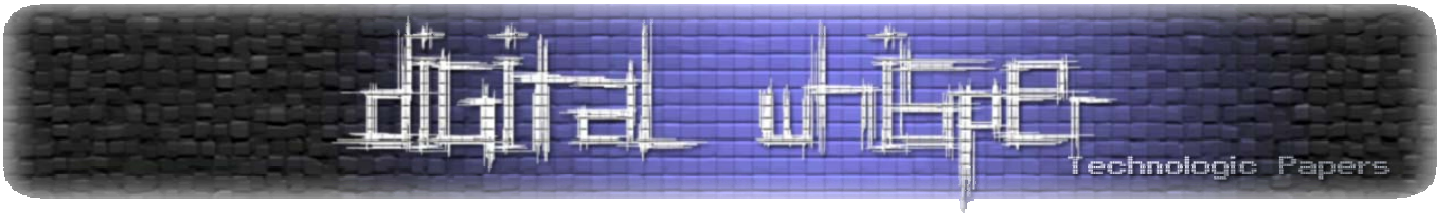


תכנות בטוח

מאת עידו קנר

הקדמה

מאמר זה הוא חלק ראשון המנסה ללמד גישה בפיתוח תכנות השונה מהצורה המקובלת במרבית המקומות. במאמר ננסה להשתמש בדוגמאות ובהסברים הפשוטים ביותר על מנת להסביר מהו "תכנות בטוח" ומה הם הצעדים הנדרשים על מנת לממש זאת. חלק זה מתייחס באופן כללי לבעיות בתכנות ובחלק הבא נדבר על הדרכים להתמודד עם אותן הבעיות.

במאמר נשתמש בשפת פסקל על מנת להציג את הפשטות, מצד אחד, ומצד שני עדיין להראות איך נוצרות בעיות, דבר שרק שפה זו מסוגלת לעשות זאת טוב וכמו שצריך לדעתי האישית.

מאמר זה מספק רק הצצה מקדימה על הנושא, אך חשוב להבין כי הנושא "תכנות בטוח" דורש הרבה יותר מאשר מאמר שכזה ולכן מאמר זה אינו נחשב לשלם. עוד חשוב להדגיש כי במידה ואתם מחפשים מידע איך להזיק, "לפרוץ" ולחבל במערכות מבוססות מחשבים, מאמר זה אינו מסייע בכך אלא על מסביר איך ניתן להתגונן מפני התקפות שכאלו כאשר כותבים תוכנה.

הבנה כללית

כאשר כותבים תכנה, סביר להניח כי היא תגיב לפעילות של משתמשים בצורה זו או אחרת, גם אם זה רק אומר שהתכנה משתמשת במידע קיים שהוזן על ידם בשלב כלשהו או סתם יש למשתמשים אפשרות לשלוט במידע.

בדרך כלל כאשר לומדים להשתמש בשפת תכנות בבתי הספר ובאוניברסיטאות, הדברים הראשונים הנלמדים הם כיצד לקבל קלט מהמשתמש ולהדפיס לו פלט בהתאם. בשלבים אלו המורים והמרצים בדרך כלל מוסיפים משפט כדוגמת "והניחו שהמידע תקין"- כאן מתחילות הבעיות בעצם.

מהשנייה הראשונה בה התכנה מתחילה לקבל קלט מהמשתמש, כבר אז לא ניתן לדעת האם המידע אשר התקבל הוא תקין או לא. לא ניתן לדעת אם המידע תקין בגלל שאין לכותבי התכנה אפשרות לשלוט במידע המגיע לתכנה עצמה. שחשוב להבין כי לקרוא מידע מקובץ זה לקרוא מידע לא אמין כך גם בקבלת מידע בחבילות (packets) באינטרנט.

למה לא ניתן לסמוך על מידע?

על מנת להבין מדוע מידע הוא דבר מאוד מסוכן יש צורך להבין בראש ובראשונה מה זה בעצם מידע. מידע יכול להיות הקשת מקש תזוזת עכבר וכך גם לחיצה על אחד מכפתורי העכבר או כמה מכפתורי העכבר. קלט יכול להיות עוד הרבה דברים אחרים כדוגמת קריאה של נתונים מקובץ או מהאינטרנט, קבלת מידע מפונקציית מערכת.

חשוב להבין שאין זה משנה מה הוא סוג המידע שאנו מעוניינים לקבל היות והמשתמש יכול לספק מידע שגוי. הסיבות למידע שגוי הן רבות ומגוונות: טעויות הקלדה/הדבקה וכוונות זדון הן הסיבות העיקריות לכך, לתקלות ברשת, חומרה או תוכנה אחרת. חשוב להבין כי אין שום דרך או צורה לדעת מה המידע שינתן לתוכנה מראש ולכן לא ניתן לסמוך על מידע המגיע אל תכנה.

המידע שהתכנה יכולה לקבל הוא רב ומגוון, מידע כזה יכול להיות למשל "מידע" ריק (null), טווח ספרות גבוה מהטווח המצופה, מחרוזת עם תווים רבים יותר מהמצופה או כתובות זיכרון אותן מנסים לשנות על מנת להריץ קוד זדוני כחלק מהקלט (תלוי בסוג ההתקפה שרוצים לבצע), כך שפשוט אי אפשר לדעת מה הקלט שהתכנה תקבל ולכן לא ניתן כך סתם לסמוך על המידע. כאשר יש טיפול "לא בטוח" במידע שהתקבל מהקלט, התוצאות יכולות להיות חשיפת מידע שלא ניתן ואסור לספק בשום דרך רגילה, שינוי מידע שאין כל דרך רגילה לבצע אותו או סתם לגרום לתכנה לקרוס.

מהן סוגי הבעיות שניתן לצפות ?

ניתן ליצור כלל אצבע האומר כי על כל סוג של באג בתוכנה ניתן למצוא התקפה מסוימת, אך במאמר זה נספק רשימה מפורטת אך קצרה מאוד של התקפות נפוצות ובעיות הקשורות לאבטחת מידע בתכנות, במקום להזכיר סוגים רבים בקצרה.

Buffer Overflow:

המושג Buffer Overflow מתייחס לחוצץ אשר חוצה את נפח המידע שהוקצה לו מבעוד מועד. חריגה זו מאפשרת לתוקפים לצאת מגבולות החוצץ וכך לשכתב מידע חשוב להמשך ריצת התוכנית. במקרים רבים, ניצול חולשה זו מאפשר הרצת קוד המוזרק על ידי התוקף:

```
var
  iNums : array [0..9] of integer;
  ...
  FillChar (iNums[-1], 100, #0);
```

```
...
for i := -10 to 10 do
   readln (iNums[i]);
...
```

בדוגמה זו ניתן לראות מערך סטטי בעל 10 איברים בשם iNums. בשימוש בפרוצדורת FillChar הוזן לתא לא קיים (מינוס אחד) ועד ל100 התאים הבאים ערך "ריק" (null). מתחת לפרוצדורה, ניתן למצוא לולאת for אשר מזינה 21 ערכים מהשתמש על מערך בגודל 10 תאים בלבד. חשוב להבין כי בדוגמה הזו המהדר יצעק על גלישות, אך בדוגמאות שהן פחות ברורות המהדר אינו יוכל לנחש או לדעת מראש מה יקרה ולכן לא תתקבל שום הודעת שגיאה או התראה על בעיה ומשתמשים יוכלו לנצל זאת להתקפה או לגרום לתכנה לקרוס.

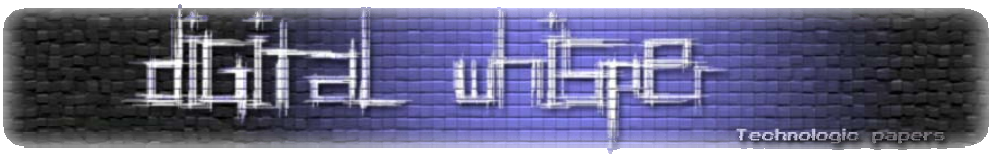
אם משתמש ירצה להזין מידע אשר ירצה להזיק לקוד, הוא יוכל להשתמש בדוגמה למעלה לצורך העניין, היות ויש חריגה מגבולות החוצץ שהוקצה עבור iNums. בעיה זו מוכרת בשם Buffer Overflow, אשר יכולה להתקיים עבור ערימות ומחסניות (stack ו heap), וההתקפות מתאימות את עצמן בהתאם לסוג הזיכרון.

מעבר לכך, במידה וניתן לגרום לבעיה זו לצוץ במערכת הרצה ברשת, פעמים רבות נראה כי נעשה שימוש בגלישת החוצץ על מנת לקבל גישה למחשב בהרשאות בה רצה התכנה וכך תוקפים יכולים לקבל גישה מרחוק אל המחשב המריץ את הקוד הבעייתי. התוקפים עושים זאת על ידי יציאה מחוץ למסגרות הזיכרון שהוקצו ודריסת המידע הקיים בחוצצים המקבילים, דבר זה מאפשר לתוקף לגרום לשנות את מהלך התוכנית ולהריץ קוד חדש, כאמור לפי ההרשאות שיש לתוכנה הרצה שבה התגלתה הבעיה. הרצת קוד זדוני מתבצעת בעיקר על ידי הרצת shell code שהוא בעצם קוד שהודר לשפת מכונה ומבצע דבר מה.

חשוב לדעת שבמערכות הפעלה חדישות כדוגמת Linux והחל מ- Windows XP SP2 ישנן הגנות שונות אשר מנסות להפריע להרצת קוד בצורה חופשית בזיכרון על ידי שינוי כתובות זיכרון בכל הקצאה, בניגוד לכתובת קבועה אשר מתקבלת בדרך כלל. אף על פי כן, צריך להבין כי אין להסתמך על הגנות אלו ויש לכתוב קוד טוב אשר לא יזדקק להגנות שכאלו, אשר מסייעות בצורה חלקית בלבד למנוע התקפות.

התקפת DoS:

פירוש DoS הוא Denial Of Service או מניעת שירות בעברית. אנשים רבים ודאי זוכרים כי בשנות ה-90 ניתן היה להשתמש בהרצת פקודת ה- ping פעמים רבות על מנת להפריע למחשבים, וחושבים שזו הבעיה היחידה שיש בנושא אך אין זה נכון. התקפת ה- ping נקראת ping of death וכיום, בשל גידול ברוחב פס הגלישה, מתקפה זו אינה מהווה סיכון.
ישנם כמה סוגים של התקפות מניעת, שירות אחת היא מקומית על המחשב והשנייה מתבצעת בצורה מרוחקת. ההתקפה המרוחקת, במידה והיא מגיעה ממקורות רבים, תקרא DDoS ונרחיב עליה יותר בהמשך.



מניעת שירות מקומית:
התקפות מניעת שירות מקומיות יכולות להתהוות בדרכים רבות, לדוגמא:

```
procedure Recurse;
begin
  while (True) do
    begin
      Recurse;
    end;
end;
```

הדוגמא מציגה רקורסיה ללא תנאי עצירה אשר בצורה אין סופית יוצרת עוד רקורסיה עד "אין סוף", כאשר ה"אין הסוף" הזה הוא המשאבים של מערכת ההפעלה הפנויים במערכת. למרות שזו דוגמה סטטית, עדיין קיימת מניעת שירות לכל דבר בשל "גניבת" כל זיכרון אפשרי ממערכת ההפעלה, וכן מניעת שירות על ידי חסימת עבודה סדירה עם המחשב.

בלינוקס למשל, מערכת ההפעלה תנסה להקצות יותר ויותר זיכרון וכאשר זה לא יהיה זמין יותר, מערכת ההפעלה "תהרוג" תוכנות בכדי לרצות את בקשת התכנה שיצרה את הרקורסיה עד אשר לא יישאר מה להרוג, אך במקביל תעניש את התכנה בכך שתתן לה קדימות פחותה בריצה, כך שיהיה קל יותר להרוג את התכנה הרצה.

ב-Microsoft Windows לעומת זאת, הזיכרון לעולם לא ישוחרר וגם לא יתקבל עונש כלשהו על הבקשה. למעשה, עד לאתחול מערכת ההפעלה, הזיכרון יישאר תפוס גם אם נצליח לחסל את ריצת התוכנה (וזה לא יהיה פשוט). מעבר לכך, כנראה שמערכת ההפעלה תתקע לגמרי ורק אתחול כפוי של המחשב יגרום לו לחזור ולהגיב.

דוגמא נוספת לחוסר שחרור משאבים:

```
...
begin
  while (True) do
    begin
      Getmem (OurPtr, 10);
      OurPtr := Something;
    end;
end.
```

דוגמא זו מציגה קוד המקצה זיכרון (הפקודה GetMem זהה לפקודה malloc של C) למשתנה בשם OutPtr אך הזיכרון לעולם לא ישוחרר, כתובת הזיכרון שאותחל אינה נשמרת באתחול הזיכרון הבא. בעיה זו בקוד סטטי, נקראת זליגת זיכרון (memory leak) ומאוד נפוצה בתכנות, בייחוד בשפות המאפשרות זאת. פעמים רבות מדובר "רק" בסוג של באג ולא התקפה מכוונת, אך עדיין מדובר בסוג של מניעת שירות.

חשוב להבין כי מחסור בשחרור משאבי מערכת כדוגמת זיכרון, שקעים (socket), קבצים פתוחים - כולם נחשבים לסוג של מניעת שירות מקומית, אך הם אינם היחידים. כמו כן, בעוד שמרבית מערכות ההפעלה ישחררו את הזיכרון חזרה אל המערכת כאשר התכנה תסיים את הריצה, מערכת ההפעלה Microsoft Windows לא עושה זאת (עד כמה שידוע לי) ורק אתחול ישחרר חזרה את הזיכרון במקרה זה.

מניעת שירות מבוזרת:

סוג נוסף ונפוץ למדי של מניעת שירות הוא מניעת שירות מבוזרת (DDoS – Distributed Denial of Service). מניעת שירות שכזו גורמת לנקודות שונות ורבות לבצע בקשה אחת או יותר כלפי שירות מסויים ברשת כלשהי והיא בדרך כלל מתבצעת באמצעות מחשבים רבים בשליטתו של מפעיל בודד, כאשר מחשבים אלו נקראים zombies. התקפה זו מצליחה ברוב המקרים משום שיש הרבה מאוד בקשות בו זמנית והשירות לא מסוגל לענות לכל הבקשות. במקרה הטוב השירות רק נחסם לעוד בקשות ובמקרה הרע גורם למערכת לקרוס מחוסר במשאבים פנויים להתמודד עם הבקשות השונות, גם לאחר שהמתקפה מסתיימת.

במידה והתקפה זו (או כל התקפת DoS אחרת) מתבצעת, לא ניתן לדעת או לנחש מה תהיה תגובת התכנות או המחשב אל מול ההתקפה ולכן קשה מאוד להיערך אליה מראש. עם זה, חשוב לדעת כי מערכות הבנויות על עבודה מבוזרת יצליחו במקרים רבים להתמודד עם התקפות DDoS טוב יותר ממערכות אחרות.

Code Injection:

כאשר המשתמש מזין קלט לתוכנה היא צריכה לעבוד עם הקלט ולהשתמש בו לצרכיה ומטרותיה. הבעיה היא שכאשר משתמשים במידע בצורה המדויקת בה הוא התקבל, ללא יצירת מסננים או טיפול במידע בעייתי, המשתמש יכול להזין קוד כלשהו (בין אם זהSQL או כל קוד אחר), להריץ אותו דרך התכנה ולעשות כל העולה ברוחו באמצעות הקוד שהוא הזריק. 2 דוגמאות מאוד נפוצות בעולם ה- web הן הזרקת SQL והזרקת HTML/Javascript (המוכר בשם Cross Site Scripting או XSS בקיצור), אך ישנם עוד סוגים רבים ונוספים של הזרקות קוד שונות.

דוגמא להזרקת SQL :

User Input:

Please enter your name: a' OR 1=1

```
...
write ('Please enter your name: ');
readln (sName);
Query1.SQL.Add ('SELECT Password FROM tblUsers WHERE Name='#32 +
sName + #32);
...
```

דוגמא להזרקת HTML:

url: http://example.com/?paramA=a”><script>alert('bla');</script>

```
...
<input type=”text” name=”paramA” value=”<%template
write(var['paramA']); %>” />
...
```

התוצאה של 2 הדוגמאות למעלה הן של הזרקת קוד אל משתנה כאשר במקום לאחסן את התוכן ← התוכן הוא בעצם קוד אשר מורץ על ידי מפרש כלשהו. ריצת הקוד במקום אכסון תוכן היא אינה התוצאה הרצויה, מן הסתם, ולכן זו בעיה.

Format String:

Format String היא תת-מחלקה בעלת גישה לכלל משתני הפונקציה המאפשרת לציין חוקים הקובעים כיצד ניתן להתייחס, להציג ולחבר משתנים יחד עם משתנים נוספים או את קביעת צורתם בתבניות השונות בהתאם לצורך.

הבעיה היא במידה ועושים שימוש לא נכון בשפה זו, הדבר מאפשר הזרקת קוד וגם בדרך כלל הפעלה של בעיית גלישת החוצץ על מנת לנצל לרעה את המחרוזת ולהריץ קוד זדוני במערכת, או ביצוע מניעת שירות על ידי גרימה לקריסת התכנה.

שפות C ו ++C הן בדרך כלל הפגיעות ביותר להתקפה מסוג זה, היות וכמעט כל קוד הנכתב בהן משתמש במצביעים וכך גם בנושא של Format String. גם שפות או כלים המבוססים על שפות אלו הרבה פעמים יורשים את הבעיות האלו בצורות שונות, אך זה מאוד תלוי. אך גם אם השפה אינה פגיעה להרצת קוד באמצעות Format String, אין זה אומר שהיא לא פגיעה לבעיה זו ולצורות ניצול אחרות שיכולות להיות יחודיות לאותה שפה, כלי או טכנולוגיה.
קוד בעייתי של Format String נראה כך:

```
...
char * some_variable;
...
printf(“Hello %s” + some_variable, “world”);
...
```

הבעיה המוצגת כאן היא בשפת C וגורמת למשתמש להזין ערך ל- some_variable. לאחר הזנת ערך למשתנה, מבוצעת פעולת חיבור (concat) של המידע אל מחרוזת המכילה format string ואז ניתן להזריק בעזרת שפת Format String קוד שיוכל להשפיע על רצף ריצת התוכנית (Execution flow) ולקבל שליטה מלאה על התהליך.

Race Condition:

פירוש Race Condition בעברית הוא התנגשות משאבים (Resource Condition). התנגשות משאבים יכולה להיגרם ממגוון רחב מאוד של סיבות, אשר תלויות בהמון גורמים. בעיית התנגשות המשאבים בתוך תוכנה מתייחסת לכך שאותו משאב נמצא בשימוש של יותר מחלק קוד אחד בתוכנה, נמצאת בעיקר בתוכנה שהיא מרובת חוטים (multi threaded), אך זו אינה הסיבה היחידה בה ניתן למצוא התנגשויות שונות בין משאבים באותה תכנה או בכלל.

התקפה על משאבים יכולה להתבצע במצב דומה לזה שהוצג למעלה, בעקבת גריעת זיכרון מהמערכת כאשר יש יותר מתוכנה אחת שרוצה הרבה מאוד זיכרון, יותר מהזיכרון שמערכת ההפעלה מסוגלת להקצות. גם הצורך לקרוא קובץ נעול יכול לגרום לבעיה זו.

מערכת ההפעלה Microsoft Windows ידועה בכך שכאשר משהו פותח קובץ מסוים הוא ננעל ולא ניתן לפתוח אותו מחדש, אלא אם הקובץ מוגדר כפתוח עם שיתוף, בניגוד למערכות ה- unix בהן צריך לבקש במיוחד מהמערכת לנעול קובץ שכזה או חלק מסוים של זיכרון הממופה עבור תכנה או מספר תכנות. התנגשויות משאבים ניתן למצוא גם במסדי נתונים במידה ועובדים לא נכון, אז מגלים כי בזמן שצד אחד מנסה לכתוב מידע, צד שני גורם ל- dead lock בנסיון לקרוא את אותו המידע בדיוק.

מיתוסים והנחות

לא מעט מבעיות האבטחה והבאגים הנמצאים בתוכנה נוצרים עקב חוסר מעקב או התעלמות של המתכנתים מהודעות המהדר או המפרשים. מעבר לכך, מתכנתים רבים חושבים שאם הקוד שלהם מפורש או מהודר, אז הוא לא מכיל בעיות כלשהן או שלפחות את חלקן ניתן לנצל נגד התכנה או המחשב המריץ אותם.
ניתן להבהיר כמה מבעיות אלו בנקודות הבאות:

<u>מיתוסים:</u>

- **אבטחה בהסתרה (security by obscurity)** – אם אף אחד לא יודע על בעיה, לא ניתן לנצל אותה.
- **שפת תכנות בטוחה** – שפות עיליות רבות מספקות את ההרגשה שהן נקיות וחפות מבעיות אבטחה, כגון אלו שניתן למצוא בשפות נמוכות כדוגמת שפת C. חושבים שבעיות אלו חפות מגלישת חוצצים ומעוד הרבה בעיות אבטחה ובאגים הקיימים בעולם.
- **סיסמאות מעורבלות בצורה חד כיוונית** – קבצים למשל המכילים סיסמאות המעורבלות באופן חד כיווני (hashing). התוקפים אינם יכולים לשחזר את הססמה לכן הם יקראו את המידע המעורבל וישתמשו בו כססמה עצמה.
- **שום דבר לא יכול לשבור את התוכנה**
- **ניתן לתקן ולפתור בעיות "תוך כדי תנועה"**

<u>הנחות:</u>

- צוות ה- QA יצליח לאתר בעיות ולתקן אותן
- המשתמש לא יזיק למידע או לתכנה
- התכנה תהיה רק בשימוש המתאים ליעד המקורי שלה
- קוד מהודר לשפת מכונה אינו ניתן לפירוש
- קידוד של סמלים בשפת מכונה מהווה סוג של הגנה

בחלק הבא של המאמר אסביר לעומק את כל הסעיפים הללו ואנסה להבהיר את הבעיות שצוינו בסעיף במיתוסים והנחות.

סיכום

בחלק זה הוצגו מעט בעיות המתרחשות בתכנות וגורמות לבעיות רבות בעת שחרור התכנה. ההבנה המרכזית היא שהבעיה העיקרית היא בראש ובראשונה גישה, והגישה הרגילה בה מלמדים לתכנת יכולה לגרום לנזקים רבים. בחלק השני נבין כיצד לשנות את הגישה בזמן שכותבים ומתכננים קוד ובכך להימנע מאותן לבעיות להכנס. ננסה להתחיל לחשוב קצת אחרת לגבי נקודות מבט שונות בפיתוח בכדי להגיע למצב בו נוכל לשלוט בכמות הבעיות שאנחנו מייצרים לעצמנו.