) שמשווה באופן בטוח בין שתי מילים שהם באורכים זהים.

**פרק 3: תקשורת**

העברת מידע בין מחשבים ללא העברה פיזית של מידע, מוגדרת כתקשורת מחשבים.

העברת המידע בין המחשבים מתבצעת ע"י כבלים, Wi-Fi או תקשורת אלחוטית וכדו'.

**פרוטוקול TCP/IP**

חבילת פרוטוקולי תקשורת שעליה מושתת רשת האינטרנט, הפרוטוקול פותח בשנות ה-70 וה-80 בידי חוקרים של משרד ההגנה האמריקאי .

שני עקרונות חשובים הנחו את מפתחי הפרוטוקול:

1. עמידות הרשת: לאפשר העברת מידע בין מחשבים ברשת, גם כאשר חלק מהנתבים יהרסו ויושמדו ע"י תקיפה.
2. רשת לא מוגבלת: הרשת לא תהא מוגבלת לסוג מסוים של מידע אלא המחשבים ברשת יוכלו לשלוח דרך הרשת כל סוג של מידע, מקובץ, ועד אודיו וידאו וכדומה.

פרוטוקול TCP/IP מורכב מהרבה פרוטוקולים, אבל נקרא על שם שני הפרוטוקולים המקוריים שלו, שהם IP ו-TCP.

הפרוטוקולים מחולקים ל-5 שכבות, כאשר כל שכבה נותנת שירות לשכבה מעליה, וכן מקבלת שירות מהשכבות שמתחתיה. וכן כל שכבה מקיימת דיאלוג עם השכבה המקבילה לה במחשב היעד.

נפרט בקצרה את המטרה של כל שכבה, וכן נמנה חלק מהפרוטוקולים בשכבה [20], [21]:

השכבה הראשונה נקראת The physical layer. מטרת השכבה לטפל בהעברת המידע הפיזי בחומרה בין המחשבים, לדוגמא, האם המידע יעבור בצורה דיגיטלית או אנלוגית וכן מה יהיה קצב העברת השידור וכדו'.

השכבה השנייה נקראת The data Link Layer. מטרת השכבה לטפל בשגיאות שקרו במהלך העברת המידע, כגון רעש שהיה במהלך השידור בכבל שגרם לביטים להתחלף, וכן בבקרת זרימה, שלא יהיו התנגשויות במהלך העברת המידע בכבל.

השכבה השלישית נקראת Internet Layer. מטרת השכבה למצא את הנתיב לשליחת המידע ממחשב המקור למחשב היעד.

נציין כמה פרוטוקולים בשכבה הזאת:

* IP Protocol: הפרוטוקול אחראי על כתובות המחשבים ודרך הגישה אליהם.

כתובת IP של מחשב הוא מספר זיהוי ייחודי ברשת, שניתן לכל מחשב המחובר לרשת. בגרסה IPv4, גודל הכתובת הוא 32 סיביות (4 בתים).

נהוג לכתוב את הכתובת לפי מספרים עשרוניים, כאשר כל בית נכתב בפני עצמו ומופרד בנקודה. לדוגמא: 250.3.122.111 (הטווח של כל אחד מהארבע ספרות הוא מספר בין 0 ל-255).

עקב הגידול המהיר במכשירים המחוברים לאינטרנט, נוצרת מצוקה של כתובות IP פנויות, ולכן הוגדר תקן חדש IPv6 שבו למרחב הכתובות הוקצה 128 סיביות שיכולה להחזיק 2128 כתובות.

* Address Resolution Protocol (ARP): הפרוטוקול מוצא את כתובת הMAC- של מחשב היעד ע"י כתובת הIP- של מחשב היעד.

כתובת MAC הוא מספר ייחודי שניתן לכל כרטיס רשת.

* Reverse Address Resolution Protocol (ARAP): הפרוטוקול מוצא את כתובת הIP של המחשב השולח.
* Neighbor Discovery Protocol (NDP): פרוטוקול לשימוש כאשר משתמשים בIPv6. הפרוטוקול נועד לאסוף מידע קריטי להעברת ההודעות.

השכבה הרביעית של פרוטוקולי TCP\IP נקראת Transport Layer. הפרוטוקולים המרכזיים בשכבה זו הם TCP וUDP-.

פרוטוקול TCP אחראי להקמת הקשר בין המחשבים, ומוודא ששני המחשבים מעוניינים בקשר. הפרוטוקול גם מוודא שחבילות שנשלחו הגיעו לייעדן. וכן הפרוטוקול אחראי על סידור ההודעות, כך שמחשב המקבל את ההודעות, יקבל אותם לפי הסדר שבהם הם נשלחו. (כל פאקט יכול לעבור דרך נתיב שונה, ולכן אין וודאות שההודעות מגיעות למחשב היעד באותו סדר שנשלחו).

פרוטוקול UDP בניגוד ל-TCP, לא מוודא שכל המידע הגיע לייעדו, ובפרוטוקול זה, אם ההודעה נאבדה בדרך, המחשב השולח לא ישלח שוב את ההודעה, והמחשב המקבל לא ידע שניסו לשלוח לו הודעה. פרוטוקול UDP נועד לאפשר העברת מידע רב בזמן קצר. ובשונה מ-TCP הפרוטוקול לא חושש מהתנגשויות ולכן לא מחכה שהקו יתפנה, ושולח מייד את ההודעות.

פרוטוקול חשוב נוסף בשכבה זו נקראת Transport Layer Security (TLS), פרוטוקול TLS אחראי על שמירת הפרטיות במהלך תקשורת של שרת לקוח.

השכבה החמישית נקראת Application Layer, תפקידה לקשר בין 2 המחשבים, וכן להמיר את המידע הנשלח לפורמט שהשכבה הרביעית יכולה לקבל. נמנה חלק קטן מהפרוטוקולים הרבים בשכבה זו:

* Domain Name Service (DNS): פרוטוקול להמרת כתובת אינטרנט בצורת דומיין, לדוגמא www.example.com, יומר לכתובת IP שמורכבת מ32 סיביות.
* File Transfer Protocol (FTP): פרוטוקול להעתקת קבצים בין מחשבים.
* Simple Mail Transfer Protocol (SMTP): פרוטוקול המאפשר שליחת הודעות אימייל בין מחשבים.
* HTTP ו-HTTPS: פרוטוקולים לשליחת דפי אינטרנט ברשת.

**איומי אבטחה והתגוננות**

אפשר לסווג בצורה כללית את צרכי האבטחה ברשת ל-4 קטגוריות [22]:

* סודיות: מניעת אינפורמציה סודית מאנשים לא מורשים.
* אימות זהויות(Authentication): וידוא שהאדם בצד השני של התקשורת שטוען שהוא X, אכן באמת X.
* אי-התכחשות(Nonrepudiation): אפשרות להוכיח שאדם שעשה פעולה ברשת אכן עשה אותה - לדוגמה: אם X ביקש מהבנק לעשות העברת כסף ל-Y, והבנק העביר את הכסף, אבל לאחר ביצוע העברה, X מכחיש שנתן הוראה לבנק להעביר את הכסף.
* אינטגריטי: וידוא שההודעה לא שונתה בדרך.

נציין כי רוב הפתרונות לאחד מהסיווגים, פותרים גם חלק משאר הסיווגים.

סודיות ע"י אלגוריתם RSA

כל המחשבים ברשת, מחוברים ביניהם דרך נתבים. כאשר מחשב X שולח הודעה למחשב Y, ההודעה עוברת ממחשב X דרך רשת של נתבים למחשב Y. ע"י פרצה לאחד מהנתבים, אפשר לקרא את ההודעה ששלח מחשב X למחשב Y[23].

לפתור בעיה זו, ולשמור על סודיות המידע המועבר, משתמשים בקריפטוגרפיה להסתרת המידע. קריפטוגרפיה היא מילה ביוונית שמשמעותה "כתיבה סודית", והיא עוסקת בהצפנת מידע ופיענוחו [24].

אלגוריתם RSA מאפשר תקשורת מוצפנת בין 2 ישויות גם כאשר לא היה ביניהם קשר מוקדם. האלגוריתם נחשב כחזק ביותר, ונבחר לשימוש להצפנת המידע ברשת, כך שיפתור את בעיית הסודיות.

האלגוריתם מהווה את הבסיס של אבטחת הרשת בימינו. ומפני השפעתו הרבה, קיבלו מפתחיו בשנת 2002 את פרס טיורינג היוקרתי[24].

האלגוריתם מבוסס על בעיה פתוחה בתורת המספרים, לגבי פירוק לגורמים של מספר פריק שהוא כפולה של שני מספרים ראשוניים גדולים.(האלגוריתם אמור להיות לא בטוח כאשר יהיו מחשבים קוואנטים, כי קיים אלגוריתם שפותר בעיה זו בזמן קצר במחשב קוואנטי).

נתאר את מנגנון האלגוריתם [23]:

1. נבחר באופן אקראי 2 מספרים ראשוניים גדולים p ו-q, ונקבע את n להיות- n=p\*q.
2. נבחר באופן אקראי מספר e, כך שהוא זר ל(q-1)\*(p-1).
3. נבחר מספר d מהצורה: (ed-1)/(p-1)(q-1)
4. המפתח הפומבי הוא: (n, e).

כאשר מחשב X מעוניין לתקשר עם מחשב Y, מחשב X רק צריך לבקש ממחשב Y את המפתח הפומבי שלו(n, e). המפתח אינו סודי ולכן מחשב Y יכול לשלוח את המפתח למחשב X בצורה לא מאובטחת.

כאשר מחשב X מקבל ממחשב Y את המפתח הפומבי, מחשב X מצפין את ההודעה M שמעוניין לשלוח, ע"י הנוסחה: C=Me Mod n, כאשר C מסמנת את ההודעה המוצפנת. .(אם ההודעה M גדולה מ-ח, צריך לחלק את ההודעה לבלוקים נפרדים, כאשר כל בלוק יהיה בגודל 2k , כאשר k הוא מספר הגדול ביותר המקיים 2k < n).

ההודעה המוצפנת ,C אינה ניתנת לפענוח ללא המפתח הפרטי של מחשב Y לכן ,מחשב X יכול לשלוח את ההודעה C למחשב Y ברשת, בצורה לא מאובטחת.

כאשר מחשב Y מקבל את ההודעה המוצפנת C, המחשב Y יפענח את ההודעה C ע"י המפתח הפרטי שלו(d), לפי הנוסחה: M=Cd Mod n.

החיסרון בשימוש באלגוריתם RSA, שהאלגוריתם נחשב כבטוח ואינו ניתן לפריצה רק כאשר שני המספרים הראשוניים p ו-q שנבחרו, הם לפחות בגודל של 1024 סיביות, מה שגורם לאלגוריתם ההצפנה והפיענוח להיות איטי.

נראה דוגמה הממחישה הצפנה ופיענוח בעזרת אלגוריתם RSA.

***דוגמא במספרים קטנים***

נניח שבוב רוצה לשלוח לאליס את המספר של הסיסמא של הבנק, והמספר הוא 123. אליס צריכה לייצר מפתח פרטי ולשמור אותה לעצמה, ומפתח ציבורי לשלוח לבוב בכדי שיוכל להצפין את הסיסמא.

בכדי לייצר מפתח פרטי וציבורי אליס צריכה קודם לבחור 2 מספרים ראשונים(בעיקרון המספרים צריכים להיות מאד גדולים), נניח שאליס בוחרת את המספרים הראשונים הבאים:

P=5581, q=8059, ולכן n=5581\*8509=44977279.

אליס יכולה לבחור גם את e=233 כי הוא זר ל-(p-1)(q-1).

לפי המספרים האלה d=192977.

וא"כ המפתח הפרטי של אליס הוא 192977d=, והמפתח הציבורי הוא (n=44977279, e = 233) .

בוב מקבל מאליס את המפתח הציבורי שלה ומצפין בה את ההודעה שלו ע"י הנוסחה הבאה:

C=M^e Mod n

ולכן:

C = 123^233 mod 44977279 = 32107344

בוב שולח לאליס דרך תקשורת גלויה את ההודעה המוצפנת C – 32107344.

אליס מפענחת את ההודעה ע"י הנוסחה: M = C^d mod n

ולכן אליס תקבל:

M = 32107344^192977 mod 44977279 = 123

אי-התכחשות – חתימה דיגיטלית

פרצה נוספת שיש ברשת, היא אי היכולת להוכיח פעולות שנעשו ברשת. לדוגמה: פלוני שלח למנהל החשבונות שלו הודעה למכור את המניות, מיד אח"כ מחירי המניות ההמירו, ופלוני טוען שהוא לא נתן את הוראת המכירה, איך מנהל ההשקעות יכול להוכיח שהוראת המכירה אכן ניתנה?

הפתרון לבעיה זו, היא שימוש בחתימה דיגיטלית. יש 3 אופנים ליישום חתימה דיגיטלית[22], כאשר נרחיב על 2 מהם:

* חתימה ע"י הצפנה סימטרית: קיים "אח גדול" שכל המחשבים ברשת מכירים וסומכים עליו, והאח הגדול שומר את המפתחות הפרטיים של כל המחשבים. כאשר X רוצה לחתום על מסמך ולהעבירו ל-Y, הוא כותב את המסמך, ומוסיף במסמך שהוא מעוניין להעביר את המסמך ל-Y, ואח"כ מצפין את המסמך עם המפתח הפרטי שלו ומעבירו לאח הגדול. האח הגדול מפענח את המסמך ע"י המפתח הפרטי של X שנמצא אצלו, ומעביר את המסמך המפוענח ביחד עם ההודעה המוצפנת ל-Y, כך יכול האח הגדול להוכיח שאכן המסמך חתום ע"י X , כי רק כך Y יכול להביא את ההצפנה והפיענוח.
* חתימה ע"י הצפנה אסימטרית: נראה את העיקרון ע"י שימוש באלגוריתם RSA.

כאשר X מעוניין לשלוח מסמך חתום ל-Y, X מצפין את החוזה ע"י שימוש במפתח הפרטי שלו, ואח"כ שוב מצפין את ההודעה ע"י מפתח הציבורי של-Y. Y מקבל את ההודעה, ומפענח אותה תחילה ע"י המפתח הפרטי שלו, שמבטל את ההצפנה השנייה של-X, וא"כ אחרי הפיענוח הראשון הוא מקבל חוזה מוצפן ע"י המפתח הפרטי של Y, שלפי האלגוריתם של RSA , ניתן לפענוח רק ע"י המפתח הציבורי של X, וא"כ פיענוח ההודעה ע"י המפתח הציבורי של X מוכיחה ש-X כתב אותה.

* חתימה ע"י שימוש בפונקציית גיבוב, כגון SHA-1.

אימות- Certificates

נניח ש-X מעוניין להרחיב את מעגל חבריו, והוא מעוניין לתקשר עם אדם Y,אבל אין לו מכירים שמכירים את Y, איך הוא יוכל ליצור עם Y קשר בצורה מאובטחת? מאיפה יהיה לו את המפתח הציבורי של Y? ואם משהו Z ייתן לו מפתח ציבורי ויומר לו שהוא של Y איך הוא יכול לוודא שהמפתח הציבורי אכן של Y, כי יתכן שזה המפתח הציבורי של Z, ו-Z מאזין לשיחות שלו ומעביר אותם הלאה ל-Y?

בשביל לפתור את הבעיות האלו, הוקמו אוטוריטות ידועות, שיודעות את המפתח הציבורי של כל אחד, ואפשר לפנות אליהם לוודא שייכות של מפתח ציבורי למשתמש. האוטוריטות מוודאות גם דברים נוספים, וכן נקבע סטנדרט אחיד לאישורים – X.509 .



פרוטוקולי SSL ו-TLS מיישמים את העקרונות שמנינו למעלה, ומאפשרים תקשורת סודית ע"י מבחר אלגוריתמי הצפנה, וכן מאפשרים אימות ע"י אלגוריתמי מפתח ציבורי, ואי-התכחשות ע"י חתימה דיגיטלית, וכן אינטגריטי ע"י מבחר פונקציות גיבוב.

**פרק 4: איומי אבטחה מערכתיים**

בפרק זה נתמקד בהגנה מפני מתקפות כלליות וכן בהגנה מפני מתקפת DDos הגורמת למניעת שרות.

**נוזקה**

נוזקה(Malware) היא מונח כללי, לתוכנת מחשב, אשר מבצעת את התכנון הזדוני של יוצרה [25].

נפרט חלק מהתוכנות הזדוניות הקיימות [25]:

* וירוס: תוכנה שרצה על מחשב, ללא קבלת רשות מבעל המחשב, הגורמת נזק למחשב.
* תולעת: וירוס המתפשט ברשת ומזיק מחשבים אחרים.
* סוס טרויאני: תוכנה מזיקה המתחזה לתוכנה תמימה שאינה מזיקה.
* רוגלה: תוכנה שעוקבת בחשאי אחר פעולות המשתמש במחשב, ומעבירה את המידע לכותב התוכנה.
* תוכנת כופר: תוכנה המגבילה גישה למערכות המחשב, ובתמורה להסרת ההגבלות, התוקף סוחט כסף מהמותקף.

חברת האבטחה Kaspersky פרסמה [26] שבשנת 2019 , 19.8% מהמשתמשים במחשבים היו מטרה ללפחות תקיפה אחת של נוזקת מחשב.

הדרך העיקרית להתגונן מנוזקה היא בשיטת "זיהוי חתימה"(Signature-based detection) [25]. בשיטה זו, קיים מאגר של נוזקות, ולכל נוזקה קיימת חתימה מיוחדת. כאשר מזהים תוכנה חדשה, סורקים את התוכנה, ובודקים האם יש בה חתימה של נוזקה הקיימת במאגר. חתימה, היא תבנית מיוחדת הייחודית לנוזקה המסוימת.

שיטת זיהוי החתימה מועילה רק לגלות נוזקות ידועות שבוצע להם חתימה, אבל נוזקה שלא בוצע לה חתימה, לא תתגלה בסריקה, ולכן שיטת זיהוי החתימה לא תגן מנוזקות חדשות(מתקפת אפס ימים) [25].

אנליזה סטטית היא שיטה למציאת התבנית(חתימה) של נוזקה חדשה שהתגלתה, ע"י ניתוח הקוד של הנוזקה בלבד ללא הרצת התוכנה.

קיימים מספר שיטות סטטיות לניתוח הקוד בכדי לאפיין את התבנית של הנוזקה. נפרט שתיים מהם [25]:

* File Format Inspection: ניתוח המטא-דאטה של הקובץ. לדוגמה: בקבצים של Windows PE אפשר למצא מידע על הפונקציות וכן מידע על זמן ההידור שיכול לסווג את הנוזקה.
* String Signature: סריקת כל המחרוזות בקוד. מציאת שמות קבצים מסוימים ו\או URL מסוים, יכול להראות על ייחודיות של נוזקה.

הבעיה בגישה זו היא שבכדי לבצע אנליזה על הקוד צריך קודם לפענח את קובץ ההרצה של הנוזקה, אבל טכניקה בשם Binary obfuscation מקשה מאד על פיענוח קובץ ההרצה, כך שיש קושי רב להגיע לשלב של האנליזה על הקוד. קושי זה גרם לפיתוח שיטת הזיהוי השנייה – אנליזה דינמית [25].

אנליזה דינמית משתמשת בשיטת "זיהוי מבוסס התנהגות" (Behavior-based analysis), בשיטה זו, זיהוי הנוזקה מתבצעת בזמן ריצת התוכנה, ובדיקה האם ההתנהגות תואמת להתנהגות של נוזקה ידועה. התנהגויות חשודות יכולות להיות שינוי בקובץ Host או שינוי בRegistry Key וכדו'.

התוכנה מורצת בסביבה מוגנת(כגון מכונה וירטואלית), כך שהמחשב לא יינזק מהנוזקה. ובשונה מאנליזה סטטית אין צורך בפיענוח קובץ ההרצה.

אחד מהחסרונות בשיטת האנליזה הדינמית היא השוני בין הסביבה הוירטואלית שבה מורצת התוכנה לבין הסביבה בעולם האמיתי, וכן בשוני בין סביבות עבודה שונות. לא בהכרח שתוכנה שנהגה בצורה מסוימת במחקר תנהג כך גם בעולם האמיתי או בסביבות עבודה שונות. בנוסף, פעמים רבות הנוזקה מופעלת ע"י פעולה ספציפית כגון פקודה ספציפית וכדו', מה שמקשה על גילוי הנוזקה בשיטה הזאת.

כפי שצוין, הבעיה הגדולה בשיטת זיהוי החתימה, היא בחוסר יכולתה לגלות נוזקות חדשות, ובנוסף, התוקפים פיתחו נוזקות חדשות המשנות תדיר את חתימתם והתנהגותם(Polymorphic Malware) כך שמקשים מאד על הגילוי בשיטה זה. בעיה נוספת היא ההסתמכות על ביצוע אנליזה מוקדמת לכל נוזקה חדשה שמתגלה, חברת SonicWall פרסמה, שבשנת 2019 גילתה 439,854 נוזקות חדשות [27], בקצב זה של גילוי נוזקות חדשות, אין אפשרות לבצע אנליזה ידנית לכל נוזקה חדשה.

עקב כך, פותחו שיטות חדשות שמבצעות אנליזה סטטית ודינמית על התוכנה בזמן אמת, באופן אוטומטי, ע"י שימוש בלמידת מכונה. בדרך זו אפשר לגלות גם נוזקות חדשות, וכן לגלות נוזקות שמשנים את חתימתם. בנוסף, הדבר פותר את הצורך באנליזה ידנית לכל נוזקה(עדיין קיים צורך באנליזה למחקר על סוגים שלא התגלו וכדו') [25].

לימוד מכונה מוגדרת כאוסף מתודות וטכנולוגיות המאפשרת למחשבים ללמוד בעצמם, ללא הצורך לתכנת אותם באופן ספציפי. המחשב יכול ללמוד ע"י "למידה מונחית" – למידה מתוך בסיס מידע המכיל פתרונות, או ע"י "למידה בלתי מונחית" – איפון של קטגורית של בסיס מידע המכיל מידע גולמי.

ב"למידה מונחית" קיימת למחשב 2 מצבים, מצב אימון ומצב קרב. במצב אימון המחשב מקבל מאגר מידע המכיל פתרונות, ובונה מודל ניבוי, שע"פ ינבא האם התוכנה מכילה קוד זדוני. במצב קרב, המחשב משתמש במודל לניבוי שבנה, בכדי להכריע האם התוכנה מכילה קוד זדוני.

בשנים הראשונות של המחקר על למידת מכונה, הדעה הרווחת הייתה, שלמידת מכונה תהווה פתרון מושלם לגילוי נוזקות, אבל היום התבססה ההבנה שלמידת מכונה היא רק חלק ממארג של טכניקות לגילוי נוזקה [28].

אחת מהבעיות המדוברות של פתרונות המשתמשים בלמידת מכונה, היא האפשרות לבצע מניפולציות על מחשבים המשתמשים בלימוד מכונה. לדוגמא, ע"י תליית תמונה על אדם, מחשב המשתמש בלמידת מכונה לזהות אנשים, לא יזהה את האדם. וכן ע"פ מחקרים שנעשו, הוספת מדבקה קטנה לשלט עצור, גורמת למחשב המשתמש בלמידת מכונה לזהות שלטים, לא לזהות את השלט עם המדבקה, וכן עוד דוגמאות רבות.

נציין בעיות נוספות שהתגלו בשימוש בלמידת מכונה לגילוי נוזקות: [28]

* Label poisoning: בשיטה זו, מחדירים למאגר המידע שבו המחשב מתאמן, מידע רעיל (מידע שיקרי), לדוגמא, נכניס נוזקה למאגר אבל נסווג אותה כתוכנה לא מזיקה, וכך המחשב בונה מודל ניבוי שגוי. שיטה זו כבר קרתה במציאות לאתר VirusTotal של גוגל.
* Training dataset poisoning: בשונה מהשיטה הקודמת שמחדירים מידע רעיל, בשיטה הזאת מחדירים מידע שהוא נכון, אבל כזה שיגרם לבניית מודל ניבוי שאינו מנבא נכון נוזקות. לדוגמא, נכניס למאגר קובץ שיש לו הרבה מהמאפיינים של קוד זדוני, אבל משמיטים את הקוד הזדוני. הכנסת מידע מסוג זה יגרום לכך שמודל הניבוי לא יסווג את המאפיינים האלה כמאפיינים זדוניים. שיטה זו מסוכנת במיוחד, הואיל והרבה מהמידע הגולמי הנלקח לאמן את המחשב, נלקח ממאגרים פתוחים באינטרנט.
* Attack on pre-trained and outsourced ML models: תוקף יכול להשיג את מודל הניבוי של המחשב, וכך ע"י ניסויי וטעיה למצא את נקודות התורפה של הקוד.

לסיכום: לא קיים אסטרטגיה אחת מושלמת להגנה מנוזקות, ויש להשתמש בשילוב של אסטרטגיות להגיע להגנה טובה.

**התקפת מניעת שירות מבוזרת(DDoS)**

תקיפה DDoS, מורכבת מתקיפות רבות של DoS המבוצעות ממחשבים שונים אשר מתואמים ביניהם בתקיפה. התקיפה ממקורות שונים מקשה מאד על הגילוי של המחשבים התוקפים, והופכת את הפסקת התקיפה לקשה מאד.

תקיפת DoS שונה מרוב נוזקות המחשב, בכך שאין מטרתה לפרוץ לרשת או למחשב, אלא מטרת התקיפה, למנוע מהשרת או המחשב לתת את השירות אשר הוא אמור לתת. לדוגמא, המטרה בתקיפת שרתי החיפוש של גוגל ב-DDoS, היא למנוע ממשתמשים המבצעים חיפוש באתר של גוגל מלקבל מענה לשאילתת החיפוש שלהם.

תקיפת DoS מתבצעת ע"י שליחת מידע רב לשרת כך שלא יוכל לטפל בבקשות אחרות.

ב-1999 התבצעה התקיפה הראשונה הידועה ב-DDoS, ובשנת 2000 ארע תקיפה שהפילה את אתרי האינטרנט של Amazon, eBay, CNN, Yahoo! למשך שעות רבות.

נציין כמה דרכים לגרום לכך שהמחשב יוצף כך שלא יוכל לתת שירות [29], [30]:

* ICMP Flood attack: חלק מהפרוטוקול של הודעת ICMP היא, שמקבל ההודעה צריך לשלוח הודעה חזרה לשולח המעדכנת את השולח שההודעה ששלח התקבלה. לכן ע"י שליחת הודעות רבות של ICMP ,השרת יהיה עסוק כל הזמן בעדכון השולח שההודעה התקבלה, מה שיגרום לכך שלא יוכל לתת שירות. התוקף יכול לזייף את כתובת השולח של הודעת ה-ICMP כך שהתוקף לא יתגלה.
* Amplification attack: קיימים שרתים רבים שמחזירים תשובה על משלוח שאלה. התוקף משתמש בשרתים אלו לשלוח מידע רב לנתקף ע"י ששולח לשרתים שאלה, כאשר התוקף מזייף את כתובת ה-IP של שולח השאלה ומכניס את כתובת המותקף, כך השרתים ישלחו למותקף מידע רב של תשובות לשאלות שלא שאל. תקיפה זו חמורה יותר מהתקיפה הראשונה, כי עומס המידע שהנתקף מקבל מגיע ממקורות מוסמכים ובטוחים.

**הגנה ממתקפת מניעת שירות מבוזרת**

נחלק את דרכי ההתגוננות ל-4 [29], [30]:

* מניעה התקיפה.
* גילוי התקיפה.
* תגובה לתקיפה.
* סבילות.

נפרט את אסטרטגית ההתגוננות בכל אחד מהדרכים:

מניעת התקיפה

* שימוש בפילטרים ברשת למניעת מעבר של הודעות עם כתובות IP מזויפות. Park and Lee הציעו Route based filtering שבודק את הנתיב של ההודעה, ולפי זה קובעת אם ה-IP מזויף.
* ניתוק שירותים שאינם בשימוש. לדוגמא, אם אין צורך באינטרנט או אין צורך בUDP echo-, אז יש לנתק את השירותים האלה.
* לבצע עדכוני אבטחה תקופים. חלק גדול מהתקיפות מנצלות פרצות אבטחה בכדי לבצע את התקיפה, ולכן עדכוני אבטחה שסותמים פרצות אלו יכולים למנוע חלק בלתי מבוטל מהתקיפות.

גילוי התקיפה

Anomaly detection: מציאת אנומליות שמרמזות על תקיפה. נציין כמה תוכנות שמשתמשות באסטרטגיה הזאת:

* : NOMAD תוכנה שמנתחת את הheader של חבילת IP.

NetRanger של סיסקו: תוכנה שמזהה דפוסים ידועים של פרצות אבטחה שמנוצלים לפריצות DDoS, וחוסמת מידע שיש בהם אחד מהדפוסים האלה.

תגובה לתקיפה

כשהתקיפה מתגלה, הנתקף צריך לחסום את האפשרות של קבלת מידע מהתוקף. החסימה מתבצעת ע"י עדכון המנהל האדמיניסטרטיבי של הנתב של התוקף, ועדכון של רשימת בקרת הגישה.

החלק המאתגר הוא מציאת כתובת ה-IP של התוקף. האתגר נובע מהארכיטקטורה של האינטרנט, שבה הנתבים לא שומרים מידע על החבילות העוברות דרכם.

נציין 2 אסטרטגיות שנחקרו למציאת ה-IP של התוקף:

* ICMP traceback: על כל N הודעות שעוברות בנתב (כאשר נבחר N שהוא מספר גדול), הנתב בוחר הודעה אחת, שבה הוא שולח ליעד ההודעה הודעת ICMP ובה פרטים על השולח. כאשר התוקף שולח הודעות רבות, הסיכוי שייבחר אחד מההודעות שלו הוא גדול, וכך ימצא ה-IP של התוקף.
* Link testing traceback: מבצעים תקיפה הפוכה למקור ההודעות. אם מגלים האטה בחומרת התקיפה, המשמעות היא שהמקור שהותקף הוא משמש להעברת הודעות מהתוקף ובגלל ההתקפה ההפוכה ההודעות האלו לא יכלו לעבור ולכן היה האטה בחומרת התקיפה.

סבילות

הגישה הזאת מכירה בעובדה שקשה למנוע באופן הרמטי תקיפות DDoS, ולכן מתמקדת בהקטנת חומרת הפגיעה ממתקיפה.

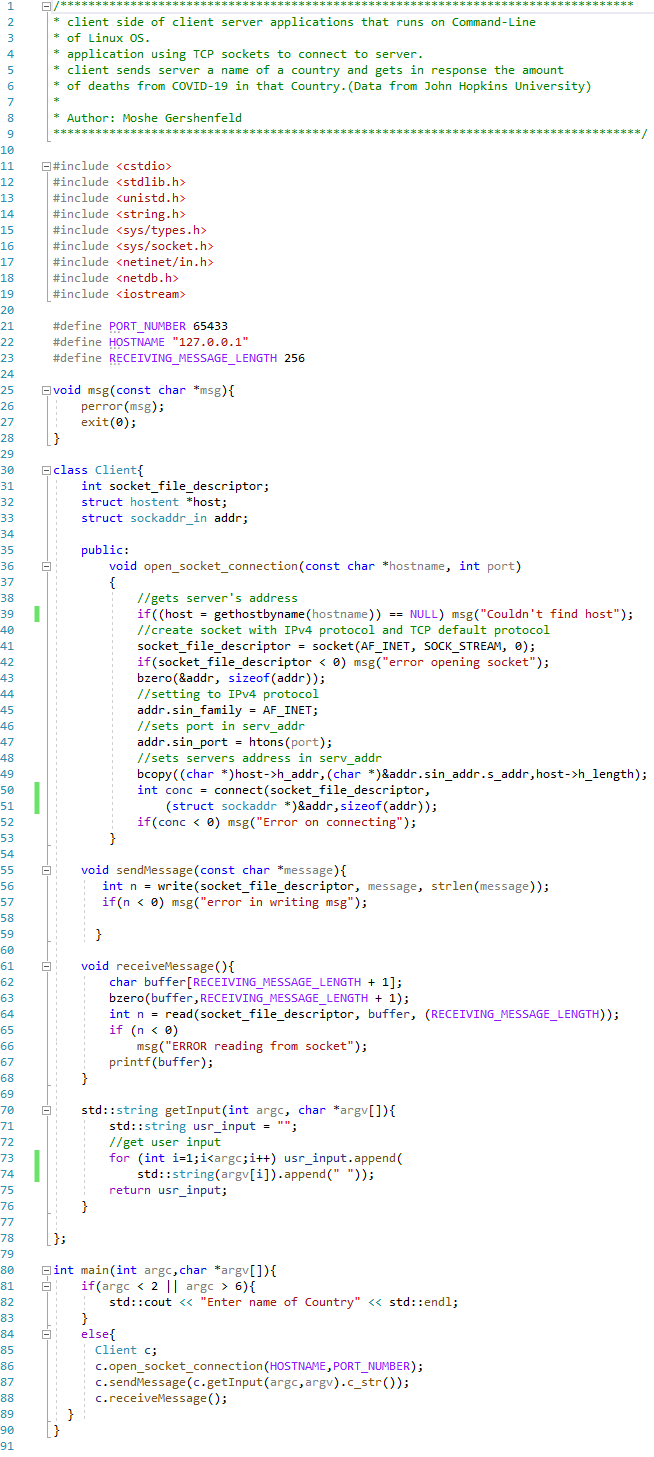
דרכים אפשריות להקטנת חומרת הפגיעה הם, שכפול שירותי הרשת וגיוון נקודת הגישה כך שגם אם השרת יותקף, השרת עדיין יוכל לספק ללקוחות את השירותים שלה.

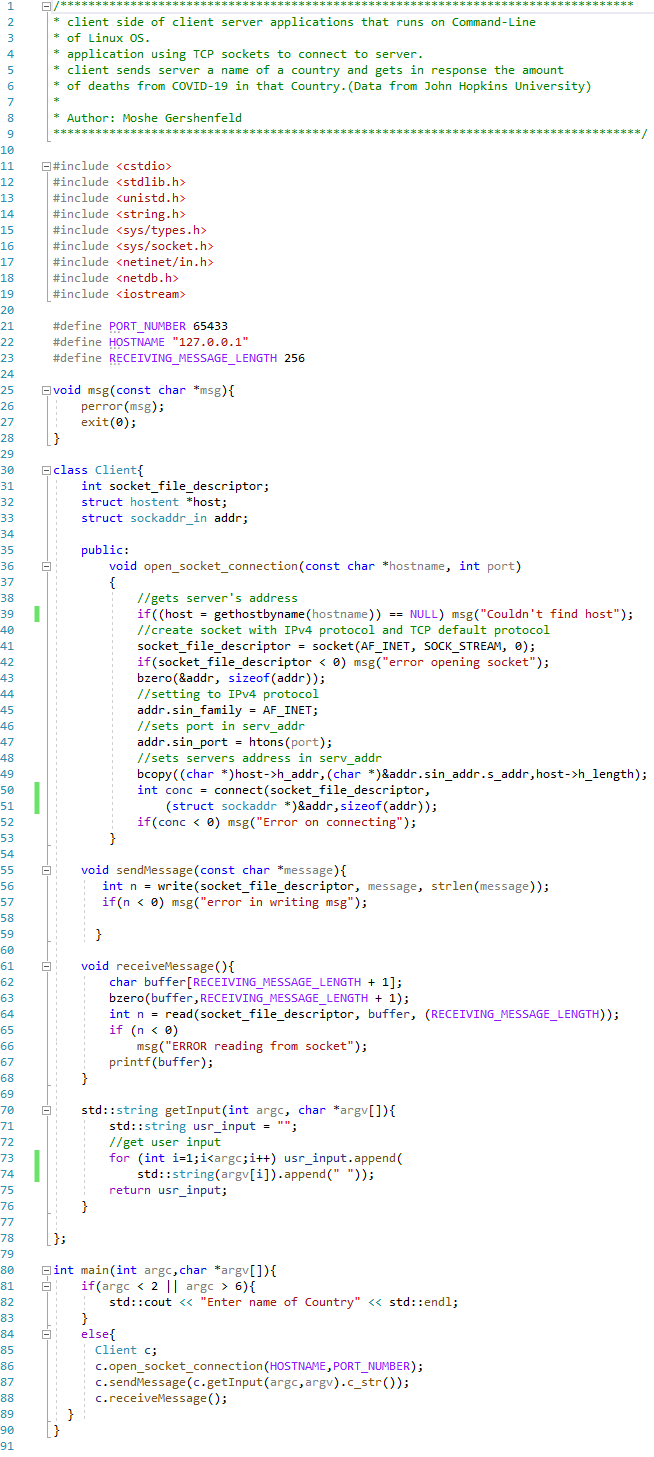
**פרק 5: מימוש מערכת שרת לקוח**

בפרק זה נראה מימוש של הגנות שנידונו בפרקים הקודמים, כאשר בתחילה נציג מערכת שרת לקוח אשר הצד לקוח ממומש ב-++C, וצד שרת בפייתון, כאשר אין הגנות לפרצות האבטחה שנידנו בפרקים הקודמים. בשלב שני נשתמש בחלק מהפתרונות שהבאנו להגן על המערכת מפרצות האבטחה.

המערכת שרת לקוח שישמש אותנו להדגים את ההגנות, היא מערכת שרת לקוח הרצה על ממשק שורת הפקודה של לינוקס, אשר המשתמש מריץ את התוכנה וכותב שם מדינה, ומקבל תשובה מהשרת, את כמות המתים שיש במדינה הזו מנגיף הקורונה. הנתונים לקוחים ממידע שמפרסם אוניברסיטת ג'ונס הופקינס.

**מימוש ללא הגנות**

צד לקוח



איומי אבטחה בצד לקוח ותיקונם

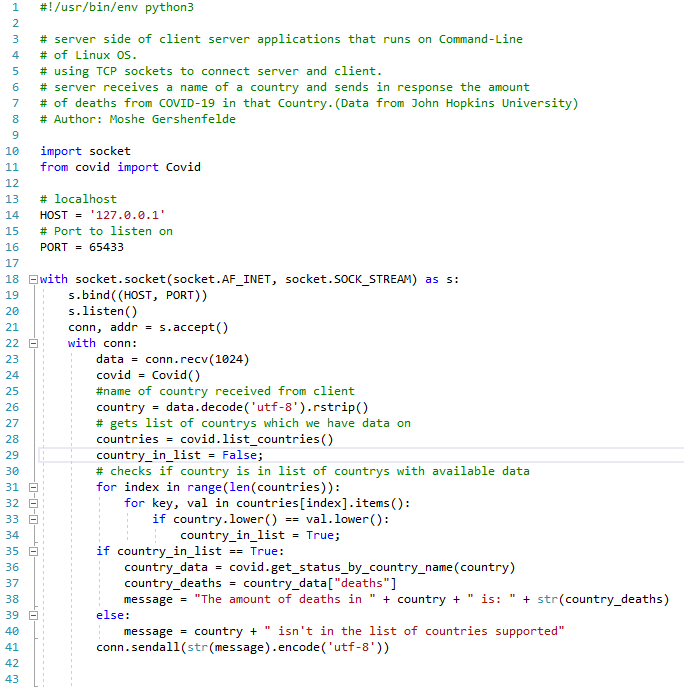
1. בשורה 67 קיימת פרצת האבטחה של Format String Attack, ע"י שימוש לא נכון בפונקציה printf.



לפתור בעיה זו נשתמש בפונקציה של C++ במקום printf: std:: cout<< buffer<< std::endl

1. בקוד יש שימוש בפונקציות שהוצאו משימוש, bcopy ו- bzero, במקומם נשתמש ב- memcpyו- memst בהתאמה.

נציין שלרוב מערכות הפעלה מותקנות תוכנות הממשות אסטרטגיות שפירטנו בפרק הראשון למניעת גלישת חוצץ. לדוגמא האסטרטגיות של Executable space protection ו- Address space layout randomization קיימות ברוב מערכות ההפעלה, כאשר בחלקם הם מאופשרים בברירת מחדל. [31], [32].

צד שרת

איומי אבטחה בצד שרת ותיקונם

צד השרת משתמש בחבילה covid שתלויה בחבילות pydantic ו- requests, שתלויות בחבילות רבות נוספות.

כפי שפרטנו בפרק 2, קיימת בעיית אבטחה כאשר מתגלה באג באחד החבילות התלויות ובוצע לחבילה עדכון, החבילה לא מעודכנת אצל מי שכבר הוריד ומשתמש בחבילה, ולכן פורץ יתקוף ע"י ניצול הבאג הידוע את כל החבילות שתלויות בחבילה עם הבאג.

נפתור את הבעיה ע"י שימוש ב-GitHub לאחסון הקוד.

GitHub סורק באופן תמידי את הקוד המאוחסן בה למציאת תלויות עם בעיות אבטחה. כאשר נמצאה אחת כזו, היא שולחת הודעה לבעל החשבון עם פרטים על בעיית האבטחה [33].

הסריקה מתבצעת בברירת מחדל בחשבונות המוגדרים public, ובחשבונות המוגדרים private יש צורך להצטרף לשרות לקבל את ההודעות.

ב-GitHub קיים גם שרות של Dependabot המעדכן באופן אוטומטי את החבילות לגרסה המעודכנת שלהם [34].

איומי אבטחה בתקשורת ואבטחתה

באפליקציה שבנינו המחזירה את מספר המתים במחלת הקורונה, אין צורך שהמידע המועבר יהיה מוצפן, כי אין במידע המועבר מידע סודי והמידע גלוי לכל, אבל קיימים 2 פרצות אבטחה אחרות, הראשונה היא "אימות", וידוא שהשרת שעונה על הבקשה הוא השרת שאליו פנינו ולא שרת מתחזה, והבעיה השנייה היא וידוא שההודעה לא שונתה בדרך.

לטפל בבעיות האלו נשתמש בספריית openssl בגרסת 1.0.2 שמותקנת במערכות הפעלה של לינוקס, לביצוע ההתקשרות בפרוטוקול TLS בגרסה 1.2.

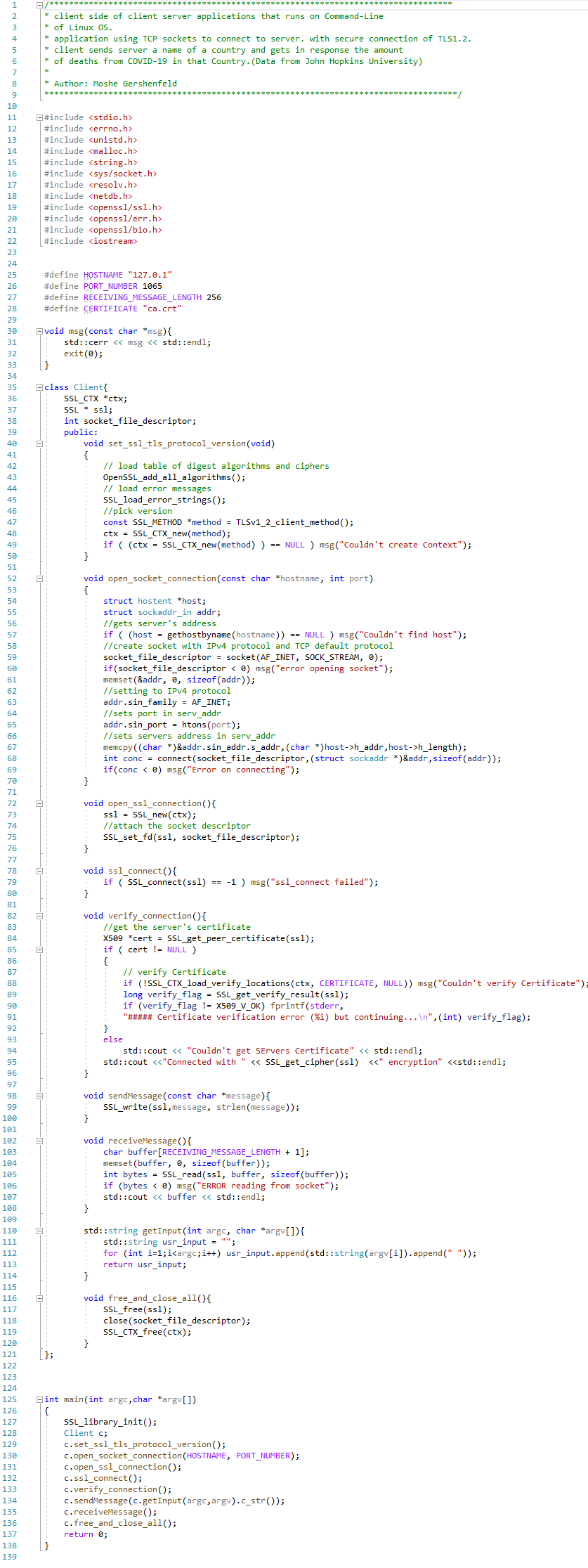
פרוטוקול TLS מבטיח תקשורת סודית ע"י הצפנה סימטרית, וכן אימות ע"י המפתחות הציבוריים והפרטיים, ובנוסף הפרוטוקול גם מבטיח אינטגריטי ע"י קוד אימות מסרים שמוודא שהמידע לא שונה

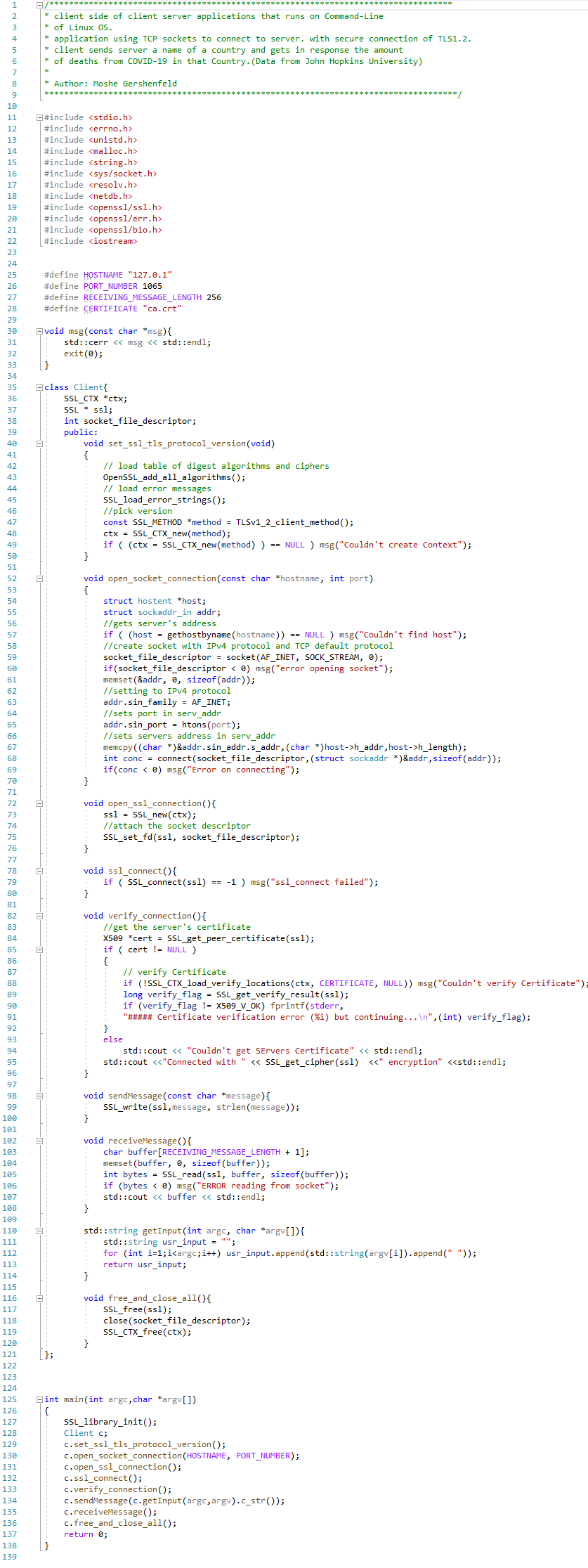
איומי אבטחה מערכתיים ותיקונם

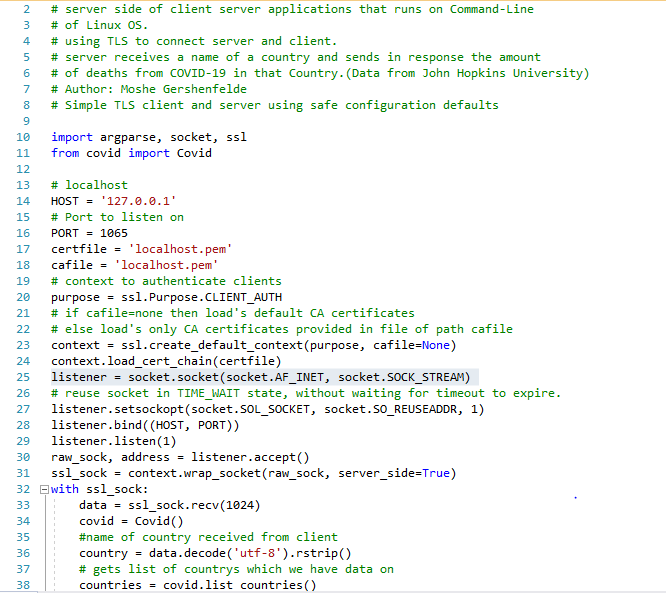
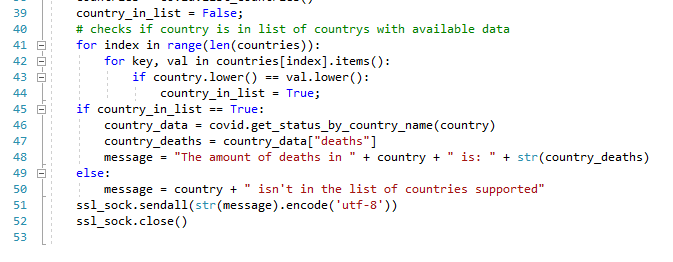
הגנות מנוזקות ווירוסים נעשות ע"י שימוש בתוכנות של חברות המומחות בתחום. כגון Avast Antivirus שמגנה מנוזקות, ו-Cloudflare Spectrum שמגנה ממתקפות DDos.

נציין שרוב חברות שרותי הענן ושרותי האחסון גם מספקות שירותי הגנה מ-DDos.

**מימוש עם הגנות**

צד לקוח



צד שרת

**סיכום**

היראתי בקצרה במהלך עבודה זו מספר פרצות אבטחה ידועות במערכת שרת לקוח, ואת הדרכים להימנע מהם.

במיוחד היראתי שרבות מהפרצות האבטחה נפתרות בקלות ע"י קוד שסותם פרצות אלו, ע"י שימוש בספריות מיוחדות או פרוטוקולים ידועים.

ישנם פרצות נוספות שנפוצות מאד ולא הרחבנו עליהם, כגון הזרקת קוד(SQL injection וכדומה), אבל מה שחשוב מאד לקחת מחיבור זה, הוא העיקרון שמתכנת אמור להכיר את בעיות האבטחה הנפוצות שניתנות לפתרון ע"י קוד, וכבר במהלך הכתיבה הראשונית של הקוד להתייחס אליהם ולנסות לפתור אותם.

במהלך המחקר על כתיבת עבודה זו, שמתי לב לפרדוקס מעניין שמחזק את הנקודה הזאת. מצד אחד תקיפות הסייבר מתרבות ומשתכללות בקצב הולך וגדל, אבל מצד שני תקיפות הסייבר מנצלות בעיקר את אותם פרצות אבטחה הידועות לנו, ולא מנצלות פרצות חדשות.

לכן כתיבת קוד המתחשבת בפרצות האבטחה הידועות יכולה לפתור הרבה מבעיות האבטחה, ואין צורך שהמתכנת יהיה איש אבטחה בשביל להכיר בעיות אלו.

**References**

1. W. Hicks, "Software Security is a Programming Language Issue", *The Programming Languages Enthusiast Blog*. 2018 [Online]. Available: <http://www.pl-enthusiast.net/2018/08/13/security-programming-languages-issue> . [Accessed: Dec 2020].
2. B. Stroustrup, *THE C++ PROGRAMING LANGUAGE*, New Jersey. Addison Wesley Longman, 1997.
3. "C++", *in Wikipedia, the Free Encyclopedia,* [Online]. Available: <https://en.wikipedia.org/wiki/C%2B%2B>. [Accessed: Dec 2020].
4. B. Eckel, *Thinking in C++*, New Jersey, Prentice Hall Inc, 2000.
5. 'Buffer Overflow', *in Wikipedia, the Free Encyclopedia,* [Online]. Available: <https://en.wikipedia.org/wiki/Buffer_overflow> [Accessed: Dec 2020].
6. Y. Seema, A. Khaleel, and S. Jayant, "Classification and Prevention Techniques of Buffer Overflow Attacks", in *5th National Conference INDIACom-2011 Computing for Nation Development, BharatiVidyapeeth’s Institute of Computer Applications and Management, New Delhi, March 10 – 11, 2011*.
7. P. Lacroix and J. Desharnais, "Buffer Overflow Vulnerabilities in C and C++", *Laval University*, 2008.
8. "Uncontrolled format string", *in Wikipedia, the Free Encyclopedia**,* [Online]. Available: <https://en.wikipedia.org/wiki/Uncontrolled_format_string>. [Accessed: Dec 2020].
9. "Use-After-Free", in *Encyclopedia by Kaspersky*, [Online]. Available: <https://encyclopedia.kaspersky.com/glossary/use-after-free>. [Accessed: Dec 2020].
10. "CWE-416: Use After Free", in **CWE™**, [Online]. Available:

<https://cwe.mitre.org/data/definitions/416.html>.

[Accessed: Dec 2020].

1. "Smart Pointer", *in Wikipedia, the Free Encyclopedia,* [Online]. Available: <https://en.wikipedia.org/wiki/Smart_pointer> .

[Accessed: Dec 2020].

1. "Python (programming Language)", *in Wikipedia, the Free Encyclopedia,* [Online]. Available: <https://en.wikipedia.org/wiki/Python_(programming_language)> [Accessed: Dec 2020].
2. S. Pramanick, "History of Python", in *GeeksForGeeks*, [Online]. Available: <https://www.geeksforgeeks.org/history-of-python/> . [Accessed: Dec 2020].
3. "The Python Tutorial", in Python.org, [Online]. Available: <https://docs.python.org/3/tutorial/> . [Accessed: Dec 2020].
4. D. Beazley and B. Jonrs, *Python Cookbook*, California, O'Reilly Media Inc, 2013.
5. A. Scopatz and K. Huff, *Effective Computation In Physics*, California, O'Reilly Media Inc, 2015.
6. R. Vaidya, L. Carli, D. Davidson, and V. Rastogi, "Security issues in language-based software ecosystems". in *arXiv* [Online]. Available: <https://arxiv.org/pdf/1903.02613.pdf> . [Accessed: Dec 2020].
7. S. Georgian, "Common Python Security Problems", in *itnext.io - platform for software developers*, [Online]. Available: <https://itnext.io/common-python-security-problems-ffedbae7b11c> . [Accessed: Dec 2020].
8. "Timing Attacks against String Comparison", in *Developers Security Best Practices: best practices for avoiding vulnerabilities*, [Online]. Available: <https://sqreen.github.io/DevelopersSecurityBestPractices/timing-attack/python> . [Accessed: Dec 2020].
9. C. Leiden and M. Wilensky, *TCP/IP for Dummies*, New Jersey, Wiley Publishing, Inc, 2009.
10. P. Loshin, *TCP/IP clearly explained*, California, Morgan Kaufman Publishers, 2003.
11. A. Tenenbaum and A. Wetherall, *Computer Networks*, Pearson Education Inc, 2011.
12. Y. Goshwe, "Data Encryption and Decryption Using RSA Algorithm in a Network Environment", in *IJCSNS International Journal of computer Science and Network Security, Vol. 13 No. 7 pp. 9-13*, 2013.
13. "RSA", *in Wikipedia, the Free Encyclopedia,* [Online]. Available: <https://he.wikipedia.org/wiki/RSA>. [Accessed: Dec 2020].
14. E. Gandotra, D. Bansal and S. Sofat, "Malware Analysis and Classification: A Survey", in *Journal of Information Security, April 2014, pp. 56 -64*.
15. *Kaspersky Security Bulletin 2019: Statistics.*
16. *SonicWall Cyber Threat Report 2020.*
17. A. Chistyakov and A. Andreev, "AI under Attack: How to secure Machine Learning in Security Systems", *Kaspersky Threat Research*, 2019.
18. C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: a classification," *Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology* (IEEE Cat. No.03EX795), Darmstadt, Germany, pp. 190-193, 2003.
19. C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art", *Computer Networks, Volume 44, Issue 5, PP. 643-666*, 2004.
20. "Executable space protection", *in Wikipedia, the Free Encyclopedia,* [Online]. Available: <https://en.wikipedia.org/wiki/Executable_space_protection> . [Accessed: Dec 2020].
21. "Address space layout randomization", *in Wikipedia, the Free Encyclopedia,* [Online]. Available: <https://en.wikipedia.org/wiki/Address_space_layout_randomization>.[Accessed: Dec 2020].
22. R. Schultheis, "Security vulnerability alerts for Python", in *GitHub blog*. [Online]. Available: <https://github.blog/2018-07-12-security-vulnerability-alerts-for-python/> .[Accessed: Dec 2020].
23. P. Kalkman, "How to keep your dependencies secure and up to date", *in medium Blog*,[Online]. Available: <https://medium.com/better-programming/how-to-keep-your-dependencies-secure-and-up-to-date-92578c7f3c9c> . [Accessed: Dec 2020].