

המרכז האקדמי לב

המחלקה למדעי המחשב

דו"ח סופי פרויקט באבטחת תוכנה

**Secure an Editor**

שבט תשע"ו

01.2016

מגיש: יעקב כהן

ת.ז. **302845185**

שנה: ג'

חוג: הנדסת תוכנה

בהנחיית מר אריה הנל

תוכן עניינים

[תודות 3](#_Toc440458211)

[Abstract 4](#_Toc440458212)

[תקציר 5](#_Toc440458213)

[מבוא 6](#_Toc440458214)

[רקע תאורטי וטכני 7](#_Toc440458215)

[הסבר קצר על entry point ב- DLL 7](#_Toc440458216)

[DLL INJECTION 8](#_Toc440458217)

[AppInit\_DLLs registry key 8](#_Toc440458218)

[בדיקת היתכנות באמצעות התוכנה dependency walker 9](#_Toc440458219)

[אופן השימוש 9](#_Toc440458220)

[דוגמא מתוך הקוד 10](#_Toc440458221)

[CreateRemoteThread 11](#_Toc440458222)

[דוגמא מתוך הקוד 12](#_Toc440458223)

[API HOOKING 13](#_Toc440458224)

[API/Function Hooking/Interception Using JMP Instruction 13](#_Toc440458225)

[חישוב כתובת הקפיצה וקבלת ארגומנטים 15](#_Toc440458226)

[דוגמת קוד לביצוע Hooking על הפונקציה WriteFile 15](#_Toc440458227)

[בעיית העבודה ב-Multithreading כאשר מבצעים API hooking 16](#_Toc440458228)

[הצפנה 17](#_Toc440458229)

[ניהול היררכית המפתחות 18](#_Toc440458230)

[מבנה הקובץ המוצפן 19](#_Toc440458231)

[מימוש ניהול היררכית המפתחות 20](#_Toc440458232)

[תהליך ההצפנה 21](#_Toc440458233)

[קריאה מקובץ 23](#_Toc440458234)

[כתיבת קובץ חדש 24](#_Toc440458235)

[עריכת קובץ 25](#_Toc440458236)

[שימוש באפליקציה 26](#_Toc440458237)

# תודות

* בראש ובראשונה אני מודה לבורא עולם על שסייע בעדי לסיים את הפרויקט!
* תודה מיוחדת למנחה - מר אריה הנל, על שליווה את כל התהליך ביחס אישי, נדיבות וסבלנות לאורך כל הדרך! עד לסיום הפרויקט. על הרצון והנכונות לעזור ולענות על שאלות בכל עת על אף העומס.
* תודה מיוחדת לד"ר אריה טיטלבואם שייעץ ועזר בהמון שעות מזמנו.
* תודה מיוחדת לחברי חיים נפחא שעזר בכל מה שיכל לאורך כל הדרך.

# Abstract

The aim of the project is to adding security capabilities to an existing program that allows keeping the simplicity of using that program and in addition encrypt all documents generated by it.

Secure an Editor

The project aims to produce an extension to an existing product without changing the PE file, the product will be transparent to the user but will add option of encrypt files (possibly even without the user's knowledge), In order to achieve the goal the project will produce a DLL, then using techniques of DLL Injection to inject the DLL into a running process and API Hooking for the purpose of monitoring system functions such as writing, reading and opening a file and then calling functions from the DLL for encrypt or decrypt the file before reading or writing it.

In order to allow the reading of the document by a third party (could be sending a document to another user, a demand of IT staff in the company or even as a backup), The program will support in hierarchy of keys to allow more than one entity to decrypt the file.

All work on the project was on notepad ++ text editor 64 bit edition, this is a very common open source text editor, used by end users of all kinds.

Most thought in the project was a practical product to require as little user intervention to “Keep it simple” without the requirement of multiple passwords and so on.

# 

# 

# תקציר

מטרת הפרויקט היינה הוספת יכולות אבטחה למוצר קיים המאפשרת שמירה על פשטות השימוש של המוצר המוכר בתוספת אבטחת כל המסמכים המופקים מהתוכנית ע"י הצפנה.

Secure an Editor

מטרת הפרויקט היא לייצר הרחבה למוצר קיים ללא שינוי של קובץ ה - EXE, מוצר שיהיה כמה שיותר שקוף למשתמש אך יגרום להוספת אפשרות של הצפנה לקבצים (ייתכן אפילו ללא ידיעת המשתמש), לצורך כך תוצר הפרויקט הוא DLL שבטכניקות של DLL Injection מוחדר לתוך תהליך שרץ ו - API Hooking לצורך מעקב אחרי פונקציות מערכת כמו כתיבה, קריאה ופתיחת קובץ וקריאה לפונקציות מתוך ה-DLL.

ע"מ לאפשר קריאה של המסמך על ידי צד שלישי (יכול להיות שליחת מסמך למשתמש אחר, דרישה של צוות IT בחברה או גם לצורך גיבוי) השתמשנו בהיררכיית מפתחות כדי שיהיה יותר מגורם אחד שיוכל לפענח את הקובץ.

כל העבודה על הפרויקט בוצעה על עורך הטקסט notepad++ בגירסת 64 בית, זהו עורך טקסט בגרסת קוד פתוח נפוץ מאוד, בשימוש של משתמשי קצה מכל הסוגים.

עיקר החשיבה בפרויקט הייתה להביא מוצר פרקטי שידרוש כמה שפחות התערבות של המשתמש Keep it simple ללא דרישה של סיסמאות מרובות וכו'.

# 

# מבוא

נתאר תרחיש בו חברה גדולה משתמשת במוצר אשר אינו מפותח ואינו מתוחזק יותר ע"י המפתחים, כך שלא נוכל לקבל אפשרות לתמיכה ותחזוקה על ידי המפתחים המקוריים והחברה רוצה להוסיף למוצר יכולת להצפנה של כל המסמכים המופקים ממנו כך שלא יהיה ניתן לגשת מתוך עורכי טקסט אחרים (לא מאובטחים) אלא רק באמצעות התוכנה שעברה הרחבה כך שתתמוך בהצפנה ופענוח הקובץ או שייתכן מקרה שהחברה רוצה לשמור על סודיות של כל המסמכים שמפיקות תוכנות מסויימות או אפילו כלל התוכנות שרצות על המכונה.

בנוסף נרצה שקובץ שעבר הצפנה על ידי התוכנה המורחבת יוכל להיקרא על ידי משתמשים מורשים נוספים באמצעות תוכנה מורחבת מקבילה או דרישה מארגון IT שכל הקבצים המוצפנים יוכלו להיות מפוענחים ולהיקרא על ידי ארגון ה- IT לצורך שחזור או בקרה על הקבצים במערכת.

הפרויקט מנסה לתת מענה להרחבה של מוצר קיים ללא שינוי נוסף ככה שגם המשתמש הפשוט יוכל להשתמש בתוכנה הישנה ללא שינוי של ממש וללא סרבול מיותר ובעצם ללא הרגשה של שינוי בשימוש היום יומי שלו.

# 

# רקע תאורטי וטכני

## הסבר קצר על entry point ב- DLL

ל - dll יש אפשרות להוסיף entry point - מה קורה כאשר פעולות מסוימות מבוצעות על ה dll, יתכן אפשרות להוסיף פעולות מסוימות ללא קשר לפונקציונאליות שה DLL מייצא, אלא רק בהקשר של פעולה שמבוצעת על ה-DLL ולכן גם כאן קיימת פונקציית main:

INT APIENTRY DllMain(HMODULE hDLL, DWORD Reason, LPVOID Reserved)

{

switch (Reason)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_PROCESS\_DETACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

break;

}

return TRUE;

}

פונקציית הmain מקבלת 3 פרמטרים:

* hDLL - מצביע ל-base address של ה- Dll.
* Reason - הסיבה שהייתה קריאה לפונקציית הmain, קיימות 4 סיבות
  + DLL\_PROCESS\_ATTACH - במקרה של טעינה של ה-DLL למרחב הזיכרון של תהליך או ע"י פניה אליו בטבלת IAT או ע"י קריאה לפונקציה LoadLibrary.
  + DLL\_PROCESS\_DETACH - במקרה של הסרה של ה-DLL ממרחב הזיכרון של תהליך או ע"י סיום התהליך או ע"י קריאה לפונקציה FreeLibrary.
  + DLL\_THRED\_ATTACH - במקרה של יצירה של תהליכון חדש ע"י התהליך, כאשר זה קורה מערכת ההפעלה קוראת לכל הentry point של ה- dll שקשורים לתהליכון החדש.
  + DLL\_THRED\_DETACH - במקרה של סיום תהליכון, כאשר זה קורה מערכת ההפעלה קוראת לכל הentry point של ה- dll שקשורים לתהליכון שמסתיים.
* Reserved - משתנה שמור שמתנה לפי סוג הספריה (דינאמית או סטטית) וגם לפי המשתנה Reason.

הפונקציה אמורה להחזיר true כאשר היא מצליחה.

# 

# 

# DLL INJECTION

כל קוד שאנחנו מריצים נטען למרחב זמן הריצה מקובץ ה- EXE ע"י מערכת ההפעלה, ולכן אם ברצוננו לכתוב קוד משלנו ולהריץ אותו בתוך תהליך קיים נצטרך להכניס למרחב זיכרון זמן הריצה קוד (במקרה הזה dll) ולקרוא לו מתוך התהליך שרץ, כדי לאפשר שימוש מירבי ב - dll שכתבנו (כלומר שנוכל להשתמש בו בכל תוכנית שרצה במערכת) עם מינימום שינויים ( ללא צורך כתיבת קובץ הרצה נוסף וכו').

קיימות מספר שיטות לבצע dll injection במהלך הפרויקט השתמשנו ב-2 שיטות לכל שיטה יתרונות וחסרונות משלה:

* AppInit\_DLLs registry key
* CreateRemoteThread

## AppInit\_DLLs registry key

נעשה שימוש בשיטה שתגרום לdll injection בכל תהליך במערכת שמריץ את הספריה user32 (כמעט כל התהליכים טוענים את הספריה הזו) באמצעות אפשרות שמערכת ההפעלה windows מספקת ע"י registry key בשם AppInit\_DLLs.

יש לשים לב: כי רק למשמש עם הרשאות של מנהל יש אפשרות לבצע את השינוי במפתחות המערכת אבל מכיוון שמדובר בתוכנה לגיטימית (ולא בפריצה) הנחתי שבשביל האבטחה צריך לאפשר רק משתמש עם הרשאות admin להחליט אם הוא מאפשר ל - dll לרוץ על המכונה, ובנוסף לעומת שיטות אחרות ל- Dll Injection דבר זה חוסך מכתיבה של קובץ הרצה נוסף שיגרום ל-משתמש להבין כי יש כאן תוכנה נוספת שרצה ברקע.

משפט נחמד מתוך מסמך של Microsoft על האפשרות הזאת שמצאתי:

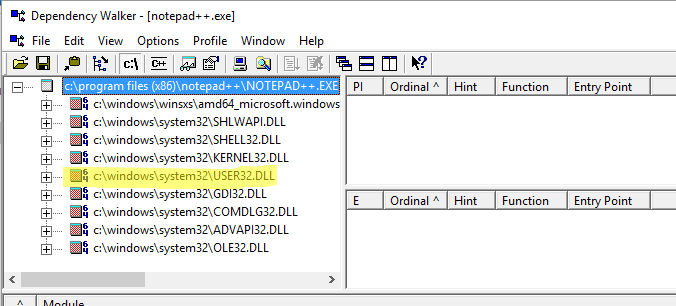
"Today, only a small set of legitimate applications use this mechanism. Unfortunately, a larger set of malware use this mechanism[[1]](#footnote-1). Applications and malicious software both use AppInit DLLs for the same basic reason, which is to hook APIs”[[2]](#footnote-2)

אל דאגה אנחנו מבצעים בה שימוש לגיטימי :)

יש אפשרות לאפשר רק ל-dll חתומים להיות מוזרקים לתוך תהליכים רצים על ידי האפשרות הזאת אבל זה אופציונאלי ונתון לבחירת המשתמש.

### בדיקת היתכנות באמצעות התוכנה dependency walker

השתמשתי בתוכנה בשם [[3]](#footnote-3)dependency walker כדי להראות שעורך הטקסט שעליו אני רוצה לבצע את ההרחבה אכן טוען את הספריה USER32.DLL כדי שיהיה ניתן להשתמש באפשרות של dll injection בעמצעות ה registry.



איור 1: ספריות התלויות של notepad++.exe

### אופן השימוש

עבור 32 ביט DLL ומערכת הפעלה 32 ביט הנתיב ב - registry:

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit\_DLLs]

עבור 64 ביט DLL ומערכת הפעלה 64 ביט הנתיב ב - registry:

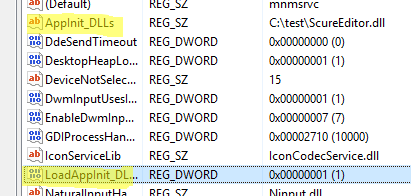
[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit\_DLLs]

עבור 32 ביט DLL ומערכת הפעלה 64 ביט הנתיב ב - registry:

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\AppInit\_DLLs]

לפי סוג ה Dll וסוג מערכת ההפעלה נוסיף למפתח AppInit\_DLLs את נתיב ה-Dll (עדיפות לנתיב ללא רווחים).

בנוסף קיים מפתח נוסף באותו מיקום בשם LoadAppInit\_DLLs שצריך לקבל את הערך 0x00000001 על מנת לאפשר טעינה של Dll לכל תהליך במערכת.



איור 2: מתוך ה registry manger של windows

מכיוון שה - DLL מוזרק לכל תהליך שירוץ במערכת נצטרך לבדוק שהתהליך שהוזרק לו ה - DLL זהו התהליך שאנחנו רוצים להרחיב (במקרה של הפרויקט notepad++...)

ולכן ב- entry point של ה - DLL נבדוק שהתהליך הוא התהליך שאנחנו רוצים שהDLL ירוץ בו ע"י בדיקה אם ה-PID תואם ל-PID של התהליך שאנו רצים בו (נוכל לקבל את ה PID של notepad++ ממערכת ההפעלה באמצעות שמו), אחרת נסיים את פעולת ה - DLL.

### דוגמא מתוך הקוד

ProgramToHook.push\_back(L"notepad++.exe");

if (!StartHooking(ProgramToHook, GetCurrentProcessId()))

{

return 0;

//exit from the program

}

//check if the Attach Process is in the List of Program To Hook list

bool StartHooking(std::vector<std::wstring> ProgramToHook, DWORD CurrentProcessId)

{

for each (std::wstring var in ProgramToHook)

{

if (GetProcessIdByName(var.c\_str()) == CurrentProcessId)

return true;

}

return false;

}

הסבר: מבצעים השוואה בין PID (מספר תהליך) של התהליך הנוכחי לבין מספר התהליך שנקבל ממערכת ההפעלה לפי שם התהליך, מכיוון שרציתי שתהיה אפשרות הרחבה לכמה תהליכים, נוכל להכניס מספר שמות של תהליכים, ונבדוק אם ה PID של אחד מהם שווה לPID הנוכחי נוכל להמשיך לשלב ה- api hooking.

## [[4]](#footnote-4)CreateRemoteThread

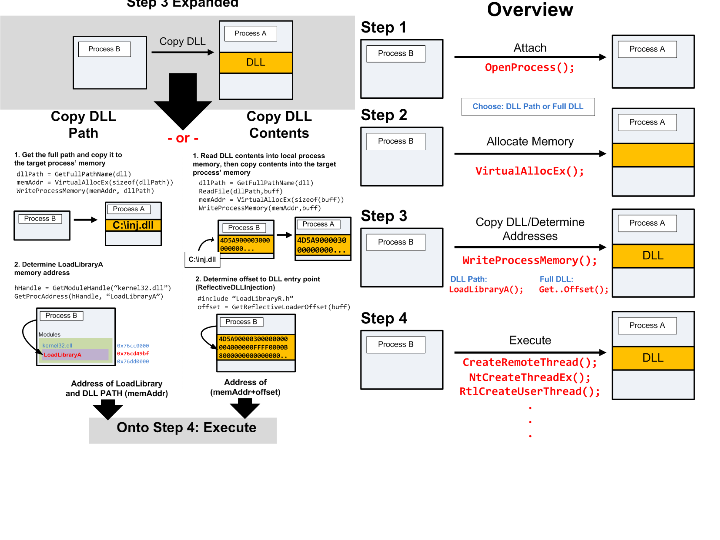
שיטה נוספת לגרום ל-Dll injection היא ע"י שנגרום לתהליך המקורי ליצור thread חדש שיטען את ה DLL למרחב זמן הריצה של התהליך.

היתרונות בשיטה הזאת היא שלא נדרש משתמש עם הרשאות Admin כדי להגדיר למערכת לבצע את פעולת ה - injection אך מצד שני יש כאן פחות שקיפות למשתמש שהוא נדרש להפעיל את התוכנה מתוך קובץ הרצה חיצוני ובעצם להחליט אם הוא רוצה להשתמש ביכולות הצפנה או לא.

השלבים בתהליך:

1. נקבל מצביע לתהליך שאליו אנחנו רוצים לבצע dll injection.
2. בתוך התהליך נבצע הקצאת זכרון חדשה בגודל נתיב הdll שאנחנו רוצים להזריק למערכת.
3. לזכרון החדש המוקצה נבצע כתיבה של הנתיב ונקבל את הכתובת לפונקציה loadLibrary
4. בעזרת הפונקציה CreateRemoteThread אנו גורמים לתהליך המקורי להריץ thread חדש שיפעיל את הפונקציה loadLibrary עם הפרמטר של הdll ומכיוון שתהליכון הוא חלק מהתהליך ביצענו Dll injection לתהליך.

תרשים לדוגמא[[5]](#footnote-5)



איור 3: תהליך dll injection בשיטת CreateRemoteThread

יש לשים לב כי בשיטה של CreateRemoteThread צריך לדאוג שהתהליך שאליו אנחנו רוצים את ה- dll injection נמצא כבר בריצה, אבל בתוכנית שלנו יש דרישה נוספת שהתהליך יבצע dll injection לפני שהתוכנית פותחת את הקבצים באופן אוטומטי, כדי שהמשתמש לא יצטרך לפתוח את הקבצים כל פעם באופן ידני, לכן נייצר תוכנית (launcher) שיגרום להרצת התהליך שאנחנו רוצים להריץ (בפרויקט שלנו לדוגמא: notepad++) במצב מושעה, נבצע dll injection ואז נעיר את התהליך שיתחיל את כל תהליך ה-Api Hooking.

### דוגמא מתוך הקוד

//pe32.th32ProcessID contain the notepad++ PID

//get handle to the notepade++ process

HANDLE hProcess = OpenProcess(PROCESS\_CREATE\_THREAD | PROCESS\_VM\_OPERATION |

PROCESS\_VM\_WRITE, FALSE, pe32.th32ProcessID);

//get pointer the LoadLibraryW function

LPVOID LoadLibraryAddr = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"),

"LoadLibraryW");

//Full Path contain the dll path

int len = wcslen(FullPath);

//memory allocation for the dll path

LPVOID LLParam = (LPVOID)VirtualAllocEx(hProcess, NULL, len,

MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

//write the path to process len\*2 due to the size of WCHAR (2 bytes)

WriteProcessMemory(hProcess, LLParam, FullPath, len\*2, NULL);

//the start off the new thread is LoadLibraryW, the path passed as the parameter

CreateRemoteThread(hProcess, NULL, NULL, (LPTHREAD\_START\_ROUTINE)LoadLibraryAddr,

LLParam, NULL, NULL);

# API HOOKING

כדי לקבל שקיפות מול המשתמש ושליטה מלאה על הקריאה והכתיבה לקובץ לצורך הצפנה ופיענוח של טקסט, נצטרך לקבל שליטה על פונקציות הקריאה(ReadFile) והכתיבה לקובץ (WriteFile) הבסיסיות ביותר שמערכת ההפעלה מספקת, כך שאם המשתמש ירצה לכתוב או לקרוא קובץ הוא יהיה חייב לקרוא לפונקציות הללו וככה נוכל לתפוס את כל הקריאות והכתיבות מקבצים שהאפליקציה מבצעת

.

לדוגמא בקריאה מקובץ במקום לקרוא מהקובץ נרצה שיתבצעו הפעולות הבאות:

Read file

Encrypted file

MyReadFile()

ReadFile()

Read encrypted file

Decrypt file

Clear Text

ReadFile()

איור 4: תרשים כללי של Api Hooking

נרצה שבכל קריאה לפונקציית המערכת Read File קודם תתבצע קריאה לפונקציה My Read File שתקרא את הקובץ ובנוסף תדאג לפענוח הקובץ.

נשתמש בApi Hooking כדי לבצע את הפעולה הזאת.

## API/Function Hooking/Interception Using JMP Instruction[[6]](#footnote-6)

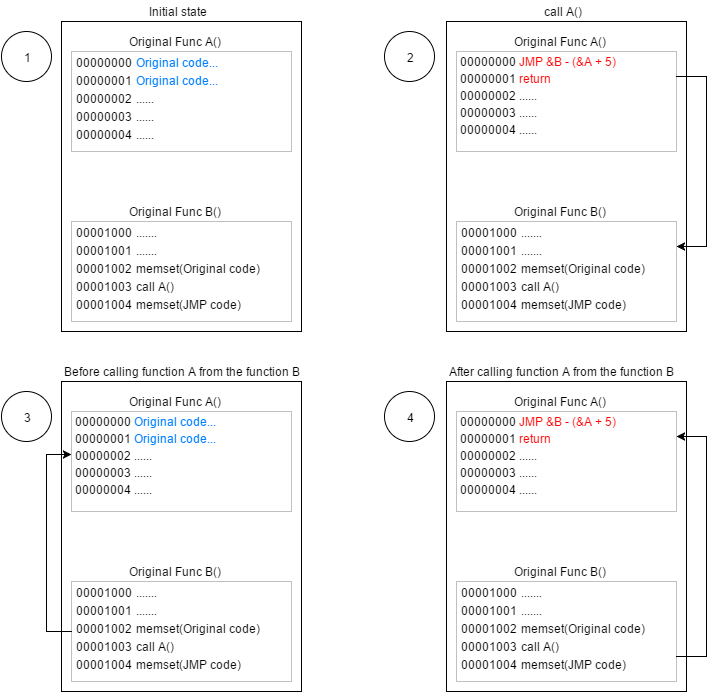
אנחנו נשכתב את 6 bytes הראשונים של הפונקציה המקורית 5 Bytes כדי לקפוץ לפונקציה שכתבנו ועוד Byte אחד כדי לכתוב פעולת return – בצורה שהתבצע הhooking בפרויקט אין צורך בreturn.

כאשר תתבצע קריאה לפונקציה שעליה אנחנו מבצעים hooking:

* קריאה לפונקציה שעליה מבצעים hooking.
* קפיצה לפונקציה אחרת בגלל פקודת הjump.
* בסיום הפונקציה תתבצע קפיצה חזרה לפונקציה המקורית.

נשמור את 6 הבתים הראשונים של הפונקציה עליה אנחנו מבצעים hooking אם נרצה לחזור ולהשתמש בפונקציה המקורית.

תרשים לדוגמא:



איור 5: תרשים לדוגמא לביצוע hooking

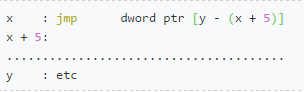
תרשים 1 - מייצג את המצב ההתחלתי של התהליך אנחנו רוצים לבצע hooking על הפונקציה A.

תרשים 2 - לאחר שינוי הפקודה הראשונה, כל קריאה לפונקציה A תבצע קפיצה לפונקציה B קודם ולאחר סיום הפונקציה B יתבצע חזרה למקום הקריאה בפונקציה A ששם נמצאת הפקודה return aשתגרום חזרה למקום הקריאה לפונקציה A.

תרשימים 3 ו-4 - כאשר נרצה מתוך הפונקציה B לקרוא לפונקציה המקורית A, מכיוון ששינינו את הפונקציה A אנחנו שוב נגיע לפונקציה B, לכן קודם נשחזר את הפקודה הראשונה לפונקציה A ואז נבצע קריאה לפונקציה המקורית A, כאשר נסיים את הפעולה ב-A ונחזור לפקודה אחרי הקריאה לפונקציה A בפונקציה B נבצע שוב hooking על הפונקציה A כדי שבקריאה הבאה לפונקציה A שוב נגיע לפונקציה B.

### חישוב כתובת הקפיצה וקבלת ארגומנטים

מרחק הקפיצה = כתובת הפונקציה שאליה אנחנו רוצים לקפוץ - (כתובת הפונקציה הישנה + 5)[[7]](#footnote-7)



איור 6: דוגמא לחישוב כתובת הקפיצה

מכיוון שאין שינוי במחסנית, הפונקציה שאיתה אנחנו מבצעים hooking תקבל את כל הארגומנטים באותו הסדר ולכן על מנת לקבל את כל הארגומנטים עלינו לכתוב פונקציה שמקבלת את אותו סוג ארגומנטים ובאותו הסדר.

## 

## דוגמת קוד לביצוע Hooking על הפונקציה [[8]](#footnote-8)WriteFile

pOrigWFAddress = (pWriteFile)GetProcAddress(GetModuleHandle(L"kernel32.dll"), "WriteFile");

// get address of original Write File

BYTE tempJMP[SIZE] = { 0xE9, 0x90, 0x90, 0x90, 0x90, 0xC3 };// 0xE9 = JMP 0x90 = NOP 0xC3 = RET

memcpy(JMPWF, tempJMP, SIZE); // store original jmp instruction to JMP

DWORD JMPSize = ((DWORD)newFunction - (DWORD)pOrigWFAddress - 5);// calculate jump distance

VirtualProtect((LPVOID)pOrigWFAddress, SIZE, // assign read write protection

PAGE\_EXECUTE\_READWRITE, &oldProtectWF);

memcpy(oldBytesWF, pOrigWFAddress, SIZE); // make backup

memcpy(&JMPWF[1], &JMPSize, 4); // fill the nop's with the jump distance (JMP,distance(4bytes),RET)

memcpy(pOrigWFAddress, JMPWF, SIZE);// set jump instruction at the beginning of the original function

VirtualProtect((LPVOID)pOrigWFAddress, SIZE, oldProtectWF, NULL); // reset protection

הסבר:

בשורה הראשונה נקבל את הכתובת המקורית של הפונקציה writeFile,

נעתיק את 6 הבתים הראשונים של הפונקציה ונשמור את זה לצורך שחזור עתידי כאשר נרצה לקרוא לפונקציה המקורית נצטרך לשחזר את הפונקציה המקורית קודם.

ונשכתב את 6 הבתים הראשונים לפקודת ה JMP למקום החדש.

## בעיית העבודה ב-Multithreading כאשר מבצעים API hooking

עורכי טקסט רבים כדי לייעל את זמן פתיחת הקבצים פותחים אותם בצורה מקבילית וכן אותו הדבר בשמירה של מספר קבצים במקביל, דבר זה יוצר בעיה לתכנון שיצרנו, יתכנו קריאות נוספות לפונקציות שביצענו עליהם hooking בצורה מקבילית בזמן ששחזרנו את הפונקציה המקורית, ולכן ייתכן שמספר קריאות לפונקציה לא יגיעו לפונקציה שלנו אלא ישירות לפונקציה המקורית.

לכן עלינו לדאוג שכאשר מישהו מבצע ביטול של ה- hooking תהליכים אחרים לא יוכלו להתקדם ולקרוא לאותה פונקציה.

ובנוסף קיימים מספר משתנים גלובליים שרציתי למנוע גישה אליהם במקביל כדי למנוע race condition (בעיית reader writer).

לכן בתחילת הפונקציה מנעתי גישה מקבילית לפונקציות הקריאה והכתיבה ולכן כל תהליכון שביצע ביטול את ה- Hooking נועל את הקטע הקריטי עד שהוא משנה בחזרה למצב של hooking וגם לא ייתכן מצב של גישה במקביל למשתנים גלובליים.

בנוסף ייתכן מצב של deadlock כאשר תהליך אחד מייד יבקש להיכנס שוב לקטע הקריטי ותהליך אחר כבר ידרוש אותו ומצד שני אותו התהליך לא יכול להיכנס בגלל גישה אחרת, לכן השתמשתי ב -mutex אחר (Recursive mutex[[9]](#footnote-9)) שנותן לבעלים שלו אפשרות להיכנס שוב בלי לחכות.

כדי להימנע מבעיית נעילה במקרה של exeption השתמשתי במחלקה [[10]](#footnote-10)lockguard שיודעת לשחרר mutex גם במקרה של exeption.

# הצפנה

מטרת הפרויקט היא בעצם ליצור מנגנון הצפנה לקבצים שמפיקים עורכי טקסט ומצד שני לשמור על פשטות השימוש של המשתמש.

הדרישות שרציתי שהמערכת תתמוך בהם:

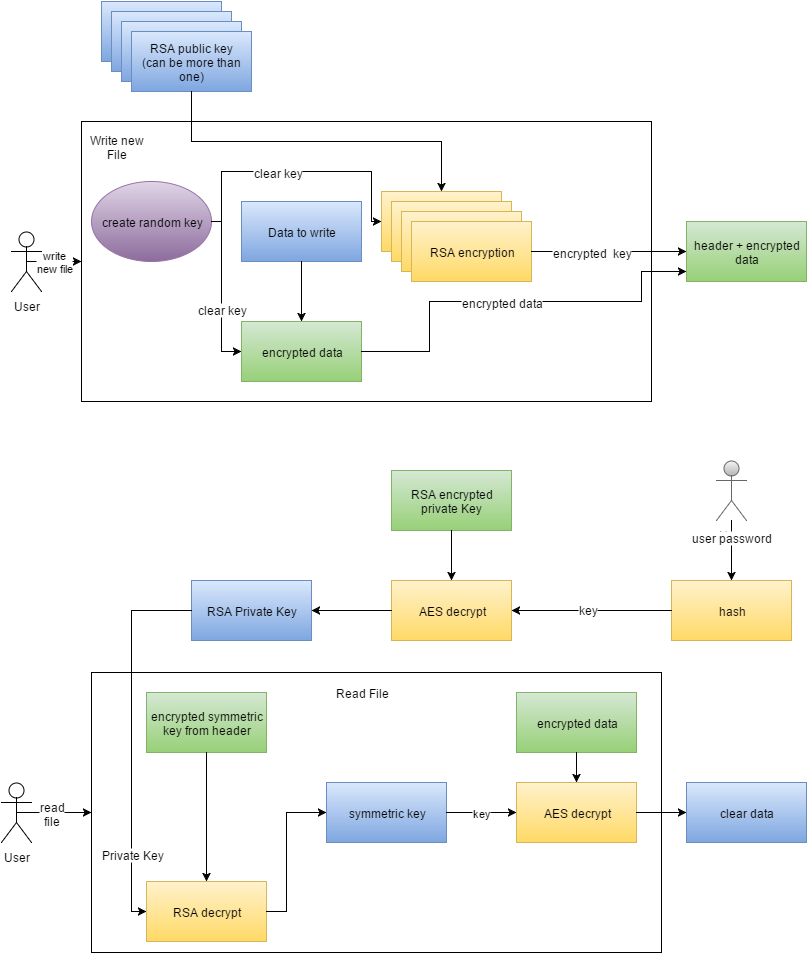
* הצפנת הקבצים תתבצע באמצעות מפתח רנדומאלי ויחודי לכל קובץ, שאם יגלו מפתח אחד לא יהיה ניתן לפענח את כל הקבצים.
* על מנת לשמור על פשטות השימוש נדרוש רק סיסמא אחת מהמשתמש בזמן הפעלת ה - Editor.
* ניתן יהיה לבצע שליחה של מסמכים מוצפנים בין 2 משתמשים או לחילופין ניתן יהיה לתת לגורם מנהל (כמו מחלקת IT בארגון) אפשרות לפענח את כל הקבצים.

בכל הפרויקט מימוש ההצפנות (RSA ו- AES) נעשו באמצעות הספרייה [[11]](#footnote-11)mbedtls.

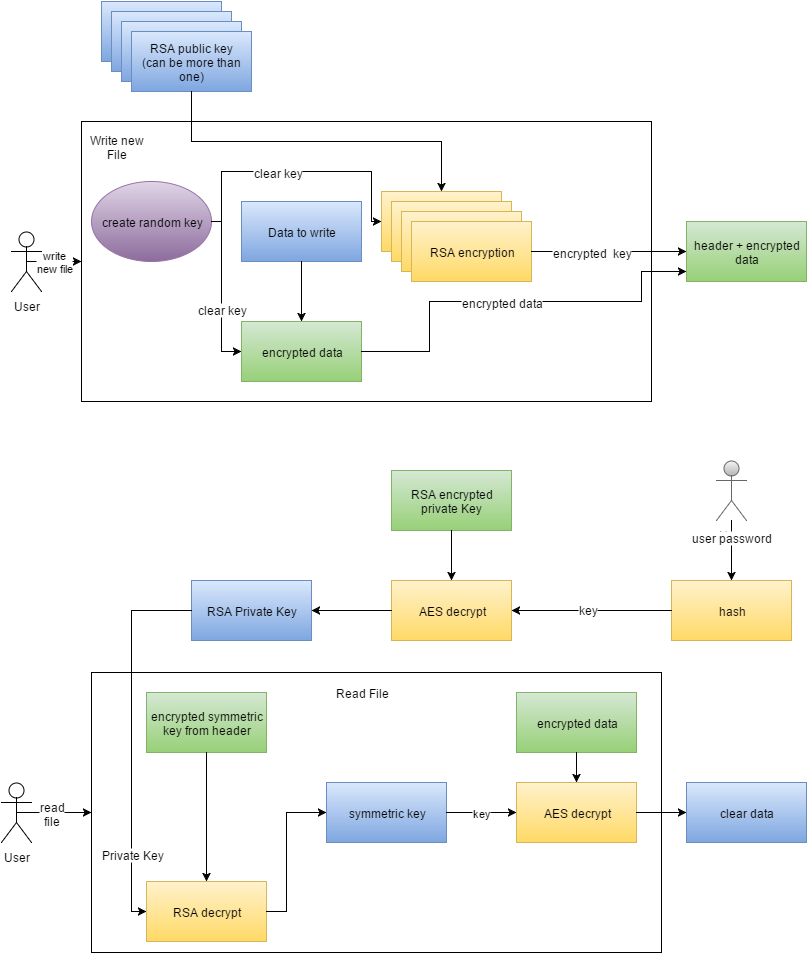
## ניהול היררכית המפתחות

כדי לעמוד בכלל דרישות המערכת נצטרך לנהל מערכת לשמירת מפתחות צורת שמירת המפתחות תתבצע בצורה הבאה:

1. לכל משתמש יהיו זוג מפתחות א-סימטרי ציבורי ופרטי.
2. המערכת תייצר מפתח סימטרי לכל קובץ (מכיוון שאין יעילות בהצפנת קבצים גדולים במפתחות א-סימטריים).
3. המערכת תצפין כל מפתח סימטרי באמצעות כלל המפתחות הציבוריים הא-סימטרים שקיימים (כדי לאפשר קריאה מאובטחת ע"י צד שלישי) ותשמור אותם בheader של הקובץ.
4. כאשר המשתמש ירצה לקרוא קובץ הוא יצטרך לפענח את המפתח הסימטרי באמצעות המפתח הפרטי ה-אסימטרי ובאמצעות המפתח הסימטרי לפענח את הקובץ.
5. המפתח הפרטי הא-סימטרי יוחזק מוצפן באמצעות סיסמת משתמש, בתחילת התוכנית באמצעות סיסמת המשתמש יפוענח המפתח הפרטי, כך שכלל הדרישות לסיסמא הם רק בתחילת התוכנית (עמידה ביעד פשטות השימוש).

תרשימים לדוגמא:

איור 7: סכמת כתיבת קובץ חדש



איור 8: סכמת הקריאה מקובץ ופענוח המפתח ה-אסימטרי בתחילת התוכנית

## 

## מבנה הקובץ המוצפן

בגלל הדרישות נדרשנו להוסיף Meta Data לקובץ, ולכן הקובץ המוצפן מכיל:

* header במבנה הבא:
  + 2 בתים - 0x12, 0x34 כדי לזהות שמדובר במבנה בקובץ שעבר הצפנה
  + 512 בתים - מפתח סימטרי לאלגוריתם AES מוצפן באמצעות מפתח ציבורי RSA 2048 בתים, נתון להרחבה לפי מספר המפתחות שברצוננו לשמור לכל מפתח נצטרך לשמור 512 בתים
  + 8 בתים -לשמירת הnonce מכיוון שהעבודה היא ב CTR mode נשמור nonce יחודי לכל קובץ
* הקובץ עצמו (data) מוצפן.

כלומר יש לקחת בחשבון שלכל קובץ יתווסף header של 10 byte ועוד 512 byte \* מספר הפעמים שנצפין את המפתח האקראי של הקובץ עם מפתחות ציבוריים שונים כאשר כל מפתח ציבורי שייך לאדם אחר שאני רוצה שיוכל לקרוא את הקובץ.

## 

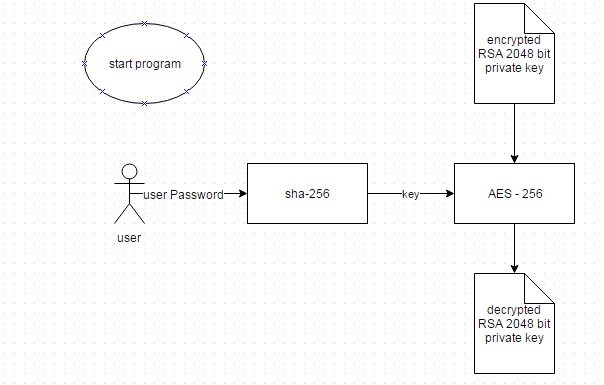
## 

## מימוש ניהול היררכית המפתחות

כדי לשמור על פשטות השימוש נדרוש מהמשתמש סיסמא שעליה נבצע פעולת hash באמצעות אלגוריתם sha2 - 256 bit פלט האלגוריתם ישמש כמפתח לאלגוריתם AES-256 CTR mode באמצעותה נפענח את מפתח ה-RSA הפרטי של המשתמש (כמובן שאם למשתמש אין מפתח כזה התהליך יצור לו אחד כזה)[[12]](#footnote-12).

כעיקרון נשתמש באלגוריתם הצפנה א-סימטרית כדי להצפין מפתחות לאלגוריתם סימטרי אבל להצפנת הקובץ עצמו נשתמש באלגוריתם סימטרי, מכיוון שלא מומלץ להשתמש במפתח אסימטרי להצפנת data מעל גודל המפתח (בעיות עיקריות הם איטיות, וחוסר אבטחה בגלל הצפנת בלוקים חוזרת באותו המפתח).

מכיוון שמדובר במפתח הפרטי בחרתי להשתמש באלגוריתם AES - 256 bit למרות שהשימוש בו איטי יותר ב-40%, אבל מכיוון שאנחנו רוצים אבטחה מירבית למפתח הפרטי בחרתי באלגוריתם שיתן את מספר האפשריות הרב ביותר שניתן ()[[13]](#footnote-13) למניעת מתקפת brute force יותר משאר האלגוריתמים של AES.



איור 7 : תרשים פענוח המפתח הפרטי בתחילת התוכנית

התוכנית תייצר מפתח אקראי לכל קובץ ותצפין אותו באמצעות רשימת המפתחות הציבוריים שנתונה לה, נשמור את המפתח האקראי המוצפן ב header של הקובץ.

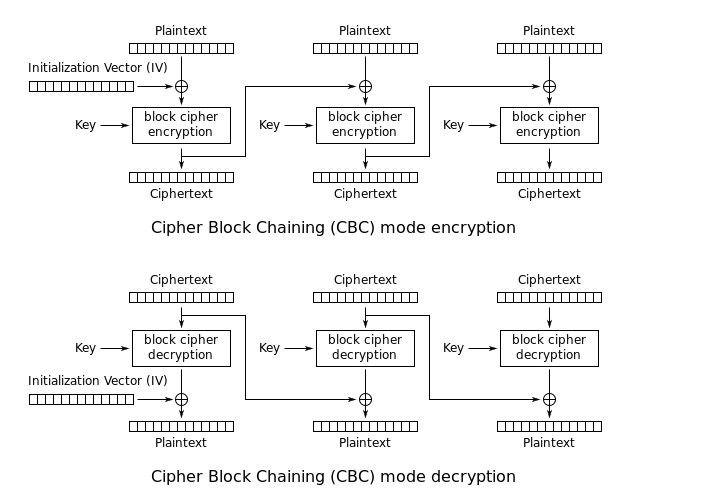
כאשר נקרא קובץ נשתמש במפתח הפרטי המתאים למפתח הציבורי שהצפין (צריך להיות לפחות אחד) כדי לקבל את המפתח הסימטרי האקראי שהשתמשנו בו לפענוח הקובץ המוצפן.

## תהליך ההצפנה

הפונקציות שלהם ביצענו Hooking הם הפונקציות ReadFile ו - WriteFile,

הפונקציות האלה מקבלות כל פעם buffer של מידע לכתוב לקובץ או שנקרא מהקובץ,

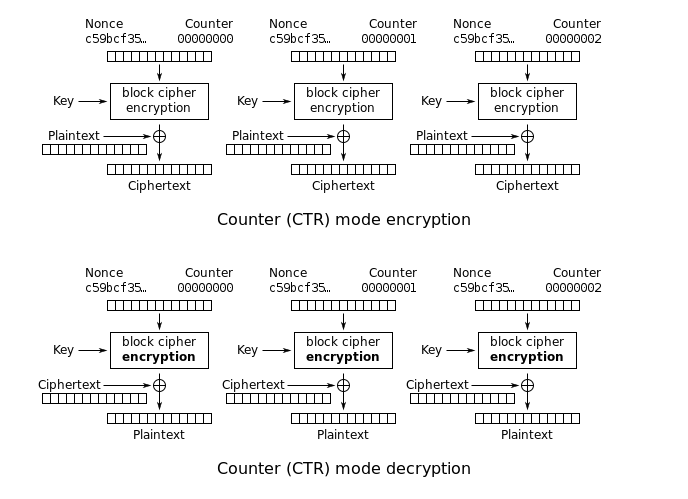
מכיוון שאין לנו את כל הקובץ אלא קריאה של הקובץ בבלוקים (גודל הבלוק תלוי במכונה ובמערכת ההפעלה) לפני ביצוע הפעולה להצפנה או לפענוח, לא נוכל להשתמש ב CBC mode, ההצפנה היא פחות בעיתית כי נוכל לקרוא את ה - Block האחרון שהצפנו ולהשתמש בו להמשך התהליך, אך מכיוון שכדי לפענח נצטרך את הblock המוצפן האחרון שקראנו מכיוון שהוא לא קיים כבר בקובץ כי כתבנו את הblock המפוענח לקובץ כבר, ולכן נצטרך לפענח את הכל בבת אחת.



איור 8: CBC mode

כמובן שב ECB mode לא נשתמש בגלל חוסר האבטחה שלו מכיוון שניתן לזהות בלוקים שווים.

ולכן ההצפנה שבחרתי להשתמש היא AES - 192 Bit[[14]](#footnote-14), הבחירה ב 192 bit היא כדי לשמור על יעילות (20% יותר מהיר מ 256 bit) אבל עדיין מספר האפשרויות הוא מספיק גדול )[[15]](#footnote-15)) והבחירה ב - CTR mode בגלל הפסילה של שאר ה-mode האחרים, ה - nonce הוא מספר אקראי שנשמר לא מוצפן ב header של הקובץ, בגודל 8 bytes כחלק מדרישות הmode דבר שעוזר למנוע זיהוי של block זהים, ואת ה - counter נוכל לחשב לפי מיקום המצביע בקובץ.

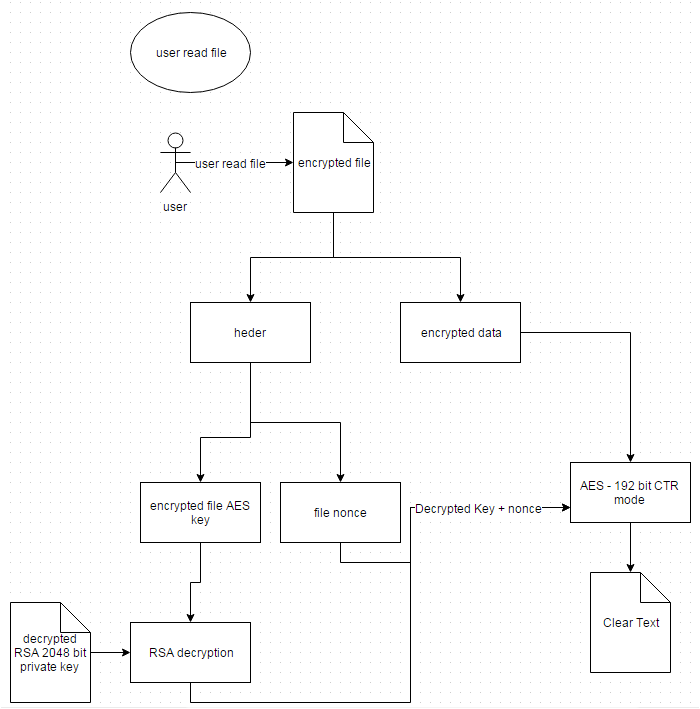


איור 9: CTR mode

### 

### קריאה מקובץ

כאשר נקרא מקובץ מוצפן נצטרך לקרוא את המפתח המוצפן והNonce מה- header של הקובץ, לפענח את המפתח על ידי המפתח RSA הפרטי של המשתמש, באמצעות המפתח שיתקבל וה nonce לפענח את המידע המוצפן בקובץ.



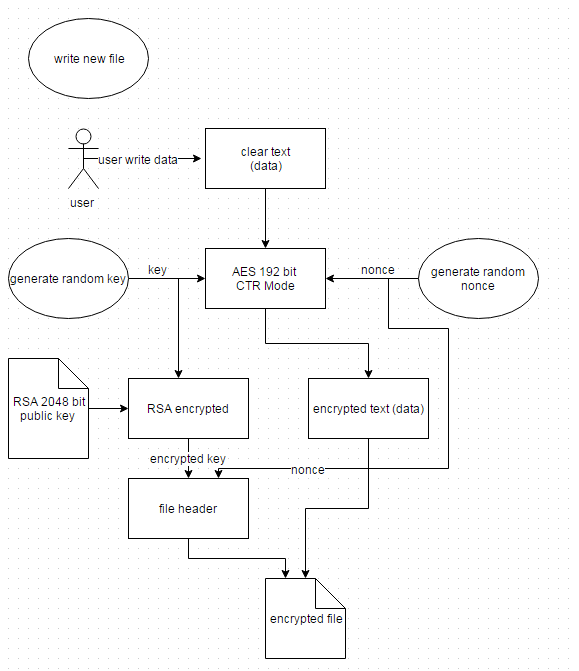
איור 10 : תרשים הזרימה של קריאת קובץ מוצפן

### 

### 

### כתיבת קובץ חדש

נייצר מפתח רנדומלי לאלגוריתם AES בגודל 24 Byte ו-nonce בגודל 8 byte, נצפין באמצעותם את הdata לאחר מכן נצפין את המפתח שיצרנו באמצעות המפתח הציבורי (יכול להיות יותר מאחד) ונשמור את הnonce ואת המפתח המוצפן ב - heder של הקובץ, בנוסף כמובן שנשמור גם את הdata המוצפן גם בתוך הקובץ



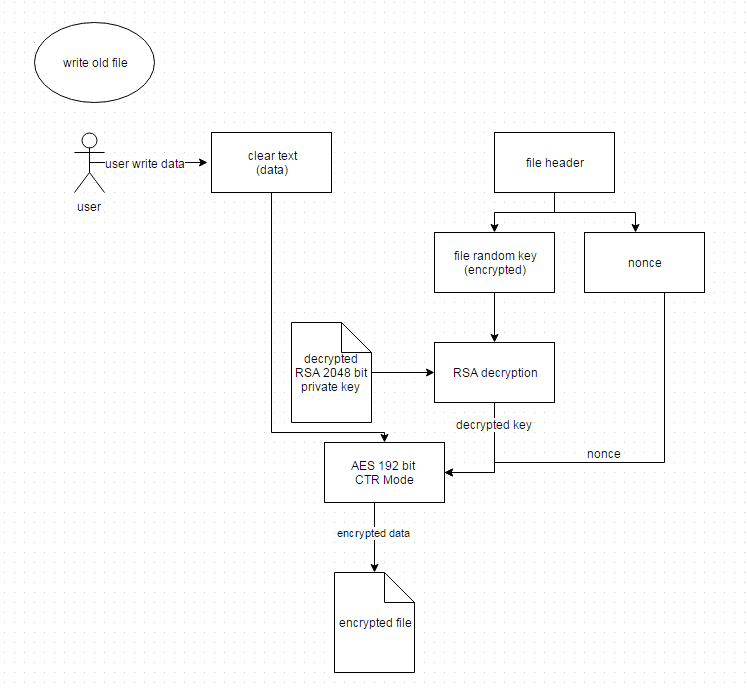
איור 11 : תרשים הזרימה של יצירת קובץ חדש

### 

### 

### עריכת קובץ

נקרא מה- hader של הקובץ את המפתח ראנדומלי לאלגוריתם AES בגודל 24 Byte (מוצפן) ו-nonce בגודל 8 byte, נפענח את המפתח ונצפין באמצעותם את הdata, נשמור גם את הdata המוצפן בתוך הקובץ.



איור 12 : תרשים הזרימה של עריכת קובץ

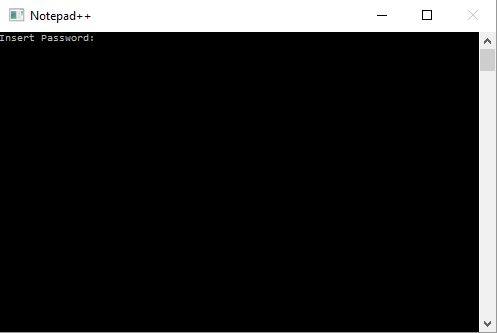
# 

# 

# שימוש באפליקציה

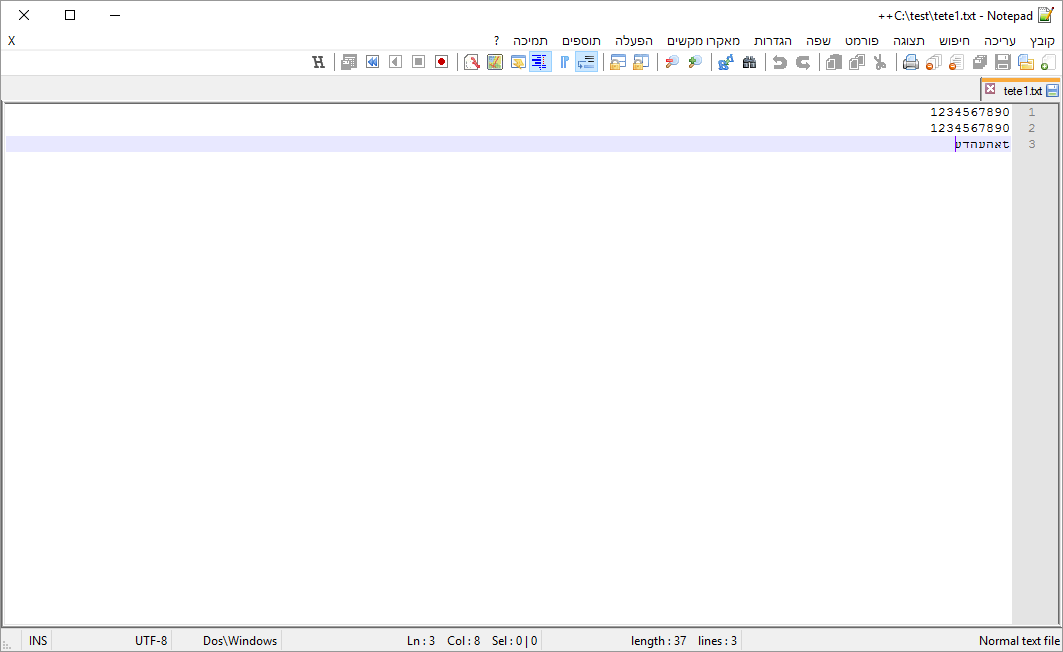
1. כפי שהוסבר בחלק של ה-DLL injection צריך להגדיר את הregistry key עם הנתיב של DLL הפרויקט ScureEditor.dll (נתיב ללא רווחים) במפתח AppInit\_DLLs ובמפתח LoadAppInit\_DLLs לשים את הערך 0x00000001.
2. או לחילופין ניתן להריץ את קובץ ההרצה SecureStartUp.exe המצורף הנמצא בתיקיית launcher (יש לשים לב כי גם Dll הפרויקט ScureEditor.dll נמצא באותה תיקייה).
3. האפליקציה תחפש אם קיימים זוג מפתחות RSA, אם לא האפליקציה תייצר זוג מפתחות כאלו[[16]](#footnote-16).

1. בפעם הראשונה הסיסמא תשמש כדי להצפין את קובץ הprivate שנוצר, ולאחר מכן כדי לפענח אותו בכל פעם שהתוכנית עולה.

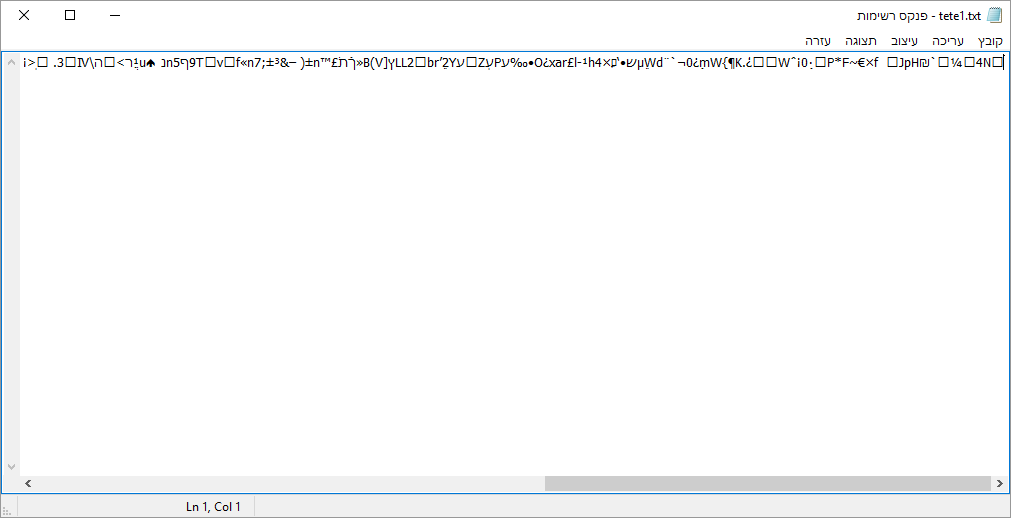


איור 13: הפעלת notepad++ כאשר ה-dll injection מופעל

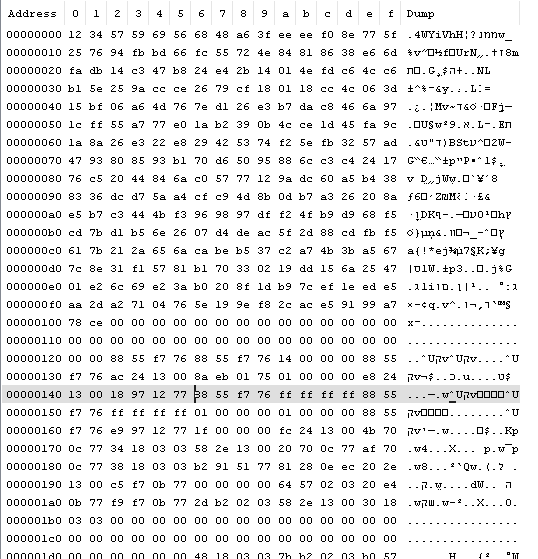
1. המשך שימוש בעורך הטקסט כרגיל :).



איור 14: כיצד הקובץ נראה בתוכנה notepad++



איור 15: כיצד הקובץ נראה בתוכנה notepad



איור 16: הקובץ המוצפן ב hax editor

הערה : לצורך הרחבה ומימוש נכון להגברת השימושיות עדיף יהיה להכניס פרטים כמו שמות של תהליכים שברצוננו להפעיל עליהם את ה-DLLinjection ומפתחות public נוספים בעזרת קובץ קונפיגורציה, מפאת חוסר זמן לא הספקתי להשלים את העבודה אבל זאת המחלקה config בפרויקט שתפקידה לקחת קובץ json ולשלוף ממנו נתונים.

1. בגלל מה שכתבנו שצריך הרשאות Admin בשביל להשתמש ב – AppInitDll, אז לכאורה כבר לנוזקה יש הרשאות Admin והיא יכולה להריץ כל קובץ EXE שהיא רוצה ולמה היא צריכה להשתמש בDll injection? ההסבר שלי הוא או שהנוזקה יעודית לאפליקציה מסויימת (כמו שעשינו בפרויקט הזה) או בשביל לשמור על סודיות ככה שביומן המערכת לא ירשם שום דבר חריג ויהיה יותר קשה לגלות שרץ קוד חריג במערכת (RootKit). [↑](#footnote-ref-1)
2. <http://download.microsoft.com/download/7/e/7/7e7662cf-cbea-470b-a97e-ce7ce0d98dc2/appinit_win7.docx> [↑](#footnote-ref-2)
3. <http://www.dependencywalker.com/> [↑](#footnote-ref-3)
4. מבוסס על המאמר:<http://www.codeproject.com/Articles/30140/API-Hooking-with-MS-Detours#DLLInject> [↑](#footnote-ref-4)
5. התרשים נלקח מהאתר: <http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html> [↑](#footnote-ref-5)
6. <https://en.wikipedia.org/wiki/Hooking#API.2FFunction_Hooking.2FInterception_Using_JMP_Instruction> [↑](#footnote-ref-6)
7. הסבר: מכיוון שגודל פקודת הjump הוא 5 Byte ,Byte אחד ל- OpCode ועוד 4 Bytes למרחק הקפיצה לאחר ביצוע פעולת הjmp הרגיסטר EIP יהיה 5 bytes לאחר תחילת הפונקציה שעליה אנחנו מבצעים hooking. [↑](#footnote-ref-7)
8. מבוסס על דוגמא מויקיפדיה: <https://en.wikipedia.org/wiki/Hooking#Sample_code> [↑](#footnote-ref-8)
9. <http://baptiste-wicht.com/posts/2012/04/c11-concurrency-tutorial-advanced-locking-and-condition-variables.html> [↑](#footnote-ref-9)
10. <http://en.cppreference.com/w/cpp/thread/lock_guard> [↑](#footnote-ref-10)
11. <https://tls.mbed.org/> [↑](#footnote-ref-11)
12. הקבצים יכתבו לתיקייה : C:\Users\user\AppData\Roaming\Notepad כברירת מחדל כדי לאפשר כתיבה של המפתחות גם user ולא רק ל-admin. [↑](#footnote-ref-12)
13. <https://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-size-and-parameters-report-2014> page number 23-24. [↑](#footnote-ref-13)
14. כמו כן היה ניתן להשתמש ב stream chiper באלגוריתמים כמו RC-4 (אלגוריתם שהוכח כבר כפרוץ לכן לא השתמשתי בו), אבל העדפתי להשתמש באלגוריתם יותר מוכר כמו AES שגם היה לו מימוש ב mbedtls. [↑](#footnote-ref-14)
15. מקור הנתונים עיין הערה מספר 10. [↑](#footnote-ref-15)
16. המימוש ליצירת מפתחות RSA נלקח מ-mbedtls:

    <https://tls.mbed.org/kb/cryptography/rsa-key-pair-generator>

    המפתחות נשמרים במבנה שספריית RSA של mbedtls יודעת לטעון אותם לתוך אובייקט. [↑](#footnote-ref-16)